ECS 3.8.x Data Access Guide

April 2024
Rev. 1.2

Contents

Notes, cautions, and warnings

NOTE: A NOTE indicates important information that helps you make better use of your product.

CAUTION: A CAUTION indicates either potential damage to hardware or loss of data and tells you how to avoid the problem.

WARNING: A WARNING indicates a potential for property damage, personal injury, or death.

1

S3

ECS supports the S3 API and the extension, this section provides information about authenticating with the service, and using the Software Development Kit (SDK) to develop clients to access the service.

Some aspects of bucket addressing and authentication are specific to ECS. To configure an existing application to talk to ECS, or develop a new application that uses the S3 API to talk to ECS, see the ECS Administration Guide https://www.dell.com/support/.

Topics:

Revision history

Table 1. Revision history

Revision Date	Description of change
December 2022	Rev 1.0 Initial release of ECS 3.8
April 2023	Rev 1.1 Updates to "CAS connection string" and "Limitations" on "Metadata search with Tokenization"
April 2024	Rev 1.2 ECS 3.8.1 updates

Amazon S3 API support in ECS

ECS supports the Amazon Simple Storage Service (Amazon S3) Application Programming Interface (API).

Table 2. S3 Object Service

Protocol	Ports
HTTP	9020
HTTPS	9021

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

Table 3. Supported S3 APIs

Feature	Notes

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

DELETE Bucket	ECS simplified bucket deletion request is initiated through this endpoint. For example: `DELETE /<bkt> -H "x-emc-empty-bucket:true"`
DELETE Bucket cors	-
DELETE Bucket life cycle	Only the expiration part is supported in life cycle. Policies that are related to archiving (AWS Glacier) are not supported. Lifecycle is not supported on file system-enabled buckets.
DELETE Bucket policy	-
GET Bucket (List Objects)	● For file system-enabled buckets, / is the only supported delimiter when listing objects in the bucket. ● ECS returns the list results in UTF-16 binary order.
	● For file system-enabled buckets, / is the only supported delimiter when listing objects in the bucket. ● ECS returns the list results in UTF-16 binary order.
	ECS simplified bucket delete request status returns empty bucket status if active: `GET /<bkt>?empty-bucket-status`
	-
	Only the expiration part is supported in life cycle. Policies that are related to archiving (AWS Glacier) are not supported. Lifecycle is not supported on file system-enabled buckets.
	-
	-
	-
GET Bucket (List Objects) Version 2	-
GET Bucket cors	-
GET Bucket acl	Where `PUT` is performed on an existing bucket, refer to Behavior where bucket already exists.
GET Bucket life cycle	-
GET Bucket policy	-

Table 3. Supported S3 APIs (continued)

Feature	Notes
GET Bucket Object versions	Only the expiration part is supported in life cycle. Policies that are related to archiving (AWS Glacier) are not supported. Lifecycle is not supported on file system-enabled buckets.
GET Bucket versioning	Cannot configure the bucket policies for file system-enabled or CAS-enabled buckets. Cannot configure the bucket policies for operations that ECS does not support. More information about bucket policy support is provided in Bucket policy support.
PUT Bucket versioning	-
DELETE Object	
Delete Multiple Objects	-
GET Object	
GET Object ACL	-
HEAD Object	-
PUT Object	Supports chunked `PUT`
PUT Object acl	-
PUT Object - Copy	If the object size is greater than 5 GB, use `Upload Part - Copy` instead of `PUT Object - Copy`.
OPTIONS object	-
GET Object tagging	-
PUT Object tagging	-
DELETE Object tagging	-
Initiate Multipart Upload	-
Upload Part	-
Upload Part - Copy	-
Complete Multipart Upload	ECS returns an ETag of 00 for this request, which differs from the Amazon S3 response.
Abort Multipart Upload	-
List Parts	-
PUT Bucket Object Lock	-
GET Bucket Object Lock	-
PUT Bucket enable-object-lock	This is a custom API (not defined by S3)
PUT Object legal hold	-
GET Object legal hold	-
PUT Object retention	-
GET Object retention	-

NOTE:

- Creation of buckets using names with fewer than three characters fails with `400 Bad Request, InvalidBucketName`.
- When creating a bucket or object with empty content, ECS returns `400 invalid content-length value`, which differs from AWS which returns `400 Bad Request`.
- Copying an object to another bucket that indexes the same user metadata index key but with a different datatype is not supported and fails with `500 Server Error`.

- When listing the objects in a bucket, if you use a prefix and delimiter but supply an invalid marker, ECS throws 500 Server Error, or 400 Bad Request for a file system-enabled bucket. However, AWS returns 200 OK and the objects are not listed.
- For versioning enabled buckets, ECS does not create a delete marker when a deleted object is deleted again. This is different from AWS, which always inserts delete marker for deleting deleted objects in versioning enabled buckets. This change in behavior is only applicable when the deleted object is deleted again from owner zone.

Table 4. Additional features

Feature	Notes
Presigned URLs	ECS supports use of presigned URLs to grant access to objects without needing credentials. More information can be found at: https://docs.aws.amazon.com/AmazonS3/latest/dev/PresignedUrlUploadObject.html.
Chunked PUT	`PUT` operation can be used to upload objects in chunks, which enable content to be sent before the total size of the payload is known. Chunked transfer uses the Transfer-Encoding header (Transfer-Encoding: chunked) to specify that content is transmitted in chunks.

Unsupported S3 APIs

Table 5. Unsupported S3 APIs

Feature	Notes
DELETE Bucket tagging	-
DELETE Bucket website	-
GET Bucket location	ECS is only aware of a single Virtual Data Center (VDC).
GET Bucket logging	-
GET Bucket notification	Notification is only defined for reduced redundancy feature in S3. ECS does not support notifications.
GET Bucket tagging	-
GET Bucket requestPayment	ECS uses its own model for payments.
GET Bucket website	-
PUT Bucket logging	-
PUT Bucket notification	Notification is only defined for the reduced redundancy feature in S3. ECS does not support notifications.
PUT Bucket tagging	-
PUT Bucket requestPayment	ECS uses its own model for payments.
PUT Bucket website	-
Object APIs	
GET Object torrent	-
POST Object	-
POST Object restore	The POST Object restore operation is related to AWS Glacier, which is not supported in ECS.
SELECT Object Content	-

Behavior where bucket already exists

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

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

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

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

Bucket policy support

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

How access to resources is managed when using the S3 protocol is described in https://docs.aws.amazon.com/AmazonS3/latest/dev/s3-access-control.html and you can use the information as the basis for understanding and using S3 bucket policies in ECS. This section provides basic information about the use of bucket policies, and to identify the differences when using bucket policies with ECS.

The following provides an example of an ECS bucket policy:

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

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

Version	The Version field specifies the policy language version and can be either `2012-10-17` or `2008-10-17`. If the version is not specified, `2008-10-17` is automatically inserted. It is good practice to set the policy language for a new policy to the latest version, `2012-10-17`.
Id	The Id field is optional.

Each statement includes the following elements:

SID	A statement ID is a string that describes what the statement does.
Resources	The bucket or object that is the subject of the statement. The resource can be associated with a Resource or NotResource statement.

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

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

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

You can use the (`*`) and (`?`) wildcard characters, where asterisk (`*`) represents any combination of zero or more characters and a question mark (`?`) represents any single character. For example, you can represent all objects in bucket that is called *bucket name*, using:

```
bucketname/*
```

Actions

The set of operations that you want to assign permissions to (enable or deny). The supported operations are listed in Supported bucket policy operations.

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

Effect

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

Principal

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

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

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

Conditions

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

The following condition operators are not supported: Binary, ARN, IfExists, Check Key Exists. The supported condition keys are listed in Supported bucket policy conditions.

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

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

Creating, Assigning, and Managing Bucket Policies

This section describes about managing bucking policies.

You can create a bucket policy for a bucket from the ECS Portal (see the ECS Administration Guide in https://www.dell.com/support/). It is also possible to create a policy using another editor, and associate the policy with a bucket using the ECS Management REST API or using the ECS S3 API.

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

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

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

The ECS S3 API provides the following APIs:

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

NOTE:

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

Details of these APIs can be found in the ECS API Reference.

Bucket policy scenarios

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

You can use bucket policies in the following typical scenarios:

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

Grant bucket permissions to a user

To grant permission on a bucket to a user apart from the bucket owner, specify the resource that you want to change the permissions for. Set the principal attribute to the name of the user, and specify one or more actions that you want to enable.

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

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

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

```
{
    "Version": "2012-10-17",
    "Id": "S3PolicyId1",
    "Statement": [
        {
            "Sid": "Grant permission ",
            "Effect": "Allow",
            "Principal": ["user1"],
            "Action": [ "s3:PutObject","s3:GetObject" ],
            "Resource":[ "mybucket/*" ]
            "Condition": {"IpAddress": {"aws:SourceIp": "<Ip address>"}
            }
     ]
}
```

Grant bucket permissions to all users

To grant permission on a bucket to a user apart from the bucket owner, specify the resource that you want to change the permissions for. Set the principal attribute as anybody (`*`), and specify one or more actions that you want to enable.

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

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

Automatically assign permissions to created objects

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

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

Supported bucket policy operations

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

Table 6. Permissions for Object Operations

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

Table 7. Permissions for Bucket Operations

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

Table 8. Permissions for Bucket Sub-resource Operations

Permission keyword	Supported S3 operations
s3:GetBucketAcl	GET Bucket acl
s3:PutBucketAcl	PUT Bucket acl
s3:GetBucketCORS	GET Bucket cors
s3:PutBucketCORS	PUT Bucket cors
s3:GetBucketVersioning	GET Bucket versioning
s3:PutBucketVersioning	PUT Bucket versioning
s3:GetBucketPolicy	GET Bucket policy
s3:DeleteBucketPolicy	DELETE Bucket policy
s3:PutBucketPolicy	PUT Bucket policy

Supported bucket policy conditions

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

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

Table 9. Supported generic AWS condition keys

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

Table 10. Supported S3-specific condition keys for object operations

Key name	Description	Applicable permissions
s3:x-amz-acl	Sets a condition to require specific access permissions when the user uploads an object.	s3:PutObject, s3:PutObjectAcl, s3:PutObjectVersionAcl
s3:x-amz-grant-permission (for explicit permissions), where permission can be:read, write, read-acp, write-acp, full-control	Bucket owner can add conditions using these keys to require certain permissions.	s3:PutObject, s3:PutObjectAcl, s3:PutObjectVersionAcl

Table 10. Supported S3-specific condition keys for object operations (continued)

Key name	Description	Applicable permissions
s3:x-amz-server-side-encryption	Requires the user to specify this header in the request.	s3:PutObject, s3:PutObjectAcl
s3:VersionId	Restrict the user to accessing data only for a specific version of the object	s3:PutObject, s3:PutObjectAcl, s3:DeleteObjectVersion

Table 11. Supported S3-specific condition keys for bucket operations

Key name	Description	Applicable permissions
s3:x-amz-acl	Set a condition to require specific access permissions when the user uploads an object	s3:CreateBucket, s3:PutBucketAcl
s3:x-amz-grant-permission (for explicit permissions), where permission can be:read, write, read-acp, write-acp, full-control	Bucket owner can add conditions using these keys to require certain permissions	s3:CreateBucket, s3:PutBucketAcl
s3:prefix	Retrieve only the object keys with a specific prefix.	s3:ListBucket, s3:ListBucketVersions
s3:delimiter	Require the user to specify the delimiter parameter in the Get Bucket (List Objects) request.	s3:ListBucket, s3:ListBucketVersions
s3:max-keys	Limit the number of keys ECS returns in response to the Get Bucket (List Objects) request by requiring the user to specify the max-keys parameter. **NOTE:** In EXF900 systems, you can set the `max-keys` parameter value up to 20000 per list request.	s3:ListBucket, s3:ListBucketVersions

Object Tagging

Object Tagging allows you to categorize the objects by assigning tags to the individual objects. A single object can have multiple tags that are associated with it, enabling multidimensional categorization.

A tag could describe some sort of sensitive information like a health record, or you can tag an object. to a certain product that can be categorized as confidential. Tagging is a subresource of an object that has a life-cycle, integrated with object operations. You can add tags to new objects when you upload them, or add tags to existing objects. It is acceptable to use tags to label objects containing confidential data, such as Personally Identifiable Information (PII) or Protected Health Information (PHI). The tags must not contain any confidential information, as tags can be viewed without having the read permission to an object.

WARNING: Object Tagging does not support file system-enabled buckets with S3 protocol.

Two parameters are available in ECS Object Tagging:

Tag	A tag is a key-value pair where both the key and the value are represented as a string.
Tag set	A set of tags associated with an object. Tags that are associated with an object must have unique tag keys. You can associate up to 10 tags with an object. However, the additional storage overhead is about 4 kb in UTF-8 and sixteen kb in UTF-32.

NOTE:
- A single tag would require about 384 bytes on disk if stored in UTF-8 encoding or about 1.5 kb if stored as UTF-32.
- Allowed characters are letters, numbers, and spaces representable in UTF-8, and the following characters: + - = . _ : / @
- A tag key can be up to 128 Unicode characters in length, and tag values can be up to 256 Unicode characters in length.
- The key and values are case-sensitive.

- Object Tagging API support is available from ECS 3.5 and later versions. For more information about Object Tagging APIs, see Manage Object Tagging.

Additional information about Object Tagging

This section provides information about Object Tagging in IAM, Object Tagging with bucket policies, handling Object Tagging during TSO/PSO, and Object Tagging during object lifecycle management.

Object Tagging in IAM	The key function of Object Tagging as categorization system comes when it is integrated with Polices from IAM. This allows you to configure specific permissions for the users. For example, You can add a policy that allows everyone to access objects with a specified tag or you can configure and grant permissions to users, who can manage the tags on specific objects. The other key aspect with Object Tagging is how and where the tags would be persisted. This is important because, it has a direct impact on various aspects of the system.
Object Tagging with bucket policies	Object Tagging allows you to categorize the objects, also tagging gets integrated with various policies. Lifecycle management policy allows you to configure at a bucket level. Earlier versions of ECS support Expiration, Abort Incomplete Uploads, and Deletion of Expired Delete Marker. The filter could include multiple conditions including a tag-based condition. Each tag in the filter condition must match the key and the value.
Object Tagging during TSO/PSO	Object Tagging is another entry set in system metadata, no special handling is required during TSO/PSO. There is a set limit on the number of tags that are allowed to be associated with each object, size of system metadata along with Object Tagging is well with in the memory limits.
Object Tagging during object lifecycle Management	Object Tagging is part of system metadata and handled simultaneously with system metadata handling, during lifecycle management. The Expiration Logic and Lifecycle Delete Scanner requires to understand tag-based policies. Object tags enable fine-grained object lifecycle management in which you can specify a tag-based filter, in addition to a key name prefix, in a lifecycle rule.

Object Tagging operations

This section describes all the API-related operations that are specific to object tagging.

By default, the bucket owner has permissions to `PUT`, `GET`, and `DELETE` tags.

All the operations that have the concept of tags are categorized into management operations and object operation with tagging. The operations use the `tagging` subresource to manipulate the tags and apply them to the current version of an object. To manipulate tags on a previous version, add `versionId` parameter.

Table 12. API operations related to object tagging

Operations	Path	Description
PUT object tagging	PUT /ObjectName?tagging	Adds or replaces a set of tags to an existing object.
GET object tagging	GET /ObjectName?tagging	Returns the set of tags that are associated with the requested object.
DELETE Object tagging	DELETE /ObjectName?tagging	Deletes the set of tags that are associated with the requested object.

The object operations with tagging include existing object operations that can perform tag related actions such as, assigning the tags at the time of object creation or during the copy operation.

Table 13. Object operations with tagging

Operations	Request/Response	Description
PUT object	Request header: `x-amz-tagging`	• Requires `WRITE` permission and `s3:PutObjectTagging` permission. • The encoding of tags should follow URL query parameter. • The tags pass as a request header, so the limit is 2 KB.

Table 13. Object operations with tagging (continued)

Operations	Request/Response	Description
Initiate Multipart Upload	Request header: `x-amz-tagging`	• Requires `WRITE` permission and `s3:PutObjectTagging` permission. • The encoding of tags must follow URL query parameter. • The tags pass as a request header, so the limit is 2 KB.
GET object	Response header: `x-amz-tagging-count`	• Returns the count of tags that are associated with the object. • Nothing returns, if the permission capability is not set. • The count is not returned, If no tags are associated with the object.
PUT object COPY	Request header: `x-amz-tagging-directive`	• Specifies source object tags copy policy: ○ COPY: Copies tags as-is ○ REPLACE: Replaces them with the new set from the request. • The default behavior is `COPY` unless explicitly specified by the user. • In case of `REPLACE`, the tags must follow URL query parameter encoding. • No tags are allowed with `REPLACE` directive, results in no tags on the target object.

Add or extend internal APIs to support tagging. The table defines the S3 APIs to manipulate the tags.

Table 14. S3 APIs

API support	Action	Description
`updateObjectTags()`	s3:PutObjectTagging	Add or replace set of tags to an existing object.
`updateObjectVersionTags()`	s3:PutObjectVersionTagging	Add or replace set of tags to an existing version of an object.
`getObjectTags()`	s3:GetObjectTagging	Retrieve all tags associated with current version of an object.
`getObjectVersionTags()`	s3:GetObjectVersionTagging	Retrieve all tags associated with specified version of an object.
`deleteObjectTags()`	s3:DeleteObjectTagging	Delete all tags associated with current version of an object.
`deleteObjectVersionTags()`	s3:DeleteObjectVersionTaggin g	Delete all tags associated with specified version of an object.

Manage Object Tagging

This section describes all the supported APIs that are required to manage Object Tagging.

PUT object tagging

- `updateObjectTags`.
- Returns with `ObjectWriteResponse`.
- Replaces the existing tag set on an object with `objectInfo` to new `objectTags`. This API does not merge existing tags with new tags,`objectTags` passed to this API replaces the existing objectTags (if any) for the object with `objectInfo`.

NOTE:

- If you send this request with an empty tag set, it deletes the existing tags for an object with `objectInfo`.
- Use the DELETE Object Tagging request to delete all the tags for an object.

GET Object Tagging	• `getObjectTags`. • Returns with set of tags associated with object. • Retrieves the set of tags that are associated with requested object with `objectInfo` and `versionId`.
DELETE Object Tagging	• `deleteObjectTags` • Returns true if deletion of object tag is successful for object. • Deletes all tags that are associated with an object that is identified by `objectInfo` for `current/versionId` of an object.

Table 15. Object Tagging parameters

Parameter	Description
`objectInfo`	Unique ID associated with the object to which, the `objectTags` must be updated.
`keypoolData`	Information regarding `keypool`.
`versionId`	Version of the object on which, the tags must be updated. Value `NULL` retrieves tags for current version of the object
`credential`	Permission to update tags for an object
`objectTags`	Set of tags that must be associated with an object.

S3 Object Lock

ECS allows you to store objects using a write-once-read-many (WORM) model through S3 Object Lock. This feature prevents objects from being deleted or overwritten for a specified time or indefinitely. Also, S3 Object Lock helps to meet WORM storage related regulatory requirements and adds a protection layer against object modifications and deletion.

NOTE:

- ECS S3 Object Lock feature supports only the versioning enabled buckets.
- There is no ECS user interface for Object Lock. It can be accessed through ECS Object Lock APIs. For the Object Lock API examples, see Object Lock API Examples and for the list of supported S3 APIs, see S3 API supported and unsupported features.
- The locked objects are protected from lifecycle deletions.

Managing Object Lock

ECS S3 Object Lock allows you to manage object retention through retention periods and legal holds.

- Through retention period, you can specify a period during which an object remains locked. During the specified period, the object is WORM-protected, that is, the object cannot be overwritten or deleted.
- Legal hold provides the same protection similar to retention period. However, legal hold is independent from retention period and it does not have an expiration date. Legal hold can be remained in objects until you explicitly remove it.

Retention period and legal hold can be specified in objects by any user who has the appropriate Object Lock permissions. For the list of supported Object Lock condition keys and permissions, see ECS S3 Object Lock condition keys and ECS S3 Object Lock permissions.

Retention mode

Retention mode

Retention modes provide additional protection to your object version that is protected by Object Lock.

Retention modes	Description
Governance mode	In governance mode, • Users cannot overwrite or delete an object version.

Retention modes	Description
	• Users with `s3:PutObjectRetention` permission can increase an object retention period. • Users with special `s3:BypassGovernanceRetention` permission can remove or shorten an object retention. • Users with `s3:BypassGovernanceRetention` permission can also delete locked objects.
Compliance mode	In compliance mode, • Users cannot overwrite or delete an object version. • Users with `s3:PutObjectRetention` permission can increase an object retention period. • Any user cannot remove or shorten an object retention.

Object Lock and ADO

Object Lock and ADO can be enabled together in a bucket by users with system administrator privileges, when the data loss risks during a temporary site outage are well understood.

Access During Outage (ADO) is a behavior that allows data access during a temporary site outage (TSO). When Object Lock and ADO are enabled together in a bucket, there is a risk of losing locked versions during a TSO. As a result, for ADO buckets, setting Object Lock is denied by default. You can allow Object Lock and ADO to co-exist, when you have system administrator privileges, and you understand the risk of losing locked versions of data during a TSO.

For more information about ADO, see the ECS Administration Guide.

NOTE:

- This feature cannot be managed through the UI, it can be viewed or modified only through the Management API.
- It is only available in ECS systems completely upgraded to 3.8 for all zones.
- Object Lock and ADO can be enabled or disabled on a namespace as default behavior, which will be applicable only to new buckets.
- Object Lock and ADO can be enabled in buckets, and if not specified will use the namespace default value.
- Once this feature is enabled in a bucket, it cannot be disabled.

Common issues while enabling Object Lock and ADO

ECS 3.8 allows you to enable Object Lock and ADO in a bucket when you have system administrator privileges to allow it. The following issues should be noted.

If you want to set up both Object Lock and ADO, system administrators can enable new flag on individual buckets or set as default for all future bucket creations for a namespace. Once set to allowed, they can enable Object Lock and ADO on that bucket.

If you want to change Object Lock and ADO from allowed to *not allowed*, note that it is not possible. Once Object Lock and ADO are enabled in a bucket, it cannot be disabled.

In Object Lock and ADO buckets, there is a risk of data loss during a temporary site outage. Users should understand the risks before enabling both the services together.

ECS S3 Object Lock condition keys

ECS S3 Object Lock condition keys allow you to limit what retention period and legal hold can be specified in objects.

Condition Key	Description
`s3:object-lock-legal-hold`	Enables enforcement of the specified object legal hold status
`s3:object-lock-mode`	Enables enforcement of the specified object retention mode
`s3:object-lock-retain-until-date`	Enables enforcement of a specific retain-until-date
`s3:object-lock-remaining-retention-days`	Enables enforcement of an object relative to the remaining retention days

ECS S3 Object Lock permissions

ECS S3 Object Lock permissions allow you to manage retention period and legal hold that are specified in objects.

Permissions	Operations
`s3:PutBucketObjectLockConfiguration`	PUT Bucket Object Lock configuration
`s3:GetBucketObjectLockConfiguration`	GET Bucket Object Lock configuration
`s3:PutObjectLegalHold`	PUT Object Legal Hold, PUT Object
`s3:GetObjectLegalHold`	GET Object Legal Hold, GET Object
`s3:PutObjectRetention`	PUT Object Retention, PUT Object
`s3:GetObjectRetention`	Get Object Retention, GET Object
`s3:BypassGovernanceRetention`	PUT Object Retention, DELETE Object, DELETE Objects
`s3:EnableObjectLock`	Enable object lock for existing buckets

Object Lock API Examples

This section lists s3curl examples of Object Lock APIs.

NOTE: Put and Get Object Lock APIs can be used with and without `versionId` parameter. If no `versionId` parameter is used, then the action applies to the latest version.

Operation	API request examples
Create lock-enabled bucket	`s3curl.pl --id=ecsflex --createBucket -- http://${s3ip}/my-bucket -H "x-amz-bucket-object-lock-enabled: true"`
Enable object lock on existing bucket	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket?enable-object-lock -X PUT`
Get bucket default lock configuration	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket?object-lock`

Put bucket default lock configuration	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket?object-lock -X PUT \`
	 `    -d "<ObjectLockConfiguration><ObjectLockEnabled>Enabled</ObjectLockEnabled>`
	
	 ` <Rule><DefaultRetention><Mode>GOVERNANCE</Mode><Days>1</Days></DefaultRetention></Rule></ObjectLockConfiguration>"`

Get legal hold	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket/obj?legal-hold`
Put legal hold on create	`s3curl.pl --id=ecsflex --put=/root/100b.file -- http://${s3ip}/my-bucket/obj -H "x-amz-object-lock-legal-hold: ON"`
Put legal hold on existing object	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket/obj?legal-hold -X PUT -d "<LegalHold><Status>OFF</Status></LegalHold>"`
Get retention	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket/obj?retention`

Operation	API request examples

Put retention on create	`s3curl.pl --id=ecsflex --put=/root/100b.file -- http://${s3ip}/`
	`my-bucket/obj -H "x-amz-object-lock-mode: GOVERNANCE" -H "x-amz-`
	`object-lock-retain-until-date: 2030-01-01T00:00:00.000Z"`

Put retention on existing object	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket/obj?`
	`retention -X PUT -d "<Retention><Mode>GOVERNANCE</`
	`Mode><RetainUntilDate>2030-01-01T00:00:00.000Z</`
	`RetainUntilDate></Retention>"`

Put retention on existing object (with bypass)	`s3curl.pl --id=ecsflex -- http://${s3ip}/my-bucket/obj?`
	`retention -X PUT -d "<Retention><Mode>GOVERNANCE</`
	`Mode><RetainUntilDate>2030-01-01T00:00:00.000Z</`
	`RetainUntilDate></Retention>" -H "x-amz-bypass-governance-`
	`retention: true"`

Enable allowing object lock and ADO as the default on existing namespace
NOTE: The flag does not turn on ADO or object lock, it just allows both to be set
	`curl -v -k -H "$TOKEN" -H 'Accept:application/`
	`json' -H 'Content-Type:application/json' -X PUT -d`
	`‘{“is_object_lock_with_ado_allowed”:”true”}’ https://$vm1:4443/`
	`object/namespaces/namespace/s3`

Enable allowing object lock and ADO as the default when creating namespace	`curl -v -k -H "$TOKEN" -H 'Accept:application/json' -H 'Content-`
	`Type:application/json' -X POST -d @./Namespace.json  https://`
	`$vm1:4443/object/namespaces/namespace`
	`Namespace.json:`
	`{ "namespace": "mynamespace","default_data_services_vpool":`
	`"urn:some-vpool","is_encryption_enabled":`
	`"false","is_stale_allowed": "false","compliance_enabled":`
	`"false","is_object_lock_with_ado_allowed": "true"}`

Enable allowing object lock and ADO on existing bucket	`curl -s -k -H "$TOKEN" -H 'Accept:application/json'`
	` -X PUT https://$vm1:4443/object/bucket/<bucket>/allow-object-`
	`lock-with-ado?namespace=<namespace>`

Enable allowing object lock and ADO when creating bucket	`curl -v -k -H "$TOKEN" -H 'Accept:application/json' -H 'Content-`
	`Type:application/json' -X POST -d @./bucket-create.json  https://`
	`$vm1:4443/object/bucket`
	`Bucket-create.json:`
	`{…`
	`"is_object_lock_with_ado_allowed": "true‟`
	`}`
	`Is_object_lock_with_ado_allowed - value used if present,`
	`otherwise will use default from namespace`

Object lifecycle management

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

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

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

- Lifecycle cannot be enabled on FS enabled buckets.
- Lifecycle is a bucket level concept.
- Maximum of 1000 lifecycle rules per bucket is applicable.
- There may be a delay between the expiration date and the date at which S3 removes an object.
- Always round up the resulting time to the next day midnight UTC.
- For expiration, the days are calculated since the last modified date (= Creation date for the objects not yet modified/ deleted).
- If you delete the data accidentally, raise a Service Request (SR) with the support team. For more information about recovering the data, see KB 539120.
- For noncurrentexpiration, the days are calculated since the object became noncurrent.
- The date-based rules trigger action on all objects created on or before this date.

Example lifecycle configurations for ECS

The following are some lifecycle configurations examples.

Aborting old MPU's (versioning and non-versioning enabled buckets)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-expire-non-current-and-dmarkers-and-mpu</ID>
    <Filter/>
    <Status>Enabled</Status>
    <AbortIncompleteMultipartUpload>
      <DaysAfterInitiation>1</DaysAfterInitiation>
    </AbortIncompleteMultipartUpload>
  </Rule>
</LifecycleConfiguration>
```

Expiring objects after a certain # of days (versioning and non-versioning enabled buckets)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-expire-one-year</ID>
    <Filter/>
    <Status>Enabled</Status>
    <Expiration>
      <Days>365</Days>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Expiring NoncurrentVersions of objects after a certain # of days (versioning enabled buckets only)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-expire-non-current</ID>
    <Filter/>
    <Status>Enabled</Status>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>1</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

Removing expired object delete markers (versioning enabled buckets only)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-expire-dmarkers</ID>
    <Filter/>
    <Status>Enabled</Status>
    <Expiration>
```

```
      <ExpiredObjectDeleteMarker>true</ExpiredObjectDeleteMarker>
    </Expiration>
  </Rule>
</LifecycleConfiguration>
```

Expire all non-current versions, dmarkers and incomplete MPU's after 1 day

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-expire-non-current-and-dmarkers-and-mpu</ID>
    <Filter/>
    <Status>Enabled</Status>
    <Expiration>
      <ExpiredObjectDeleteMarker>true</ExpiredObjectDeleteMarker>
    </Expiration>
    <AbortIncompleteMultipartUpload>
      <DaysAfterInitiation>1</DaysAfterInitiation>
    </AbortIncompleteMultipartUpload>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>1</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
```

PUT/GET lifecycle with s3curl examples

The following are PUT and GET lifecycle with s3curl examples. See Using s3curl with ECS for more information.

PUT lifecycle

```
admin@:/usr/share/s3curl> cat lifecycle.xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-non-current-expiration</ID>
    <Filter/>
    <Status>Enabled</Status>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>1</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
admin@:/usr/share/s3curl>
admin@:/usr/share/s3curl> sudo perl ./s3curl.pl --debug --id=emc --put=lifecycle.xml --
calculateContentMd5 -- "http://192.0.2.0:9020/emc_lifecycle?lifecycle" -v
s3curl: Found the url: host=10.32.169.121; port=9020; uri=/emc_lifecycle;
query=lifecycle;
s3curl: replaced string: lifecycle
s3curl: ordinary endpoint signing case
s3curl: StringToSign='PUT\nFjZKcAgVegBUaGdqfEh/Ig==\n\nTue, 06 Nov 2018 17:28:58 +0000\n/
tom_lifecycle?lifecycle'
s3curl: exec curl -v -H 'Date: Tue, 06 Nov 2018 17:28:58 +0000' -H 'Authorization:
AWS emc:xDTXdXSF+qVIQ4EreEe+iqlHRns=' -L -H 'content-type: ' -H 'Content-MD5:
FjZKcAgVegBUaGdqfEh/Ig==' -T lifecycle.xml http://192.0.2.0:9020/tom_lifecycle?lifecycle
-v
* Hostname was NOT found in DNS cache
*   Trying 192.0.2.0...
* Connected to 192.0.2.0 (192.0.2.0) port 9020 (#0)
> PUT /emc_lifecycle?lifecycle HTTP/1.1
> User-Agent: curl/7.37.0
> Host: 192.0.2.0:9020
> Accept: */*
> Date: Tue, 06 Nov 2018 17:28:58 +0000
> Authorization: AWS emc:xDTXdXSF+qVIQ4EreEe+iqlHRns=
> Content-MD5: FjZKcAgVegBUaGdqfEh/Ig==
> Content-Length: 376
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
```

```
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Date: Tue, 06 Nov 2018 17:28:58 GMT
* Server ViPR/1.0 is not blacklisted
< Server: ViPR/1.0
< x-amz-request-id: 0a20a979:166c6842ba5:82ba:5
< x-amz-id-2: 6687ce5967202724ed9a94d44c939438d39cabae9abc5a2c48a60c2c5355f95e
< Content-Length: 0
<
* Connection #0 to host 10.32.169.121 left intact


Troubleshooting LDS:
Enabling debug logging for LDS
LDS log is in resourcesvc-log4j2.xml
<Logger name="com.emc.storageos.data.object.impl.resource.LifeCycleDeleteScanner"
level="DEBUG"/>


Other relevant classes for troubleshooting lifecycle issues from blobsvc-log4j2.xml
<Logger name="com.emc.storageos.data.object.impl.gc.DeleteJobScanner" level="DEBUG"/>
<Logger
name="com.emc.storageos.data.object.impl.file.directoryTable.ObjectDirectoryOperation"
level="DEBUG"/>
<Logger
name="com.emc.storageos.data.object.impl.file.directoryTable.BlobsvcOperationBase"
level="DEBUG"/>
<Logger name="com.emc.storageos.data.object.impl.file.ObjectExpirationHelper"
level="DEBUG"/>

dataheadsvc-log4j2.xml
<Logger name="com.emc.storageos.data.object.RESTAccess.ObjectListingHelper"
level="DEBUG"/>
```

GET lifecycle

```
:/usr/share/s3curl # perl ./s3curl.pl --id=EMC -- "http://192.0.2.0:9020/test-bucket/?
lifecycle" -s | xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<LifecycleConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Rule>
    <ID>lifecycle-v2-abortmpu-one-week</ID>
    <Filter/>
    <Status>Enabled</Status>
    <NoncurrentVersionExpiration>
      <NoncurrentDays>1</NoncurrentDays>
    </NoncurrentVersionExpiration>
  </Rule>
</LifecycleConfiguration>
:/usr/share/s3curl #
```

Supported lifecycle configuration elements

Table 16. Supported lifecycle configuration elements

Name	Description	Required
`AbortIncompleteMultipartUpload`	• Container for specifying when an incomplete multipart upload becomes eligible for an abort operation. • When you specify this lifecycle action, the rule cannot specify a tag-based filter. • Child: DaysAfterInitiation • Type: Container • Ancestor: Rule	Yes, if no other action is specified for the rule.

Table 16. Supported lifecycle configuration elements (continued)

Name	Description	Required
`And`	• Container for specify rule filters. These filters determine the subset of objects to which the rule applies. • Type: String • Ancestor: Rule	Yes, if you specify more than one filter condition (for example, one prefix and one or more tags).
`Date`	• Date when you want S3 to take the action. • The date value must conform to the ISO 8601 format. The time is always midnight UTC. • Type: String • Ancestor: Expiration or Transition	Yes, if `Days` and `ExpiredObjectDeleteMarker` are absent.
`Days`	• Specifies the number of days after object creation when the specific rule action takes effect. • Type: Nonnegative Integer when used with Transition, Positive Integer when used with Expiration. • Ancestor: Expiration, Transition	Yes, if `Date` and `ExpiredObjectDeleteMarker` are absent.
`DaysAfterInitiation`	• Specifies the number of days after initiating a multipart upload when the multipart upload must be completed. If it does not complete by the specified number of days, it becomes eligible for an abort operation and S3 aborts the incomplete multipart upload. • Type: Positive Integer. • Ancestor: `AbortIncompleteMultipartUpload`	Yes, if ancestor is specified.
`Expiration`	• This action specifies a period in an object's lifetime when S3 should take the appropriate expiration action. The action S3 takes depends on whether the bucket is versioning-enabled. • If versioning has never been enabled on the bucket, S3 deletes the only copy of the object permanently. Otherwise, if your bucket is versioning-enabled (or versioning is suspended), the action applies only to the current version of the object. A versioning-enabled bucket can have many versions of the same object, one current version, and zero or more noncurrent versions. • Instead of deleting the current version, S3 makes it a noncurrent version by adding a delete marker as the new current version. **NOTE:** ○ If your bucket state is versioning-suspended, S3 creates a delete marker with version ID null. If you have a version with version ID null, then S3 overwrites that version. ○ To set expiration for noncurrent objects, you must use the `NoncurrentVersionExpiration` action. • Type: Container • Children: Days or Date	Yes, if no other action is present in the Rule.

Table 16. Supported lifecycle configuration elements (continued)

Name	Description	Required
	• Ancestor: Rule	
`Filter`	• Container for elements that describe the filter identifying a subset of objects to which the lifecycle rule applies. If you specify an empty filter (<Filter></Filter>), the rule applies to all objects in the bucket. • Type: String • Children: Prefix, Tag • Ancestor: Rule	Yes
`ID`	• Unique identifier for the rule. The value cannot be longer than 255 characters. • Type: String • Ancestor: Rule	No
`Key`	• Specifies the key of a tag. A tag key can be up to 128 Unicode characters in length. • Tag keys that you specify in a lifecycle rule filter must be unique. • Type: String • Ancestor: Tag	Yes, if <Tag> parent is specified.
`LifecycleConfiguration`	• Container for lifecycle rules. You can add as many as 1,000 rules. • Type: Container • Children: Rule • Ancestor: None	Yes
`ExpiredObjectDeleteMarker`	• On a versioned bucket (versioning-enabled or versioning-suspended bucket), you can add this element in the lifecycle configuration to direct S3 to delete expired object delete markers. On a nonversioned bucket, adding this element in a policy is meaningless because you cannot have delete markers and the element does not do anything. • When you specify this lifecycle action, the rule cannot specify a tag-based filter. • Type: String • Valid values: true | false (the value false is allowed, but it is no-op and S3 does not take action if the value is false) • Ancestor: Expiration	Yes, if `Date` and `Days` are absent.
`NoncurrentDays`	• Specifies the number of days an object is noncurrent before S3 can perform the associated action. • Type: Nonnegative Integer when used with `NoncurrentVersionTransition`, Positive Integer when used with `NoncurrentVersionExpiration`. • Ancestor: `NoncurrentVersionExpiration` or `NoncurrentVersionTransition`	Yes
`NoncurrentVersionExpiration`	• Specifies when noncurrent object versions expire. Upon expiration, S3 permanently deletes the noncurrent object versions.	Yes, if no other action is present in the `Rule`.

Table 16. Supported lifecycle configuration elements (continued)

Name	Description	Required
	• You set this lifecycle configuration action on a bucket that has versioning enabled (or suspended) to request that S3 delete noncurrent object versions at a specific period in the object's lifetime. • Type: Container • Children: NoncurrentDays • Ancestor: Rule	

`Prefix`	• Object key prefix identifying one or more objects to which the rule applies. Empty prefix (<Prefix></Prefix>) indicates there is no filter based on key prefix.
	NOTE: ECS supports <Prefix> with and without <Filter>.
	PUT Bucket lifecycle with <Filter>
	`<Filter>`
	`    <Prefix>value</Prefix>`
	`</Filter>`
	PUT Bucket lifecycle (Deprecated) without <Filter>
	`<Prefix>value</Prefix>`
	• There can be at most one Prefix in a lifecycle rule Filter.
	• Type: String
	• Ancestor: Filter or And (if you specify multiple filters such as a prefix and one or more tags)
		No

`Rule`	• Container for a lifecycle rule. A lifecycle configuration can contain as many as 1,000 rules. • Type: Container • Ancestor: LifecycleConfiguration	Yes
`Status`	• If Enabled, S3 executes the rule as scheduled. If Disabled, S3 ignores the rule. • Type: String • Ancestor: Rule • Valid values: Enabled, Disabled.	Yes
`Value`	• Specifies the value for a tag key. Each object tag is a key-value pair. • Tag value can be up to 256 Unicode characters in length. • Type: String • Ancestor: Tag	Yes, if `<Tag>` parent is specified.

Enabling Lifecycle Delete Scanner (LDS)

The purpose of the LDS scanner is to initiate expiration of objects/versions created before the lifecycle is applied. So for instance, if there is a bucket created sometime ago and has been in use and now there is a requirement to apply lifecycle, in such cases LDS must be enabled for lifecycle policies to cover previous objects/versions.

NOTE: LDS is disabled by default. For enabling pre 3.2.1, contact .

For enabling 3.2.1 and higher versions, set the `com.emc.ecs.resource.lifecycledeletescanner.enable` parameter value as true.

```
svc_param set com.emc.ecs.resource.lifecycledeletescanner.enable -v "true" -r "Enable
LDS"
```

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
- File system enabled
- Metadata Search
- S3A support
- S3 Select

Byte range extensions

The following byte range extensions are provided:

- Updating a byte range within an object
- Overwriting part of an object
- Appending data to an object
- Reading multiple byte ranges within an object

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

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

Updating a byte range within an object

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

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

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

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

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

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

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

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

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

green

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

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

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

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

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

Overwriting part of an object

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

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

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

```
PUT /bucket1/object1 HTTP/1.1
Content-Length: 9
Range: bytes=10-
ACCEPT: application/json,application/xml,text/html,application/octet-stream
Date: Mon, 12 Mar 2018 20:51:41 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:uwPjDAgmazCP5lu77Zvbo+CiT4Q=
Accept-Encoding: gzip, deflate, compress

brown cat

HTTP/1.1 204 No Content
ETag: 25
x-amz-id-2: object1
x-amz-request-id: 65be45c2-0ee8-448a-a5a0-fff82573aa3b
```

```
Content-Length: 0
Date: Mon, 12 Mar 2018 20:51:41 GMT
```

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

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

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

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

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

Appending data to an object

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

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

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

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

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

and cat

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

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

```
GET /bucket1/object1 HTTP/1.1
ACCEPT: application/json,application/xml,text/html,application/octet-stream
Date: Mon, 12 Mar 2018 20:46:56 -0000
x-emc-namespace: emc
```

```
Content-Type: application/octet-stream
Authorization: AWS wuser1:D8FSE8JoLl0MTQcFmd4nG1gMDTg=
Accept-Encoding: gzip, deflate, compress

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

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

Reading multiple byte ranges within an object

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

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

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

```
GET /bucket1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Range: bytes==4-8,41-44
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

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

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

Retention

The ECS S3 head supports retention of objects to prevent them being deleted or modified for a specified period. The ECS S3 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.

NOTE: The objects retention period can be extended. See Extending retention period on objects for more information.

Retention policy on object
A retention policy can be set on the object and the period that is associated with the policy can be set for the namespace. The retention policy 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.

When applying a retention policy to an object using a `x-emc-retention-policy` header on the object, the policy retention period must be set. The ECS administrator must set the policy retention period from the ECS Portal or using the ECS Management REST API.

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

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

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

Extending retention period on objects

ECS allows you to extend the retention period for the objects that are under retention.

To extend the object retention period, add a value to the `x-emc-retention-period` header. For example, a value of -1 implies infinitive retention period.

Limitations

- It is possible to extend the object retention period by changing the `retention_class` values in the namespace or in the bucket. The modified retention period applies to all the objects in the bucket or to all the objects that are using the `retention_class` defined in namespace. It is not possible to extend an individual object retention period.
- As to extend the retention period, namespace should have `retention_class` with some period value, buckets should have `retention-period`, and objects should have `retention-period` and `retention_class` as defined in the namespace.

NOTE: When modifying the retention value of an object, the new value is added to the object creation time. If the retention period value is greater than the current time, the operation will be blocked.

Table 17. S3 API to extend the object retention period

Action	Request	Permission	Response
Update retention period	`PUT /<bucket>/<object-key>?retentionUpdate x-emc-retention-period:<seconds>` Example: `/root/s3curl.pl --id=<id> -- -X PUT "http://$ip:9020/bucket/key1?retentionUpdate" -H 'x-emc-retention-period:3000'` **NOTE:** ● New retention period value can only be increased that is, it can be the same as the current or greater value.	`s3:PutObject`	`200 No content`

Table 17. S3 API to extend the object retention period

Action	Request	Permission	Response
	• If the new retention period value is -1, infinite retention applies on that object. For example: `-H 'x-emc-retention-period:-1'`		

File system enabled

This topic explains about File System (FS) enablement feature.

S3 buckets can be File System (FS) enabled so that the files that are written using the S3 protocol can be read using the file protocols, such as Network File system (NFS), and the opposite way.

NOTE: Only the objects that are created in the file system enabled buckets, inherit group permissions from the bucket settings if those permissions are not specified.

Enabling FS access

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

Limitation on FS support

When a bucket is FS enabled S3 life cycle management cannot be enabled.

Cross-head support for FS

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

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

You can find more information about the cross-head support with file systems in the ECS Administration Guide which is available from the https://www.dell.com/support/.

NFS WORM (Write Once, Read Many)

NFS data become Write Once Read Many (WORM) compliant when autocommit is implemented on it.

In detail, creating files through NFS is a multi step process. To write to a new file, NFS client first sends the CREATE request with no payload to NFS server. After receiving a response, the server issues a WRITE request. It is a problem for FS enabled buckets under retention as the file created with 0 bytes blocks any writes to it. Due to this reason, until ECS 3.3, retention on FS enabled bucket makes the whole mounted file-system read-only. There is no End of File (EOF) concept in NFS. Setting a retention for files, on the FS enabled buckets, after writing to them does not work as expected.

To remove the constraints that are placed on NFS files in a retention enabled bucket, the autocommit period is implemented on NFS data. For this reason, it is decided to introduce the autocommit period during which certain types of updates (for now identified as writes, Acl updates and deletes that are required for rsync, and rename that is required for Vim editor) are allowed, which removes the retention constraints for that period alone.

NOTE:

- The autocommit and the Atmos retention start delay are the same. See Retention start delay window.
- Autocommit period is a bucket property like retention period.
- Autocommit period is:
  - Applicable only for the file system enabled buckets with retention period
  - Applicable to the buckets in noncompliant namespace
  - Applies to only requests from NFS and Atmos

Seal file

The seal file functionality helps to commit the file to WORM state when the file is written ignoring the remaining autocommit period. The seal function is performed through the command: `chmod ugo-w <file>` on the file.

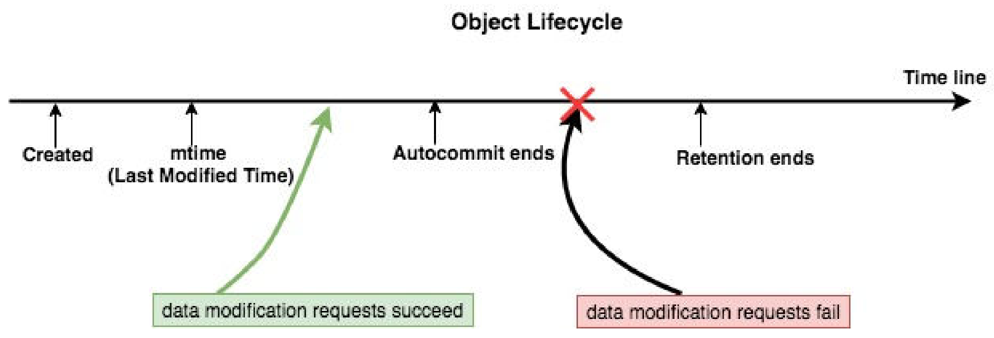
NOTE: The seal functionality does not have any effect outside the retention period.

High-level overview

Table 18. Autocommit terms

Term	Description
Autocommit period	Time interval relative to the object's last modified time during which certain retention constraints (example: file modifications, file deletions, and so on) are not applied. It does not have any effect outside of the retention period.
Retention Start Delay	Atmos head uses the start delay to indicate the autocommit period.

The following diagram provides an overview of the autocommit period behavior.



Autocommit configuration

The autocommit period can be set from the user interface or bucket REST API or S3 head or Atmos subtenant API.

User Interface

The user interface has the following support during bucket create and edit:
- When the File System is not enabled, no autocommit option is displayed.
- When the File System is enabled /no retention value that is specified, autocommit is displayed but disabled.
- When the File System is enabled/retention value selected/autocommit is displayed and enabled for selection.

NOTE: Maximum autocommit period is limited to the smaller of the Bucket Retention period or the default maximum period of one day.

REST API

Create bucket REST API is modified with the new header, `x-emc-autocommit-period`.

```
lglou063:~ # curl -i -k -T /tmp/bucket -X POST https://10.247.99.11:4443/object/bucket
-H "$token" -H "Content-Type: application/xml" -v

The contents of /tmp/bucket
<object_bucket_create>
    <name>bucket2</name>
    <namespace>s3</namespace>
    <filesystem_enabled>true</filesystem_enabled>
    <autocommit_period>300</autocommit_period>
    <retention>1500</retention>
</object_bucket_create>
```

S3 head

Bucket creation

Bucket creation flow through s3 head can make use of optional request header, `x-emc-auto-commit-period:seconds` to set the autocommit period. The following checks are made in this flow:

- Allow only positive integers
- Settable only for file system buckets
- Settable only when the retention value is present

```
./s3curl.pl --ord --id=naveen --key=+1Zh4YC2r2puuUaj3Lbnj3u0G9qgPRj0RIWJhPxH --
createbucket -- -H 'x-emc-autocommit-period:600' -H 'x-emc-file-system-access-
enabled:true' -H 'x-emc-namespace:ns1' http://10.249.245.187:9020/bucket5 -v
```

Atmos

Atmos creates a subtenant request header, `x-emc-retention-start-delay`, captures the autocommit interval.

```
./atmoscurl.pl -user USER1 -action PUT -pmode TID -path / -header "x-emc-retention-
period:300" -header "x-emc-retention-start-delay:120" -include
```

Behavior of file operations

Table 19. Behavior of file operations

File Operation	Expected within autocommit period	Expected within retention period (after autocommit period)
Change permission of file	Allowed	Denied
Change ownership of file	Allowed	Denied
Write to existing file	Allowed	Denied
Create empty file	Allowed	Allowed
Create non-empty file	Allowed	Denied
Remove file	Allowed	Denied
Move file	Allowed	Denied
Rename file	Allowed	Denied
Make dir	Allowed	Allowed
Remove directory	Denied	Denied
Move directory	Denied	Denied
Rename directory	Denied	Denied

Table 19. Behavior of file operations (continued)

File Operation	Expected within autocommit period	Expected within retention period (after autocommit period)
Change permission on directory	Denied	Denied
list	Allowed	Allowed
Read file	Allowed	Allowed
Truncate file	Allowed	Denied
Copy of local read-only files to NFS share	Allowed	Allowed
Copy of read-only files from NFS share to NFS share	Allowed	Allowed
Change atime/mtime of file/ directory	Allowed	Denied

S3A support

The AWS S3A client is a connector for AWS S3.

S3A client enables you to run Hadoop MapReduce or Spark jobs with ECS S3. For information about Hadoop S3A, see Hadoop S3A for ECS.

NOTE:
- ECS does not enable you to run S3A client on FS enabled buckets.
- S3A support is available on Hadoop 2.7 or later version.

Geo-replication status

The ECS S3 head supports Geo replication status of an object with replicationInfo. It API retrieves Geo replication status of an object using replicationInfo. This automates their capacity management operations, enable site reliability operations and ensures that the critical date is not deleted accidently.

Retrieves Geo replication status of an object by API to confirm that the object has been successfully replicated.

```
Request:
GET  /bucket/key?replicationInfo

Response:

<ObjectReplicationInfo xmlns="http://s3.amazonaws.com/doc/
2006-03001/"
  <IndexReplicated>false</IndexReplicated>
  <ReplicatedDataPercentage>64.0</ReplicatedDataPercentage>
</ObjectReplicationInfo>
```

Configuring throttle limit during bucket creation

ECS allows you to configure throttle limit to control the amount of HTTP requests on a specific resource over a given period.

To limit PUT requests on every bucket in the system per second, configure the Configuration Framework (CF) variables as below:

```
com.emc.ecs.common.request.throttle.enabled = true
com.emc.ecs.common.request.throttle.limit = <any postive integer value>
com.emc.ecs.common.request.throttle.type = resource:bucket
com.emc.ecs.common.request.throttle.method = PUT
```

NOTE: The throttle limit value must be a positive integer value. That is any value between <1 - java long range upper limit> is valid.

S3 Select

S3 select uses simple SQL expressions to retrieve a subset of data from an object, which enables your application to retrieve only the required data. This feature may improve your application performance.

Limitations

S3 Select has the following limitations:

- By default, S3 select is enabled for the 192 GB memory profiles and disabled for the 64 GB profiles.
- Only CSV, JSON, and parquet file formats are supported.
- S3 Select supports only the SELECT SQL command.

S3 Select API

ECS supports the `SelectObjectContent` API to retrieve a subset of data from an object using SQL expressions.

API	Method	Request headers	Response

`SelectObjectContent`	`POST`	`x-amz-server-side-encryption-customer-algorithm`
		`x-amz-server-side-encryption-customer-key`
		`x-amz-server-side-encryption-customer-key-MD5`
			• For all the success scenarios, the system returns an `Http response : 200 OK` and obtains the requested subset.
			• For all the error scenarios, the system returns proper error messages along with error codes.

Metadata Search

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

Metadata can be associated with objects using the ECS S3 API. If you know the identity of an object, you can read an object's metadata. Without the ECS metadata search feature, it is not possible to find an object using 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, clients write the user metadata with reference to the user requirements. Both system and user metadata can be indexed and used as the basis for metadata searches. The number of metadata values that can be indexed is limited to 30 and must be defined when the bucket is created.

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

When querying objects based on their indexed metadata, the objects that match the query and the values of their indexed metadata are returned. You can also choose to return all of the system and/or user metadata that is associated with the returned objects. In addition to system metadata, objects also have attributes which can be returned as part of metadata search results. The system metadata values that are available and can be indexed, and the metadata values that can optionally be returned with search query results, are listed ECS system metadata and optional attributes.

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

- Assign metadata index values to a bucket

- Assign metatdata to objects using S3 protocol
- Use metadata search queries

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.

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

You can set 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.

Setting index values using the ECS Management REST API

You can set index values using the ECS Management REST API

Table 20. ECS Management REST API methods

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 are 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 S3 API provides methods for working with indexes that are listed in the following table and links are provided to the API reference.

NOTE: The following characters are not accepted in S3 metadata key in ECS 3.4 and later versions: quotation marks (`""`), parentheses (`()`), comma (`,`), Forward slash (`/`), at (`@`), angle brackets (`<>`), equal to (`=`), and question mark (`?`).

Table 21. ECS Management REST API methods

API Path	Description
GET /?searchmetadata	Lists the names of all system metadata available for indexing on new buckets.

Table 21. ECS Management REST API methods (continued)

API Path	Description
PUT /{bucket} -H x-emc-metadata-search: {name[;datatype],...}	Creates a bucket with the search metadata key that is indicated in the header. **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 following example 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 an object with x-amz-meta-, values containing special characters do not have to be url-encoded.

Using encryption with metadata search

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

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

Assign metadata to objects using the S3 protocol

End users can assign user metadata to objects using the `x-amz-meta-` header. The value assigned can be any text string and is case sensitive, but the metadata names are case insensitive, so `x-amz-meta-FOO`, `x-amz-meta-foo` are referring to the same field `foo`.

NOTE: When defining the fields to index and searching, ensure that you use all lowercase.

When the metadata is indexed so that it can be used as the basis of object searches (the metadata search feature), a datatype 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 datatypes are:

String	If the search index term is marked as text, the metadata string is treated as a string in all search comparisons.
Integer	If the search index term is marked as integer, the metadata string is converted to an integer in search comparisons.
Decimal	If a search index term is marked as decimal, the metadata string is 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 is 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.

Table 22. API Syntax

API Syntax	Response Body

`GET /{bucket}/?`
`query={expression}`
`&attributes={fieldname,`
`…}`
`&sorted={selector}`
`&include_older_version=`
`{tru`
`e|false}`
`&max-keys=(num_keys)`
`&marker=(marker value)`
 	
NOTE: Prefix capability is added to the metadata search. See Prefix capability in metadata search.
	`<BucketQueryResult xmlns:ns2="http://`
	`s3.amazonaws.com/doc/2006-03-01/">`
	`  <Name>mybucket</Name>`
	`  <Marker/>`
	`  <IsTruncated>false</IsTruncated>`
	`  <MaxKeys>0</MaxKeys>`
	`  <ObjectMatches>`
	`    <object>`
	`      <objectName>file4</objectName>`
	
	`<objectId>09998027b1b7fbb21f50e13fabb48`
	`1a237ba2f60f352d437c8da3c7c1c8d7589</`
	`objectId>`
	`      <versionId>0</versionId>`
	`      <queryMds>`
	`        <type>SYSMD</type>`
	`        <mdMap>`
	`          <entry>`
	`            <key>createtime</key>`
	`            <value>1449081778025</`
	`value>`
	`          </entry>`
	`          <entry>`
	`            <key>size</key>`
	`            <value>1024</value>`
	`          </entry>`
	`          <entry>`
	`            <key>mtime</key>`
	`            <value>1449081778025</`
	`value>`
	`          </entry>`
	`        </mdMap>`
	`      </queryMds>`
	`      <queryMds>`
	`        <type>USERMD</type>`
	`        <mdMap>`
	`          <entry>`
	`            <key>x-amz-meta-INT</key>`
	`            <value>407</value>`
	`          </entry>`
	`          <entry>`
	`            <key>x-amz-meta-STR</key>`
	`            <value>String4</value>`
	`          </entry>`
	`        </mdMap>`
	`      </queryMds>`
	`      <indexKey/>`
	`    </object>`
	`    <object`
	`    ...`
	`    </object>`

Table 22. API Syntax

API Syntax	Response Body

	`</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 > 2018-03-01T11:22:00Z`
selector	A searchable key name associated with the bucket.
operator	An operator. One of: ==, >, <, <=, >=
argument	A value that the selector is tested against.
attributes=[field name,...]	Specifies any optional object attributes that should be included in the report. Attribute values will be included in the report where that attribute is present on the object. The optional attribute values comprise: ● ContentEncoding ● ContentType ● Retention ● Expiration ● Expires In addition, it is possible to return the non-indexed metadata associated with objects that are returned by the search query. The following: **ALL** Lists both system and user metadata associated with the returned objects. **ALL_SMD** Lists the system metadata associated with the returned objects. **ALL_UMD** Lists the user metadata associated with the returned objects.
sorted=[selector]	Specifies one searchable key name associated with the bucket. The key name must be a key that appears in the expression. In the absence of &sorted=keyname, the output will be sorted according to the first key name that appears in the query expression. **NOTE:** If "or" operators are used in the expression, the sort order is indeterminate.
include-older-versions=[true|false]	When S3 versioning is enabled on a bucket, setting this to true will return current and older versions of objects that match the expression. Default is false.
max-keys	The maximum number of objects that match the query that should be returned. If there are more objects than the max-keys, a marker will be returned that can be used to retrieve more matches.
marker	The marker that was returned by a previous query and that indicates the point from which query matches should be returned.

Datetime queries

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

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

User metadata upload header example:	`-H x-amz-meta-Foo:2018-03-06T12:00:00Z`
User and System query expression format:	`?query=CreateTime>2018-01-01T00:00:00Z and x-amz-meta-Foo==2018-03-06T12:00:00Z`
Query results fragment - system metadata	`<key>createtime</key> <value>1449081777620</value>`
Query results fragment - user metadata	`<key>x-amz-meta-Foo</key> <value>2018-03-06T12:00:00Z</value>`

Using markers and max-keys to paginate results

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

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

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

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

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

```
?query=CreateTime>2018-01-01T00:00:00Z and x-amz-meta-Foo==2018-03-06T12:00:00Z&max-
keys=3&marker=rO0ABXNyAD...
```

When the returned objects are the final page of objects, the `IsTruncated` tag value is returned as `false`. Else, the `IsTruncated` tag value is returned as `True`.

NOTE: Only if the `IsTruncated` tag value is returned as `True`, the `NextMarker` tag appears.

Using Partial Results

In large bucket indexes, complex queries can have a long run time. If the search cannot accumulate a full page (max-keys) of results within the timeout, you can use the `allow-partial-results` parameter to let ECS return a partial page of results.

For example:

```
?query=CreateTime>2018-01-01T00:00:00Z and x-amz-meta-Foo==2018-03-06T12:00:00Z&max-
keys=1000&allow-partial-results=true
```

indicates that you would like 1000 keys in the response, but ECS may return less than 1000 if it cannot find 1000 matching objects within the default timeout window. The partial result, like a normal page result, has the `IsTruncated` flag set to `true` with a token in the `NextMarker` field to continue the search.

Using special characters in queries

You can use 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, semicolon).

  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.

Prefix capability in metadata search

You can use prefix capability in metadata search

S3 API metadata search supports the prefix and delimiter parameters. It follows the standard S3 definition of these parameters. Prefix capability effectively transforms every single metadata query into a multi query request with AND operation between prefix and the query string. In other words, it is possible to combine the AND and OR predicates in the queries.

S3 API metadata is modified to support prefix and delimiter parameters as described below:

```
GET /bucketName/?prefix={prefix}&delimiter={delimiter}&query={queryString}
```

Limitations

- A prefix is always applied before the actual query.
- Custom sorting is not supported with prefixes. If sorting is specified together with a prefix, the API returns 400 Bad Request.
- Objects are returned in lexicographical order.
- Using `ObjectName` in a query string together with a prefix is not allowed. It creates ambiguity as both filter objects based on name. If both are specified, the API returns 400 Bad Request.

Metadata search example

You can use 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/>
  <IsTruncated>false</IsTruncated>
  <MaxKeys>0</MaxKeys>
  <ObjectMatches>
    <object>
```

```
      <objectName>file4</objectName>
      <objectId>09998027b1b7fbb21f50e13fabb481a237ba2f60f352d437c8da3c7c1c8d7589</
objectId>
      <versionId>0</versionId>
      <queryMds>
        <type>SYSMD</type>
        <mdMap>
          <entry>
            <key>createtime</key>
            <value>1449081778025</value>
          </entry>
          <entry>
            <key>size</key>
            <value>1024</value>
          </entry>
          <entry>
            <key>mtime</key>
            <value>1449081778025</value>
          </entry>
        </mdMap>
      </queryMds>
      <queryMds>
        <type>USERMD</type>
        <mdMap>
          <entry>
            <key>x-amz-meta-INT</key>
            <value>407</value>
          </entry>
          <entry>
            <key>x-amz-meta-STR</key>
            <value>String4</value>
          </entry>
        </mdMap>
      </queryMds>
      <indexKey/>
    </object>
    <object>
      <objectName>file5</objectName>
      <objectId>1ad87d86ef558ca0620a26855662da1030f7d9ff1d4bbc7c2ffdfe29943b9150</
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.

Table 23. SDK Support for Metadata Search

-	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

Table 24. System Metadata

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

Optional metadata attributes

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

Table 25. Optional metadata attributes

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

Metadata search with Tokenization

Tokenization allows you to use metadata search to search for objects that have a specific metadata value within an array of metadata values.

Tokenization is enabled using the `x-emc-metadata-search-tokens: true` header. When creating a bucket, you can specify whether the metadata value should follow the existing method (without tokenization) or the new method (with tokenization). However, both methods cannot be used on the same bucket.

Methods	**Description**
Existing method without tokenization	In this method, the metadata value is interpreted as a single value. For example, in the `x-amz-meta-countries= [france,uk]` command, the france,uk value is considered as a single value.
New method with tokenization	In this method, each element of a metadata value is searchable. For example, in the `x-amz-meta-countries= [france,uk]` command, the france,uk value is considered as two values separated with the delimiter.

Limitations

Tokenization has the following limitations:

Limitation	**Description**
Size of user metadata	User-defined metadata is limited to 2 KB in size for AWS. For ECS 2 KB is a guideline only. The size of user-defined metadata is the sum of the number of bytes in the UTF-8 encoding of each key and value. Note that metadata will not work on other platforms such as AWS.
Permitted delimiters for tokenization	The special characters that can be used to identify individual values in the metadata value are "`[`", "`]`", and "`,`".
Backward compatibility	Token-based search is not available for existing buckets.

S3 and Swift Interoperability

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

When considering whether objects created using the S3 head is accessible using the Swift head, and conversely, you should first consider whether users can access the bucket (called a container in Swift). A bucket is assigned a bucket type (S3 or Swift, for example) based on the ECS head that created it. The object users must have appropriate permissions for the type of bucket, for an application to access both Swift and S3 buckets. Consider giving the permissions, because of the way in which permissions are determined for Swift and S3 buckets is different.

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

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

Access to a resource is determined either by being the bucket owner, or by being assigned permission on the bucket using ACLs. When a S3 user creates a bucket, for example, that bucket is owned by the S3 user name. That user has full permissions on the bucket, and a Swift user with the same name similarly has full permissions on the bucket.

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

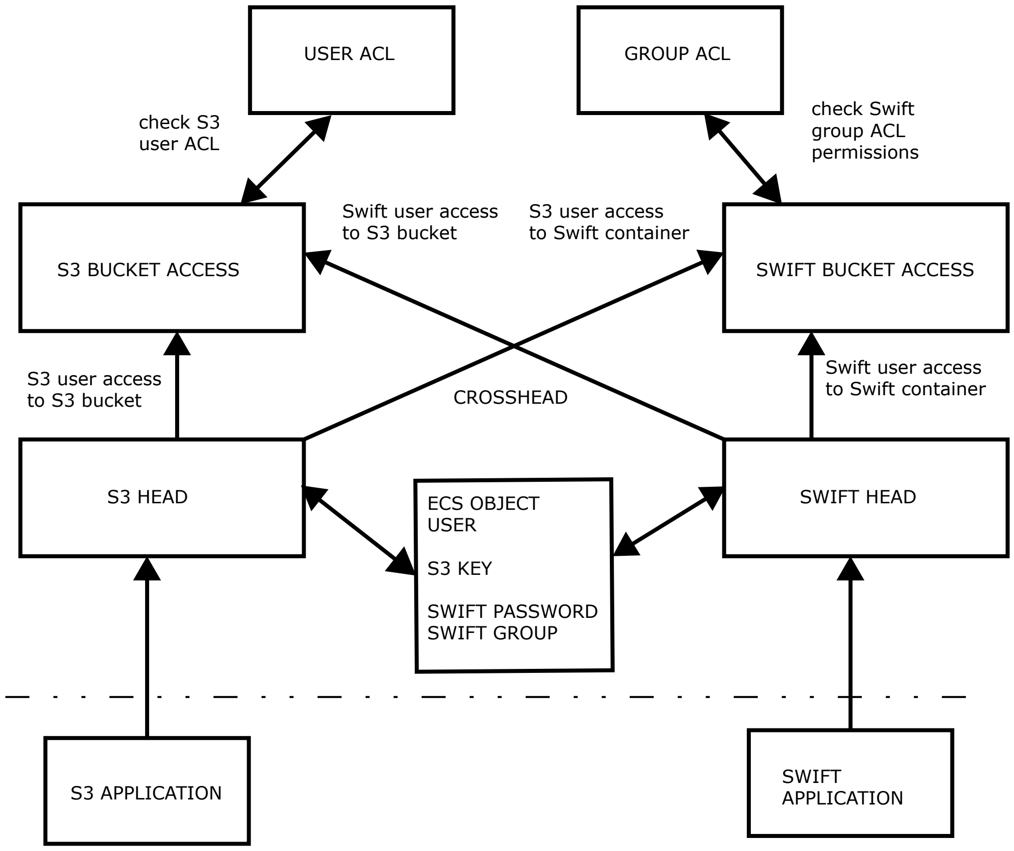


Figure 1. S3 user access checks

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

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

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

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

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

NOTE:

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

NOTE:

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

Create and manage secret keys

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

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

- An administrator creates a key and distributes to the object user (Create a key for an object user).
- A domain user creates an object user account by creating a new secret key using the self-service API provided by the self-service API (Create an S3 secrte key: self-service).

It is possible to have two 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 are 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
- Create an S3 secret key using the ECS Management REST API

For more information about ECS users, see the ECS Administration Guide which is available from the https://www.dell.com/support/.

Generate a secret key from the ECS Portal

You can generate a secret key at the ECS Portal.

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

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

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

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

Table 26. API Path

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. A Namespace Administrator can create keys for users in their namespace. A System Administrator can assign keys to users in any namespace. To change a key, a second key can be assigned and the time at which the first key expires can be specified.

You can find out more information about the API call in the ECS API Reference.

Create an S3 secret key: self-service

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

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

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

Table 27. Object Secret Keys

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

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

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

- Log in as domain user
- Generate first key
- Generate second key
- Check keys
- Delete all secret keys

Log in as a domain user

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

```
curl -ik -u user@mydomain.com:<Password> https://10.241.48.31:4443/login
HTTP/1.1 200 OK
Date: Mon, 05 Mar 2018 17:29:38 GMT
Content-Type: application/xml
Content-Length: 107
Connection: keep-alive
X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwNzQ4ODA1NTQDAC
51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8=

<?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
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{}"
https://10.241.48.31:4443/object/secret-keys | xmllint --format -

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

Generate second key

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

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAwN
zQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{\"existing_key_expiry_time_mins\":
\"10\"}"
https://10.241.48.31:4443/object/secret-keys | xmllint --format -

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

Check keys

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

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAw
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
https://10.241.48.31:4443/object/secret-keys | xmllint --format -
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<user_secret_keys>
  <secret_key_1>7hXZ9/EHTVvmFuYly/z3gHpihXtEUX/VZxdxDDBd</secret_key_1>
  <secret_key_2>l3fPCuFCG/bxoOXCPZoYuPwhXrSTwU0f1kFDaRUr</secret_key_2>
  <key_expiry_timestamp_1>2018-03-05 17:50:12.369</key_expiry_timestamp_1>
  <key_expiry_timestamp_2/>
  <key_timestamp_1>2018-03-05 17:39:13.813</key_timestamp_1>
  <key_timestamp_2>2018-03-05 17:40:12.506</key_timestamp_2>
  <link rel="self" href="/object/secret-keys"/>
</user_secret_keys>
```

Delete all secret keys

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

```
curl -ks -H "X-SDS-AUTH-TOKEN: BAAcaVAzNU16eVcwM09rOWd2Y1ZoUFZ4QmRTK2JVPQMAQQIADTE0MzAw
NzQ4ODA1NTQDAC51cm46VG9rZW46YWJmODA1NTEtYmFkNC00ZDA2LWFmMmMtMTQ1YzRjOTdlNGQ0AgAC0A8="
-H "Content-Type: application/json" -X POST -d "{}" https://10.241.48.31:4443/object/
secret-keys/deactivate
```

Authenticating with the S3 service

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

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

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

Authentication using Signature V2 and Signature V4 are introduced in:

- Authenticating using Signature V2
- Authenticating using Signature V4

The following notes apply:

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

Authenticating using Signature V2

The Authorization header when using Signature V2 looks like this:

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

For example:

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

Authentication using Signature V2 is described in:

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

Authenticating using Signature V4

The Authorization header when using Signature V4 looks like this:

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

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

Authentication using Signature V4 is described in:

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

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

```
PUT /bucket_demo HTTP/1.1
x-amz-date: 20160726T033659Z
```

```
Authorization: AWS4-HMAC-
SHA256 Credential=user11/20160726/us/s3/aws4_request,SignedHeaders=host;x-amz-date;x-emc-
namespace,Signature=e75a150daa28a2b2f7ca24f6fd0e161cb58648a25121d3108f0af5c9451b09ce
x-emc-namespace: ns1
x-emc-rest-client: TRUE
x-amz-content-sha256: 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
```

Using s3curl with ECS

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

When using ECS custom headers (x-emc), the signature element of the `Authorization` header must be constructed to include the custom headers. In addition,

- When connecting to ECS 3.0 and later, the `search` and `searchmetadata` parameters are part of the signature computation.
- When connecting to ECS 3.5.0.2 and later, the `x-amz-date` parameter is part of the signature computation.

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

Use SDKs to access the S3 service

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

The ECS Community provides information about the various clients that are available and provides guidance on their use: ECS Community: Developer Resources.

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

- Using the Java Amazon SDK>
- Java SDK client for ECS

NOTE: If you want to make use of the ECS REST API Extensions, support for these extensions is provided in the ECS Java SDK. If you do not need support for the ECS extensions, or you have existing applications that use it, you can use the Amazon Java SDK.

NOTE: Compatibility of the ECS Java SDK with the metadata search extension is described in Using Metadata Search from the ECS Java SDK.

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 attempts 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 data center 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());
        }
    }
}
```

ECS S3 APIs compatible with AWS Java SDK

The below table lists the supported ECS S3 APIs that are compatible with AWS Java SDK.

NOTE:

- All these APIs are certified against AWS Java SDK version 1.11.769.
- The Object Tagging APIs support is available from ECS 3.5.

Feature	Subfeature	API
Object	Object tagging	SetObjectTaggingRequest
Object	Object tagging	GetObjectTaggingRequest
Object	Object tagging	DeleteObjectTaggingRequest
Object	deleteobject	DeleteObjectRequest
Object	deleteobject	DeleteObjectResult
Object	deleteobject	DeleteObjectsRequest
Object	deleteobject	DeleteObjectsResult
Object	getobject	S3Object
Object	getobject	GetObjectRequest
Object	getobject	GetObjectResult
Object	getobject	S3VersionSummary
Object	getobject	ObjectSummary
Object	getobject	ListVersionsRequest
Object	getobjectacl	GetObjectAclRequest
Object	headobject	GetObjectMetadataRequest
Object	headobject	ObjectMetadata
Object	putobject	PutObjectResult
Object	putobject	PutObjectRequest
Object	putobjectACL	SetObjectAclRequest
Object	putobjectACL	CanonicalGrantee
Object	putobjectcopy	CopyObjectRequest
Object	putobjectcopy	CopyObjectResult
Bucket	DeleteBucket	Bucket
Bucket	DeleteBucket	DeleteBucketRequest
Bucket	DeleteBucket	DeleteBucketLifecycleConfigurationRequest

Feature	Subfeature	API
Bucket	DeleteBucket	BucketLifecycleConfiguration
Bucket	GetBucket	GetBucketLocationRequest
Bucket	GetBucket	GetBucketLifecycleConfigurationRequest
Bucket	GetBucket	GetBucketVersioningConfigurationRequest
Bucket	GetBucketACL	GetBucketAclRequest
Bucket	GetBucketObjectVersions	VersionListing
Bucket	GetBucketVersioning	BucketVersioningConfiguration
Bucket	BucketOperations	CreateBucket
Bucket	BucketVersioning	SetBucketVersioningConfigurationRequest
Bucket	ACL	PUTBucketACL
Bucket	ACL	SETBucketACL
Bucket	ACL	GETBucketACL
Bucket	BucketPolicy	SetBucketPolicyRequest
Bucket	BucketPolicy	GetBucketPolicyRequest
Bucket	BucketOperations	DeleteBucketPolicyRequest
Bucket	BucketOperations	ListObjectsRequest
Bucket	BucketOperations	HeadBucketRequest
Bucket	BucketOperations	HeadBucketResult
multipartupload	abortmultipartupload	AbortMultipartUploadRequest
multipartupload	abortmultipartupload	AbortMultipartUploadResult
multipartupload	completemultipartupload	CompleteMultipartUpload
multipartupload	completemultipartupload	CompleteMultipartUploadRequest
multipartupload	completemultipartupload	CompleteMultipartUploadResul
multipartupload	initiatemultipartupload	InitiateMultipartUploadRequest
multipartupload	initiatemultipartupload	InitiateMultipartUploadResult
multipartupload	listmultipartuploads	ListMultipartUploadsRequest
multipartupload	listmultipartuploads	ListMultipartUploadsResult
multipartupload	listparts	ListPartsRequest
multipartupload	listparts	PartListing
multipartupload	listparts	PartSummary
multipartupload	listparts	ListPartsResult
multipartupload	uploadpart	UploadPartRequest
multipartupload	uploadpart	UploadPartResult
multipartupload	uploadpartcopy	CopyPartRequest
multipartupload	uploadpartcopy	CopyPartResult
Service	GetService	BucketListing
CORS	GETBucketCORS	BucketCrossOriginConfiguration
CORS	GETBucketCORS	GetBucketCrossOriginConfigurationRequest

Feature	Subfeature	API
CORS	DELETEBucketCORS	DeleteBucketCrossOriginConfigurationRequest
CORS	PUT Bucket CORS	CORSRule
CORS	PUT Bucket CORS	BucketCrossOriginConfiguration
CORS	PUT Bucket CORS	SetBucketCrossOriginConfigurationRequest

AWS SDK APIs not supported in ECS S3 APIs

The below table lists the AWS SDK APIs that are not supported in ECS S3.

NOTE: All these APIs are certified against AWS Java SDK version 1.11.769.

Feature	API
Bucket Analytics	DeleteBucketAnalyticsConfiguration
Bucket Analytics	GetBucketAnalyticsConfiguration
Bucket Analytics	ListBucketAnalyticsConfigurations
Bucket Analytics	PutBucketAnalyticsConfiguration
Bucket Replication	PutBucketReplication
Bucket Replication	GetBucketReplication
Bucket Replication	DeleteBucketReplication
Bucket encryption	DeleteBucketEncryption
Bucket encryption	GetBucketEncryption
Bucket encryption	PutBucketEncryption
Bucket inventory	DeleteBucketInventoryConfiguration
Bucket inventory	GetBucketInventoryConfiguration
Bucket inventory	ListBucketInventoryConfigurations
Bucket inventory	PutBucketInventoryConfiguration
Bucket Metric	DeleteBucketMetricsConfiguration
Bucket Metric	GetBucketMetricsConfiguration
Bucket Metric	List Bucket Metrics Configurations
Bucket Metric	PutBucketMetricsConfiguration
Bucket website	DeleteBucketWebsite
Bucket website	GetBucketWebsite
Bucket website	PutBucketWebsite
PublicAccessBlock	DeletePublicAccessBlock
PublicAccessBlock	GetPublicAccessBlock
PublicAccessBlock	PutPublicAccessBlock
Bucket Accelarate	GetBucketAccelerateConfiguration
Bucket Accelarate	PutBucketAccelerateConfiguration
Bucket Logging	GetBucketLogging
Bucket Logging	PutBucketLogging

Feature	API
BucketRequestPayment	GetBucketRequestPayment
BucketRequestPayment	PutBucketRequestPayment
Bucket policy	GetBucketPolicyStatus
Object Torrent	GetObjectTorrent
Restore Object	RestoreObejct
Object Content	SelectObjectContent
Object legal hold	SetObjectLegalHoldRequest
Object legal hold	ObjectLockLegalHold
Object legal hold	ObjectLockLegalHoldStatus
Object legal hold	SetObjectLegalHoldResult
Object legal hold	GetObjectLegalHoldRequest
Object legal hold	GetObjectLegalHoldResult
Object retention	SetObjectRetentionRequest
Object retention	ObjectLockRetention
Object retention	ObjectLockRetentionMode
Object retention	SetObjectRetentionResult
Object retention	GetObjectRetentionRequest
Object retention	GetObjectRetentionResult

ECS Java SDK

The ECS Java SDK is built on the Jersey REST client, and it supports the ECS API extensions.

An example of using this SDK (S3Client) is shown below.

```
package com.emc.ecs.s3.sample;

import com.emc.object.s3.*;
import com.emc.object.s3.jersey.S3JerseyClient;

import java.net.URI;

public class S3ClientSample {
    public static void main(String[] args) throws Exception {
        URI endpoint = new URI("http://ecs.yourco.com:9020");
        String accessKey = "fred@yourco.com";
        String secretKey = "pcQQ20rDI2DHZOIWNkAug3wK4XJP9sQnZqbQJev3";

        S3Config config = new S3Config(endpoint);
        config.withIdentity(accessKey).withSecretKey(secretKey);

        S3Client s3Client = new S3JerseyClient(config);
        S3ClientSample sample = new S3ClientSample(s3Client);

        sample.runSample();
    }

    private final S3Client s3Client;

    public S3ClientSample(S3Client s3Client) {
        this.s3Client = s3Client;
    }

    public void runSample() {
        String bucketName = "mybucket";
        String key1 = "test1.txt";
        String content = "Hello World!";

        try {
            s3Client.createBucket(bucketName);
            s3Client.putObject(bucketName, key1, content, "text/plain");
        } catch (S3Exception e) {
            // handle errors
        }
    }
}
```

Disabling request timeouts

In general, using request timeouts is discouraged to prevent the possibility of lost updates (a general issue with HTTP client timeouts). When a PUT request times out, it may eventually perform on the server side. Before this delayed execution, if there is another PUT request for the same object, that update is overwritten by the delayed first request (this process causes a DL risk). To avoid this possibility, you can disable request timeouts in the client. This process avoids any possibility of an unexpected delayed execution.

NOTE: Disabling request timeouts does not prevent concurrent issues. If there are multiple threads attempt to PUT the same object, the system executes the last update. It is the responsibility of an application to provide any locking mechanisms for concurrent access.

- In ECS Java SDK Client version 3.1.3 and below, the timeout parameter value is set as one minute, which has a potential DL risk of lost updates.
- In ECS Java SDK Client version 3.2.0 and above, the timeout parameter is disabled by default.

From ECS 3.5.1.4, as an alternative to disabling the timeout parameter, you can add `IfNoneMatch` and `IfMatch` parameters in the request header. This can be used as an optimistic locking mechanism.

For example,

- You can add headers using the below command. This command creates the object when there is no such objects.

  ```
  request.withIfMatch(null).withIfNoneMatch("*")
  ```

- If you want to update an object sequentially, you can use the below command to ensure that the update follows the previous update.

  ```
  request.withIfUnmodifiedSince(null).withIfMatch(lastEtag)
  ```

Changing timeout parameters

You can set the read timeout parameter using this command:

```
// 3.2.0 and above
// milliseconds = 0 means disabled
s3Config.setReadTimeout(milliseconds);
//3.1.3 and below
// milliseconds = 0 means disabled
s3config.setProperty(ClientConfig.PROPERTY_READ_TIMEOUT, milliseconds);
```

For more information about the risk of lost updates or changing timeouts, see https://github.com/EMCECS/ecs-object-client-java/wiki/Changing-Timeouts.

ECS S3 error codes

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

Table 28. Error Codes

Error Code	HTTP Status Code	Generic Error Code	Description Error
AccessDenied	403	AccessDenied	Access Denied
BadDigest	400	BadDigest	The Content-MD5 you specified did not match that received.
BadRequest	400	BadRequest	Bucket delete not supported.
BucketAlreadyExists	409	BucketAlreadyExists	The requested bucket name is not available. The bucket namespace is shared by all users of the system. Please select a different name and try again.
BucketNotEmpty	409	BucketNotEmpty	The bucket you tried to delete is not empty.
ContentMD5Empty	400	InvalidDigest	The Content-MD5 you specified was invalid.
ContentMD5Missing	400	InvalidRequest	The required Content-MD5 header for this request is missing.
EntityTooSmall	400	EntityTooSmall	The proposed upload is smaller than the minimum allowed object size.
EntityTooLarge	400	EntityTooLarge	The proposed upload exceeds the maximum allowed object size.
IncompleteBody	400	IncompleteBody	The number of bytes specified by the `Content-Length` HTTP header were not provided.
InternalError	500	InternalError	An internal error was encountered. Please try again.
ServerTimeout	500	ServerTimeout	An internal timeout error was encountered. Please try again.
InvalidAccessKeyId	403	InvalidAccessKeyId	The Access Key Id you provided does not exist.
InvalidArgument	400	InvalidArgument	Invalid Argument.
NoNamespaceForAnonymous Request	403	AccessDenied	ECS could not determine the namespace from the anonymous request. Please use a namespace BaseURL or include an `x-emc-namespace` header.
NotSupported	403	NotSupported	Background bucket deletion is not supported for this bucket.
InvalidBucketName	400	InvalidBucketName	The specified bucket is not valid.
InvalidDigestBadMD5	400	InvalidDigest	The Content-MD5 you specified was invalid.
InvalidDigest	403	SignatureDoesNotMatch	The Content-MD5 you specified was an invalid.
InvalidRequest	400	InvalidRequest	Invalid Request.

Table 28. Error Codes (continued)

Error Code	HTTP Status Code	Generic Error Code	Description Error
InvalidPart	400	InvalidPart	One or more of the specified parts could not be found. The part might not have been uploaded.
InvalidPartOrder	400	InvalidPartOrder	The list of parts was not in ascending order. Parts list must specified in order by part number.
InvalidPartSizeZero	400	InvalidPartSizeZero	The upload part size cannot be zero.
MissingEncryption	400	InvalidRequest	The multipart upload initiate requested encryption. Subsequent part requests must include the appropriate encryption parameters.
NoEncryptionNeed	400	InvalidRequest	The multipart initiate request did not request encryption. Please resend the request without sending encryption parameters.
BadMD5	400	InvalidRequest	The calculated MD5 hash of the key did not match the hash that was provided.
BadEncryptKey	400	InvalidRequest	The provided encryption parameters did not match the ones used originally.
InvalidRange	416	InvalidRange	The requested range cannot be satisfied.
KeyTooLong	400	KeyTooLong	The specified key is too long.
MalformedACLError	400	MalformedACLError	The XML provided was not well-formed or did not validate against the ECS published schema.
MalformedXML	400	MalformedXML	Malformed xml (that does not conform to the published xsd) for the configuration was sent.
MaxMessageLengthExceeded	400	MaxMessageLengthExceeded	The request was too big.
MetadataTooLarge	400	MetadataTooLarge	The metadata headers exceed the maximum allowed metadata size. *
InvalidProject	400	InvalidProject	The specified project is Invalid.
InvalidVPool	400	InvalidVPool	The specified vPool (Replication Group) is Invalid.
InvalidNamespace	400	InvalidNamespace	The specified namespace is Invalid.
MethodNotAllowed	405	MethodNotAllowed	The specified method is not allowed against this resource.
MissingContentLength	411	MissingContentLength	The Content-Length HTTP header must be provided.
MissingRequestBodyError	400	MissingRequestBodyError	An empty XML document was sent. The error message is: Request body is empty.
MissingSecurityHeader	400	MissingSecurityHeader	The equest was missing a required header.

Table 28. Error Codes (continued)

Error Code	HTTP Status Code	Generic Error Code	Description Error
IncompleteLifecycleConfig	400	IncompleteLifecycleConfig	At least one action needs to be specified in a rule.
MalformedLifecycleConfig	400	MalformedLifecycleConfig	The XML provided was not well-formed or did not validate against the published schema.
MalformedDateLifecycleConfig	400	MalformedDateLifecycleConfig	The XML provided was not well-formed or did not validate against the published schema. Invalid Date or Days.
NoSuchBucket	404	NoSuchBucket	The specified bucket does not exist.
NoSuchBucketPolicy	404	NoSuchBucketPolicy	The bucket policy does not exist.
NoSuchKey	404	NoSuchKey	The specified key does not exist.
NoSuchRetention	404	NoSuchRetention	The specified retention does not exist.
ObjectUnderRetention	409	ObjectUnderRetention	The object is under retention and cannot be deleted or modified.
NoSuchUpload	404	NoSuchUpload	The specified multipart upload does not exist. The upload ID might be invalid.
NotImplemented	501	NotImplemented	The requested functionality is not implemented.
OperationAborted	409	OperationAborted	A conflicting conditional operation is currently in progress against this resource. Please try again.
PermanentRedirect	301	PermanentRedirect	The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint.
PreconditionFailed	412	PreconditionFailed	At least one of the preconditions you specified did not hold.
RequestIsNotMultiPartContent	400	RequestIsNotMultiPartContent	Bucket POST must be of the enclosure type `multipart/form-data`.
RequestTimeout	400	RequestTimeout	The socket connection to the server was not read from or written to within the timeout period.
RequestTimeTooSkewed	403	RequestTimeTooSkewed	The difference between the request time and the server's time is too large.
DateIsRequired	403	AccessDenied	A valid Date or `x-amz-date` header is required.
SignatureDoesNotMatch	403	SignatureDoesNotMatch	The request signature calculated does not match the signature provided. Check the Secret Access Key and signing method.
ZeroAmzExpires	403	Forbidden	Zero value specified for `x-amz-expires`.

Table 28. Error Codes (continued)

Error Code	HTTP Status Code	Generic Error Code	Description Error
InvalidAmzExpires	400	Bad Request	Invalid value specified for `x-amz-expires`.
ServiceUnavailable	503	ServiceUnavailable	Please reduce your request rate.
TemporaryRedirect	307	TemporaryRedirect	Requests are being redirected to the bucket while DNS updates.
TooManyBuckets	400	TooManyBuckets	The request attempted to create more buckets than allowed.
UnexpectedContent	400	UnexpectedContent	The request does not support this content.
UnresolvableGrantByEmailAddress	400	UnresolvableGrantByEmailAddress	The email address you provided does not match any account on record.
InvalidBucketState	409	InvalidBucketState	The request is not valid with the current state of the bucket.
SlowDown	503	SlowDown	Please reduce your request rate.
AccountProblem	403	AccountProblem	There is a problem with the specified account that prevents the operation from completing successfully.
CrossLocationLoggingProhibited	403	CrossLocationLoggingProhibited	Cross location logging is not allowed. Buckets in one geographic location cannot log information to a bucket in another location.
ExpiredToken	400	ExpiredToken	The provided token has expired.
IllegalVersioningConfiguration Exception	400	IllegalVersioningConfiguration Exception	The Versioning configuration specified in the request is invalid.
IncorrectNumberOfFilesInPost Request	400	IncorrectNumberOfFilesInPost Request	POST requires exactly one file upload per request.
InvalidAddressingHeader	500	InvalidAddressingHeader	The specified role must be Anonymous role.
InvalidLocationConstraint	400	InvalidLocationConstraint	The specified location constraint is not valid.
InvalidPolicyDocument	400	InvalidPolicyDocument	The content of the form does not meet the conditions specified in the policy document.
InvalidStorageClass	400	InvalidStorageClass	The storage class you specified is not valid.
InvalidTargetBucketForLogging	400	InvalidTargetBucketForLogging	The target bucket for logging does not exist, is not owned by you, or does not have the appropriate grants for the log delivery group.
InvalidToken	400	InvalidToken	The provided token is malformed or otherwise invalid.
InvalidURI	400	InvalidURI	Unable to parse the specified URI.
MalformedPOSTRequest	400	MalformedPOSTRequest	The body of the POST request is not well-formed `multipart/form-data`.

Table 28. Error Codes (continued)

Error Code	HTTP Status Code	Generic Error Code	Description Error
MaxPostPreDataLengthExceeded Error	400	MaxPostPreDataLengthExceeded Error	The POST request fields preceding the upload file were too large.
NoLoggingStatusForKey	400	NoLoggingStatusForKey	There is no such thing as a logging status subresource for a key.
NoSuchLifecycleConfiguration	404	NoSuchLifecycleConfiguration	The lifecycle configuration does not exist.
NoSuchVersion	404	NoSuchVersion	Indicates that the version ID specified in the request does not match an existing version.
RequestTorrentOfBucketError	400	RequestTorrentOfBucketError	Requesting the torrent file of a bucket is not permitted.
UserKeyMustBeSpecified	400	UserKeyMustBeSpecified	The bucket POST must contain the specified field name. If it is specified please check the order of the fields.
AmbiguousGrantByEmailAddress	400	AmbiguousGrantByEmailAddress	The email address you provided is associated with more than one account.
BucketAlreadyOwnedByYou	409	BucketAlreadyOwnedByYou	The previous request to create the named bucket succeeded and you already own it.
CredentialsNotSupported	400	CredentialsNotSupported	The request does not support credentials.
InlineDataTooLarge	400	InlineDataTooLarge	The inline data exceeds the maximum allowed size.
InvalidPayer	403	InvalidPayer	All access to this object has been disabled.
TokenRefreshRequired	400	TokenRefreshRequired	The provided token must be refreshed.
AccessModeNotSupported	409	AccessModeNotSupported	The bucket does not support file access or the requested access mode is not allowed.
AccessModeInvalidToken	409	AccessModeInvalidToken	The token for the file access switch request is invalid.
NoSuchBaseUrl	400	NoSuchBaseUrl	The specified BaseUrl does not exist.
NoDataStoreForVirtualPool	404	NoDataStoreForVirtualPool	No Data Store found for Replication Group of the bucket.
VpoolAccessNotAllowed	400	Cannot Access Vpool	Bucket is hosted on a Replication Group that is not accessible from S3.
InvalidCorsRequest	403	InvalidCorsRequest	Invalid CORS request.
InvalidCorsRule	400	InvalidCorsRule	Invalid CORS rule.
NoSuchCORSConfiguration	404	NoSuchCORSConfiguration	The CORS configuration does not exist.
InvalidAclRequest	404	NoACLFound	The ACL does not exist.
InsufficientStorage	507	Insufficient Storage	The server cannot process the request because there is not enough space on disk.

Table 28. Error Codes (continued)

Error Code	HTTP Status Code	Generic Error Code	Description Error
BadMaxParts	400	InvalidArgument	Argument max-parts must be an integer between 0 and 2147483647.
BucketNotFound	404	NoSuchBucket	The specified bucket does not exist.
NotSupported	400	Not Supported	The bucket may be locked.
InvalidContentLength	400	Invalid content length	The content length has invalid value.
InvalidVersioningRequest	403	Invalid request for version control	The bucket is in compliance mode.
InvalidLifeCycleRequest	403	Invalid request for life cycle	The bucket is in compliance mode.
RetentionPeriodRequired	400	Invalid request for bucket with compliance	The bucket requires a retention period.
Conflict	409	Conflict	The bucket may be locked.
MethodForbidden	403	Forbidden	Check if quota has been exceeded.
NotAcceptable	406	Content encoding not acceptable	The object `Content-Encoding` does not match requested `Accept-Content`.
InvalidEncoding	400	Invalid URL enconding	The URL encoding used is invalid.
InvalidMetadataQuery	400	Invalid metadata query entered	The metadata query entered does not conform to valid syntax
InvalidMetadataSearchList	400	Invalid metadata search list entered	A keyname on the request is not a valid indexable key, or the format of the request list is incorrect.
MetadataSearchNotEnabled	405	Metadata search not enabled	Metadata search is not enabled for this bucket.
MetadataSearchBadParameter	400	Metadata search invalid parameter used in query	Invalid search index key name, sort key name or attribute name value.
MetadataSearchInvalidArgument	400	Metadata search invalid parameter used in query	Invalid search index value format or operator used.
MetadataSearchInvalidValuefor Datatype	400	Metadata search key indexing found invalid input value	Object operation failed because a user metadata value cannot be converted to its defined datatype.
MetadataOperationNotSupported	405	Metadata search operation not supported	Metadata query with both AND and OR logical operators not supported.
MetadataSearchBadSortParameter	400	Metadata search invalid sort parameter	The sort parameter has to be present in the query as a search parameter.
MetadataSearchRestriction	400	Buckets that are encrypted or within an encrypted namespace cannot have metadata search enabled	Metadata search is mutually exclusive with bucket/namespace encryption.
MetadataSearchTooManyIndexKeys	400	The number of Index keys exceeds the maximum allowed	The number of keys to be indexed exceeds the maximum number allowed, try with fewer keys.
InvalidOrNoCustomerProvided EncryptionKey	400	Invalid or no customer provided encryption key	No encryption key, or an encryption key that did not match the one in the system, was provided.
DareUnavailable	403	Server side encryption (D@RE) is not supported	D@RE JAR/license is unavailable hence server side encryption requests are not supported.

Table 28. Error Codes (continued)

Error Code	HTTP Status Code	Generic Error Code	Description Error
SelfCopyInvalidRequest	400	InvalidRequest	The copy request is illegal because it is trying to copy an object to itself without changing the object's metadata or encryption attributes.
OverLappingPrefixes	400	Invalid Request	Found overlapping prefixes.
SamePrefix	400	Invalid Request	Found two rules with same prefix.
XAmzContentSHA256Mismatch	400	XAmzContentSHA256Mismatch	The Content-SHA256 you specified did not match what we received
InvalidJSON	400	InvalidJSON	Policies must be valid JSON and the first byte must be `{`.
InvalidBucketPolicy	400	InvalidBucketPolicy	Invalid Bucket Policy.
MalformedPolicy	400	MalformedPolicy	Malformed Policy.
MaxIDLengthExceeded	400	InvalidArgument	ID length should not exceed allowed limit of 255.
CrossHeadAccessBeforeUpgrade	400	InvalidRequest	Cross head access is not supported.
InvalidDate	400	InvalidArgument	Date must be no earlier than 1970-01-01T00:00:00.000Z.
BadContentLengthRequest	400	RequestTimeout	Content-Length specified is not matching with Length of the Content in the body.

NOTE:
- The PUT request header is limited to 8 KB in size. Within the PUT request header, the user-defined metadata is limited to 2 KB in size. User-defined metadata is a set of key-value pairs. The size of user-defined metadata is measured by taking the sum of the number of bytes in each key and value plus four: a colon and space to separate the name and value and two bytes for carriage return-linefeed.
- When the system throws a 500 error, it allows the user to retry the request. In such cases, it is recommended to use a backoff algorithm which waits progressively longer between retries for consecutive error responses. For more information about guidance on 500 error rate response in ECS, see KB 504612.

Hadoop S3A for ECS

S3A is an open-source connector for Hadoop. It helps Hadoop users to address the storage scaling issues by providing a second tier of storage that is optimized for cost and capacity.

NOTE: S3A support is available on Hadoop 2.7 or later version.

Hadoop S3A allows you to connect your Hadoop cluster to any S3 compatible object store that is in the public cloud, hybrid cloud, or on-premises.

S3A performance optimization

Performance-related S3A settings are listed in the below table.

Settings	Additional Information
`fs.s3a.multipart.size`	• Default: 100M • Recommended: 150M

Settings	Additional Information
`fs.s3a.fast.upload.active.blocks`	● Default: 4 ● Recommended: 8
`fs.s3a.fast.upload.buffer`	● Default: Disk ● Recommended: Array or bytebuffer **NOTE:** Heap space that is used is `fs.s3a.multipart.size` * `fs.s3a.fast.upload.active.block`
`fs.s3a.threads.max`	● Default: 10 ● Recommended: Change this to 'Between 25% and 50% of configured CPU cores.
`fs.s3a.multiobjectdelete.enable`	● Default: True ● Recommended: True or false
`fs.s3a.committer.threads`	● Default: 8 ● Recommended: 8

Using magic committer

It is recommended to use the magic committer to commit data to disk in various styles and to report all test performance numbers.

NOTE: The magic committer does not support all the Hadoop tools and services. In such cases, the system uses the `FileOutputCommitter` automatically.

When using the magic committer:

- Data is written directly to S3, but retargeted at the final destination.
- Conflict is managed across the directory tree.

Configure the following S3A Hadoop parameters to use the magic committer:

- `fs.s3a.committer.magic.enabled: true`
- `fs.s3a.committer.name: magic`
- `mapreduce.outputcommitter.factory.scheme.s3: org.apache.hadoop.fs.s3a.commit.S3ACommitterFactory`

Hadoop configuration analysis using ECS Service Console

The ECS Service Console (SC) can read and interpret your Hadoop configuration parameters with respect to connections to ECS for S3A and ViPRFS. Also, SC provides a function, `Get_Hadoop_Config` that reads the Hadoop cluster configuration and checks S3A and ViPRFS settings for typos, errors, and for the values that are not recommended. Contact for assistance with installing ECS SC.

```
# service-console run Get_Hadoop_Config
```

Getting temporary credentials

You require temporary credentials to securely access storage through Hadoop S3A.

Prior to ECS IAM, Hadoop access to ECS object storage using S3A required an ECS S3 object user name and a secret key. Also, ACL level security was not possible with S3A. However, with ECS IAM and Secure Token Service (STS) features, an administrator has several, more secure options for controlling access to the S3A storage. One option is to create IAM policies that define permissions which are appropriate for the customer business case. Once the policies are in place, IAM groups can be created and attached to the policies. Individual IAM users can then be created and become members of the IAM groups. IAM users are assigned S3 access keys and secret keys that can be used to access the S3A data, relative to the IAM policy for the IAM user.

Another option for administrators is to use STS and SAML Assertions to allow federated users to obtain temporary credentials. In this use case, a cross trust relationship must be established between the ECS and the Identity Provider. Similar to the previous example, IAM policies must first be created. Once the policies are defined, the administrator creates IAM roles that are attached to the IAM policies. Federated users can then authenticate and obtain a SAML assertion from the Identity Provider. The assertion is used to assume one of the possible IAM roles that are permissible for the user. Once the role has been assumed, the Hadoop user is provided with a temporary access key, a temporary secret key, and a temporary token. The Hadoop user uses these temporary credentials to access the S3A data until the credentials expire. These temporary credentials correspond to the configured policies which enforce security controls on an S3 object store.

For more information about STS, see Secure Token Service.

The temporary credentials are passed to Hadoop using these Hadoop settings:

- fs.s3a.access.key=ACCESS-KEY
- fs.s3a.secret.key=SECRET-KEY
- fs.s3a.session.token=SESSION-TOKEN
- fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider

A sample of temporary credentials provided below:

```
$ hdfs dfs -D fs.s3a.secret.key=SECRET-KEY -D fs.s3a.access.key=ACCESS-KEY -D
fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider
-D fs.s3a.session.token=SESSION-TOKEN -ls s3a://s3aTestBucket/test/SparkWordCount/
```

NOTE: The temporary credentials can last up to 12 hours.

Enabling data2 IP in ECS S3

Data2 IP allows ECS S3 to start on multiple IPs. To enable data2 on S3, contact ECS remote support.

NOTE: Data2 IP is enabled by default in S3 from ECS 3.6.1 and later versions.

2

Cloud DVR

This section describes about the Cloud DVR feature in ECS.

Topics:

Cloud DVR overview

ECS supports cloud Digital Video Recording (DVR) feature which addresses a legal copyright requirement for cable and satellite companies. The requirement is every unit of recording mapped to an object on ECS needs to be copied a predetermined number of times. The predetermined number of copies are known as fanout.

The cloud DVR requirement is to create "N" number of copies each of size "M" at the time of object creation. In this, "N" indicates fanout and "M" indicates unit recording size that can be different for each write request.

NOTE: The cloud DVR feature is available from ECS 3.6 and it is disabled by default. Contact ECS remote support to enable the cloud DVR feature.

Cloud DVR supported APIs

The following table lists the supported Cloud DVR APIs.

Method	Request Header Key	Request Header Value	Request Parameter	Response Parameter	Description
PUT	`x-fanout-copy-count`	1-10,000	-	-	Creates `x-fanout-copy-count` number of copies of object.
GET	`x-fanout-copy-index`	0-9,999	-	-	Reads `x-fanout-copy-index` of object along with 206 (partial-content) response code. Also, it returns an HTTP 404 error when there is no copy or fanout object exist.
HEAD	-	-	-	• `fanout-copy-size` • `fanout-copy-count`	Returns fanout-specific response headers along with other response headers if the object is a fanout object.
PUT-COPY	`x-fanout-copy-index`	0-9,999	-	-	Copies `x-fanout-copy-index` from the fanout source to a destination. Also, it returns an HTTP 404 error if there is no copy or fanout source exist.
DELETE	`x-fanout-delete-all`	-	-	-	Deletes all copies of the fanout object.
DELETE	`x-fanout-copy-index`	0-9,999	-	-	Deletes specific copy of the fanout object. Once this

Method	Request Header Key	Request Header Value	Request Parameter	Response Parameter	Description
					operation is succeeded, any attempt to read the deleted copy must return an HTTP 404 error.
GET	-	-	`fanout`	-	Lists all copies of specific fanout object in bucket. Returns an HTTP 404 error if there is no such fanout object or fanout copies exist.
GET	-	-	`max-keys`	• `IsTruncated` • `NextFanoutKeyMarker` • `NextFanoutIndexMarker`	When these request parameters are used along with a fanout parameter, it limits the maximum number of entries that are returned in a single listing response. When a listing response is truncated (when the bucket contains more entries to be listed), this parameter is set to true. When the parameter is set to true, the `FanoutKeyMarker` and `FanoutIndexMarker` are also present in the response, and should be used in the next listing request to receive the next page of listing results.
GET	-	-	• `fanout-key-marker` • `fanout-index-marker`	-	When these request parameters are used along with a fanout parameter, it is used as a marker to resume paginated listing.
GET	-	-	`prefix`	-	If the `prefix` parameter is present, it limits the listing results to objects, which match the supplied prefix.
GET	-	-	`delimiter`	-	If the `delimiter` parameter is present, it limits the listing results to objects up to the first delimiter encountered.

Cloud DVR API Examples

This section lists a few examples of Cloud DVR APIs in ECS.

Operation name: Fanout create/overwrite

Command:

```
./s3curl.pl --id=personal --ord http://<hostname>:9020/buck1/file1 --put=hello.txt -- -H
"x-fanout-copy-count:3" -v
```

Response:

```
> PUT /buck1/file1 HTTP/1.1
> User-Agent: curl/7.37.0
> Host: 10.247.78.184:9020
> Accept: */*
> Date: Wed, 04 Dec 2019 19:35:38 +0000
> Authorization: AWS user1:xYpE4TDv8Tj/zDZgrRTiQL6fl04=
> x-fanout-copy-count:3
> Content-Length: 11
> Expect: 100-continue
>
< HTTP/1.1 100 Continue
* We are completely uploaded and fine
< HTTP/1.1 200 OK
< Date: Wed, 04 Dec 2019 19:35:38 GMT
* Server ViPR/1.0 is not blacklisted
< Server: ViPR/1.0
< x-amz-request-id: 0af74eb8:16ed21aaacd:e6:1
< x-amz-id-2: 834d1a548e92e0e2487b59a87f45c39b522ac30fffc3cf0d12d6ccaf77c04df7
< ETag: "48e10fc1163cd2c7db2bf9a4225cd5cd"
< Last-Modified: Wed, 04 Dec 2019 19:35:38 GMT
< x-emc-mtime: 1575488138240
< x-emc-previous-object-size: 33
< Content-Length: 0
```

Operation name: Fanout object listing

Command:

```
./s3curl.pl --id=personal --ord http://<hostname>:9020/buck1/file1?fanout | xmllint --
format -
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FanoutListingResult>
 <IsTruncated>false</IsTruncated>
 <FanoutObject>
 <Key>file1</Key>
 <Index>0</Index>
 </FanoutObject>
 <FanoutObject>
 <Key>file1</Key>
 <Index>1</Index>
 </FanoutObject>
 <FanoutObject>
 <Key>file1</Key>
 <Index>2</Index>
 </FanoutObject>
</FanoutListingResult>
```

NOTE: 0-based index represents fanout copies.

Operation name: Fanout copy read

Command:

```
./s3curl.pl --id=personal --ord -- -H "x-fanout-copy-index:0" http://10.247.78.184:9020/
buck1/fo1 -v
```

Response:

```
> GET /buck1/fo1 HTTP/1.1
> User-Agent: curl/7.37.0
> Host: 10.247.78.184:9020
> Accept: */*
> Date: Wed, 29 Jan 2020 03:16:27 +0000
> Authorization: AWS user1:l6u2M9zQspTIwL2wrpGuM4O+tB8=
> x-fanout-copy-index:0
>
< HTTP/1.1 206 Partial Content
< Date: Wed, 29 Jan 2020 03:16:27 GMT
* Server ViPR/1.0 is not blacklisted
< Server: ViPR/1.0
< x-amz-request-id: 0af74eb8:16fef1759e4:80:1
< x-amz-id-2: 54baf9b41b37404e6d350a806473e5d4c7a0bb10612c1dd77157abb800b9eafa
< ETag: "48e10fc1163cd2c7db2bf9a4225cd5cd"
< fanout-copy-count: 5
< fanout-copy-size: 11
< Last-Modified: Wed, 29 Jan 2020 03:14:13 GMT
< x-emc-mtime: 1580267653773
< Content-Type: application/octet-stream
< Content-Length: 11
<
HELLOOO!!!
```

Operation name: Fanout object HEAD

Command:

```
    ./s3curl.pl --id=personal --ord --head http://<hostname>:9020/buck1/file1
```

Response:

```
HTTP/1.1 200 OK
Date: Wed, 04 Dec 2019 19:45:41 GMT
ETag: "48e10fc1163cd2c7db2bf9a4225cd5cd"
fanout-copy-count: 3
fanout-copy-size: 11
Last-Modified: Wed, 04 Dec 2019 19:35:38 GMT
x-emc-mtime: 1575488138240
Server: ViPR/1.0
x-amz-request-id: 0af74eb8:16ed21aaacd:92:d
x-amz-id-2: 834d1a548e92e0e2487b59a87f45c39b522ac30fffc3cf0d12d6ccaf77c04df7
Content-Type: application/octet-stream
Content-Length: 33
```

NOTE: `Fanout-copy-size` is the size of each fanout copy.

Operation name: Fanout put copy

Command:

```
./s3curl.pl --id=personal --ord --copysrc=buck1/file1 -- -H "x-fanout-copy-index:0"
http://<hostname>:9020/buck1/file2 -v
```

Response:

```
> PUT /buck1/file2 HTTP/1.1
> User-Agent: curl/7.37.0
> Host: 10.247.78.184:9020
> Accept: */*
> Date: Wed, 04 Dec 2019 21:11:58 +0000
> Authorization: AWS user1:4gpPVJWrAR1Dbh1LVO/HgK0n/2Y=
> x-amz-copy-source: buck1/file1
> x-fanout-copy-index:0
>
< HTTP/1.1 200 OK
< Date: Wed, 04 Dec 2019 21:11:58 GMT
* Server ViPR/1.0 is not blacklisted
< Server: ViPR/1.0
< x-amz-request-id: 0af74eb8:16ed21aaacd:126:1
< x-amz-id-2: f750866dcee769ae5e0702684e42f8b8c059f074e72e5b0d5ed907f3d6723c0b
< Content-Type: application/xml
< Content-Length: 222
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<CopyObjectResult xmlns="http://s3.amazonaws.com/doc/2006-03-01
```

```
<LastModified>2019-12-04T21:11:59.309Z</LastModified> <ETag>"1575493919143-"</ETag>
</CopyObjectResult>
```

NOTE: Fanout put copy is a deep copy operation.

Operation name: Fanout copy delete

Command:

```
./s3curl.pl --id=personal --ord --delete -- -H "x-fanout-copy-index:0" http://
<hostname>:9020/buck1/file1 -v
```

Response:

```
> DELETE /buck1/file1 HTTP/1.1
> User-Agent: curl/7.37.0
> Host: 10.247.78.184:9020
> Accept: */*
> Date: Wed, 04 Dec 2019 21:19:52 +0000
> Authorization: AWS user1:KTtMUlRmnqa0IIDF7xA2lnWtVFc=
> x-fanout-copy-index:0
>
< HTTP/1.1 204 No Content
< Date: Wed, 04 Dec 2019 21:19:53 GMT
* Server ViPR/1.0 is not blacklisted
< Server: ViPR/1.0
< x-amz-request-id: 0af74eb8:16ed21aaacd:12c:1
< x-amz-id-2: 834d1a548e92e0e2487b59a87f45c39b522ac30fffc3cf0d12d6ccaf77c04df7
< x-emc-previous-object-size: 33
< Content-Length: 0
```

Operation name: Fanout object delete

Command:

```
    ./s3curl.pl --id=personal --ord --delete -- -H "x-fanout-delete-all:true" http://
<hostname>:9020/buck1/file1 -v
```

Response:

```
> DELETE /buck1/file1 HTTP/1.1
> User-Agent: curl/7.37.0
> Host: 10.247.78.184:9020
> Accept: */*
> Date: Wed, 04 Dec 2019 21:22:44 +0000
> Authorization: AWS user1:hpuk6ElD8fHtRFJe/ozpH4ic/BY=
> x-fanout-delete-all:true
>
< HTTP/1.1 204 No Content
< Date: Wed, 04 Dec 2019 21:22:44 GMT
* Server ViPR/1.0 is not blacklisted
< Server: ViPR/1.0
< x-amz-request-id: 0af74eb8:16ed21aaacd:130:1
< x-amz-id-2: 834d1a548e92e0e2487b59a87f45c39b522ac30fffc3cf0d12d6ccaf77c04df7
< x-emc-previous-object-size: 22
< Content-Length: 0
```

Operation name: Fanout bucket listing

Command:

```
    ./s3curl.pl --id=personal --ord http://<hostname>:9020/vdc1_buck1?fanout | xmllint
--format -
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FanoutBucketListingResult>
  <IsTruncated>false</IsTruncated>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>0</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>1</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>2</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>3</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>4</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>0</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>1</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>2</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>3</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>4</Index>
  </FanoutObjects>
</FanoutBucketListingResult>
```

Operation name: Regular bucket listing

Command:

```
    ./s3curl.pl --id=personal --ord http://<hostname>:9020/vdc1_buck1 | xmllint --format
-
```

Response:

```
<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>vdc1_buck1</Name>
  <Prefix/>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  <ServerSideEncryptionEnabled>false</ServerSideEncryptionEnabled>
  <Contents>
    <Key>copyfile1</Key>
    <LastModified>2019-12-14T22:44:22.386Z</LastModified>
    <ETag>"1576363462360-"</ETag>
    <Size>11</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
      <ID>user1</ID>
      <DisplayName>user1</DisplayName>
    </Owner>
    <IsFanoutObject>false</IsFanoutObject>
  </Contents>
  <Contents>
    <Key>file1</Key>
    <LastModified>2019-12-14T22:47:34.319Z</LastModified>
    <ETag>"48e10fc1163cd2c7db2bf9a4225cd5cd"</ETag>
    <Size>55</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
      <ID>user1</ID>
      <DisplayName>user1</DisplayName>
    </Owner>
    <IsFanoutObject>true</IsFanoutObject>
  </Contents>
  <Contents>
    <Key>file2</Key>
    <LastModified>2019-12-14T22:47:44.763Z</LastModified>
    <ETag>"48e10fc1163cd2c7db2bf9a4225cd5cd"</ETag>
    <Size>55</Size>
    <StorageClass>STANDARD</StorageClass>
    <Owner>
      <ID>user1</ID>
      <DisplayName>user1</DisplayName>
    </Owner>
    <IsFanoutObject>true</IsFanoutObject>
  </Contents>
</ListBucketResult>
```

Operation name: Fanout bucket listing with max-keys

Command:

```
./s3curl.pl --id=personal --ord "http://<hostname>:9020/vdc1_buck1?fanout&max-keys=3" |
xmllint --format -
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FanoutBucketListingResult>
  <NextFanoutKeyMarker>file1</NextFanoutKeyMarker>
  <NextFanoutIndexMarker>2</NextFanoutIndexMarker>
  <IsTruncated>true</IsTruncated>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>0</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>1</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>2</Index>
  </FanoutObjects>
</FanoutBucketListingResult>
```

NOTE: The token is the last key in the result set.

Operation name: Fanout bucket listing with token

Command:

```
    ./s3curl.pl --id=personal --ord "http://<hostname>:9020/vdc1_buck1?fanout&max-
keys=3&fanout-key-marker=file1&fanout-index-marker=2" | xmllint --format -
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FanoutBucketListingResult>
  <NextFanoutKeyMarker>file2</NextFanoutKeyMarker>
  <NextFanoutIndexMarker>0</NextFanoutIndexMarker>
  <IsTruncated>true</IsTruncated>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>3</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>4</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>0</Index>
  </FanoutObjects>
</FanoutBucketListingResult>
```

Operation name: Fanout bucket listing with prefix

Command:

```
    ./s3curl.pl --id=personal --ord "http://<hostname>:9020/vdc1_buck1?fanout&max-
keys=3&prefix=file2" | xmllint --format -
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FanoutBucketListingResult>
  <NextFanoutKeyMarker>file2</NextFanoutKeyMarker>
  <NextFanoutIndexMarker>2</NextFanoutIndexMarker>
  <IsTruncated>true</IsTruncated>
  <FanoutPrefix>file2</FanoutPrefix>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>0</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>1</Index>
  </FanoutObjects>
  <FanoutObjects>
    <Key>file2</Key>
    <Index>2</Index>
  </FanoutObjects>
</FanoutBucketListingResult>
```

Operation name: Fanout bucket listing with delimiter

Command:

```
    ./s3curl.pl --id=personal --ord "http://<hostname>:9020/vdc1_buck1?
fanout&delimiter=%2f&max-keys=3" | xmllint --format -
```

Response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<FanoutBucketListingResult>
  <NextFanoutKeyMarker>file1</NextFanoutKeyMarker>
  <NextFanoutIndexMarker>0</NextFanoutIndexMarker>
  <IsTruncated>true</IsTruncated>
  <Delimiter>/</Delimiter>
  <FanoutObjects>
    <Key>file1</Key>
    <Index>0</Index>
  </FanoutObjects>
  <FanoutCommonPrefixes>
    <FanoutPrefix>dir1/</FanoutPrefix>
  </FanoutCommonPrefixes>
  <FanoutCommonPrefixes>
    <FanoutPrefix>dir2/</FanoutPrefix>
  </FanoutCommonPrefixes>
</FanoutBucketListingResult>
```

3

ECS IAM for S3

This section describes about the ECS Identity and Access Management (IAM) feature for S3.

Topics:

ECS IAM overview

ECS Identity and Access Management (IAM) enables you to have fine-grained access to the ECS S3 resources securely. This functionality ensures that each access request to an ECS resource is identified, authenticated, and authorized. ECS IAM allows you to add users, roles, and groups. You can also grant and restrict the access by adding policies to the ECS IAM entities.

NOTE:

- ECS IAM functionality is only supported for the S3 protocol. ECS IAM policies and settings have no impact when data is accessed through other protocols.
- ECS IAM is not enabled for CAS or filesystem enabled buckets.
- ECS IAM entities coexist with legacy object and management users.

ECS IAM identities

ECS IAM provides the ability to manage IAM identities within each namespace such as users, groups, roles, and namespace root user.

ECS IAM Identities	Description
Users	An ECS IAM user represents a person or an application in the namespace that can interact with the ECS resources. ● ECS IAM users belong to one or more IAM groups. ● ECS IAM users have long-term credentials associated with them which are used to access ECS S3, IAM, and STS resources. The credential consists of an access key ID and a secret access key. ● ECS IAM users can access API but not the user interface.
Groups	ECS IAM Groups is a collection of ECS IAM users. Groups let you specify permissions for all the users in the group. Groups cannot contain other groups. **NOTE:** Groups cannot access any ECS APIs only IAM users, and roles can access it.
Roles	An ECS IAM role is an identity that is assumable by trusted internal and external users. A role does not have any credentials associated with it. Instead, when an entity assumes a role, the system provides you the temporary credentials which contain an access key ID, secret access key, and a security token.

ECS IAM Identities	Description
Root user	Namespace root user is an admin user who can also access the user interface. ● Namespace root user is used as owner in ACLs for IAM access. ● Namespace root user console access can be enabled by specifying a password during namespace creation or later.

For more information about configuring these identities, see *ECS Administration Guide*.

Tagging ECS IAM users and roles

Tags are custom key-value pairs that can be associated with users and roles. These tags are used to control the access of an entity to ECS resources.

NOTE:
- Groups and policies cannot be tagged.
- You can apply the same tag to multiple entities, but the tags on an entity cannot have same key.
- Maximum 50 tags per an entity are allowed.

Backward compatibility

This section describes about IAM backward compatibility with ECS legacy entities.

ECS legacy users

Once all the Virtual Data Centers (VDC) are upgraded, it is possible to create IAM users. In detail:
- The existing users created prior to upgrade, also referred as legacy users, continue to exist.
- It is possible to create a class of users called IAM users. IAM users have S3 access, and the IAM capabilities only apply to IAM users.
- There is no behavioral change for legacy users including S3 access.
- Each namespace (account) has a root user similar to AWS root user. They have access similar to an AWS root user access.
- IAM users are similar to AWS IAM users in all respects. For example, they do not use the username as the access key id.
- It is possible to create both legacy and IAM users.

Access control

In this section, ECS IAM users and the namespace root users are referred as IAM users.
- Access control for legacy users remains the same.
- Access control for IAM users is similar to AWS IAM user access control in all respects including ACLs.
- Changing the access control for IAM users such as using identity policies, bucket policies, and ACLs do not have any impact on legacy users access control.
- IAM users can access objects and buckets that are created by legacy users if they are provided with the right permissions. By default,
  - Buckets that are created by legacy users have a default ACL associated with the namespace account that contains the bucket. This ACL provides full control over the bucket. This is true for the buckets that are created before or after the upgrade.
  - Objects that are created by legacy users have default ACL associated with the legacy user only.
  - IAM users who have the required permissions can modify ACLs in buckets and objects that are created by legacy users and set an AWS compatible ACL (account ACL or group ACL).
- Legacy users can access objects and buckets that are created by IAM users. For that, legacy user access control should follow legacy semantics. It must also have legacy ACLs associated with them. In detail:
  - The legacy owner for buckets that are created by IAM users is the namespace root user that contains the bucket.
  - The legacy owner for objects that are created by IAM users are respectively the namespace root of the IAM user.
  - Users with appropriate permissions can set or change ACLs anytime.
- A single bucket policy is supported for both IAM and legacy users.

- Management users can attach both IAM and legacy ACLs to buckets from the user interface and using the API.

ECS IAM API and SDK access

This section describes the supported ECS IAM APIs and its access methods.

ECS IAM supported APIs

The following table lists the supported ECS IAM APIs.

Actions	**Description (*required)**	**Access Level**	**Resource Types (*required)**
AddUserToGroup	Adds an IAM user to the specified IAM group. URL: /iam/?Action=AddUserToGroup Query Parameters: GroupName*, UserName* Error: LimitExceeded, NoSuchEntity, ServiceFailure	Write	group*
AttachGroupPolicy	Attach a specified managed policy to the specified IAM group. URL: /iam/?Action=AttachGroupPolicy Query Parameters: GroupName*, PolicyArn* Error: LimitExceeded, NoSuchEntity, ServiceFailure, InvalidInput	Permissions management	group*
AttachRolePolicy	Attach a specified managed policy to the specified IAM role. URL: /iam/?Action=AttachRolePolicy Query Parameters: RoleName*, PolicyArn* Error: LimitExceeded, NoSuchEntity, ServiceFailure, InvalidInput	Permissions management	role*
AttachUserPolicy	Attach a specified managed policy to the specified IAM user. URL: /iam/?Action=AttachUserPolicy Query Parameters: UserName*, PolicyArn* Error: LimitExceeded, NoSuchEntity, ServiceFailure, InvalidInput	Permissions management	user*
CreateAccessKey	Creates a new Secret Access credential for specified IAM user. URL: /iam/?Action=CreateAccessKey Query Parameters: UserName Error: LimitExceeded, NoSuchEntity, ServiceFailure	Write	user*
CreateGroup	Creates a IAM group in namespace. URL: /iam/?Action=CreateGroup	Write	group*

Actions	Description (*required)	Access Level	Resource Types (*required)
	Query Parameters: GroupName*, Path (only '/' supported) Error: EntityAlreadyExists, LimitExceeded, NoSuchEntity, ServiceFailure		
CreatePolicy	Creates a new managed policy in namespace. URL: /iam/?Action=CreatePolicy Query Parameters: Description, Path (only '/' allowed), PolicyDocument*, PolicyName* Error: EntityAlreadyExists, InvalidInput, LimitExceeded, MalformedPolicyDocument, ServiceFailure	Permissions management	policy*
CreatePolicyVersion	Creates a version of the specified managed policy in namespace. URL: /iam/?Action=CreatePolicy Query Parameters: PolicyArn*, PolicyDocument*, SetAsDefault* Error: InvalidInput, LimitExceeded, MalformedPolicyDocument, NoSuchEntity, ServiceFailure	Permissions management	policy*
CreateRole	Creates a IAM role in namespace. URL: /iam/?Action=CreateRole Query Parameters: AssumeRolePolicyDocument*, Description, MaxSessionDuration, PermissionsBoundary, Tags, RoleName*, Path (only '/' supported) Error: EntityAlreadyExists, InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure, MalformedPolicyDocument	Write	role*
CreateSAMLProvider	Creates a SAML 2.0 identity provider (IdP) in namespace URL: /iam/?Action=CreateSAMLProvider Query Parameters: Name*, SAMLMetadataDocument* Error: EntityAlreadyExists, InvalidInput, LimitExceeded, ServiceFailure	Write	saml-provider*
CreateUser	Creates a IAM user in namespace. URL: /iam/?Action=CreateUser Query Parameters: Path (only '/' supported), PermissionsBoundary, Tags, UserName* Error: EntityAlreadyExists, InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Write	user*

Actions	Description (*required)	Access Level	Resource Types (*required)
DeleteAccessKey	Deletes the specified access key credential that is associated with the specified IAM user. URL: /iam/?Action=DeleteAccessKey Query Parameters: AccessKeyId*, UserName Error: LimitExceeded, NoSuchEntity, ServiceFailure	Write	user*
DeleteGroup	Deletes the specified IAM group from namespace. URL: /iam/?Action=DeleteGroup Query Parameters: GroupName* Error: DeleteConflict, LimitExceeded, NoSuchEntity, ServiceFailure	Write	group*
DeleteGroupPolicy	Deletes the specified inline policy from its group. URL: /iam/?Action=DeleteGroupPolicy Query Parameters: GroupName*, PolicyName* Error: LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	group*
DeletePolicy	Deletes the specified managed policy URL: /iam/?Action=DeletePolicy Query Parameters: PolicyArn* Error: DeleteConflict, InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	policy*
DeletePolicyVersion	Deletes the specified version from the managed policy. URL: /iam/?Action=DeletePolicyVersion Query Parameters: PolicyArn*, VersionId* Error: DeleteConflict, InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	policy*
DeleteRole	Grants permission to delete the specified role. URL: /iam/?Action=DeleteRole Query Parameters: RoleName* Error: DeleteConflict, LimitExceeded, NoSuchEntity, ServiceFailure	Write	role*
DeleteRolePermissionsBoundary	Deletes the permissions boundary for the specified IAM role. URL: /iam/?Action=DeleteRolePermissionsBoundary Query Parameters: RoleName*	Permissions management	role*

Actions	Description (*required)	Access Level	Resource Types (*required)
	Error: NoSuchEntity, ServiceFailure		
DeleteRolePolicy	Deletes the specified inline policy from its role. URL: /iam/?Action=DeleteRolePolicy Query Parameters: RoleName*, PolicyName* Error: LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	role*
DeleteSAMLProvider	Deletes a specified SAML provider. URL: /iam/?Action=DeleteSAMLProvider Query Parameters: SAMLProviderArn* Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Write	saml-provider*
DeleteUser	Deletes the specified IAM user from namespace. URL: /iam/?Action=DeleteUser Query Parameters: UserName* Error: DeleteConflict, LimitExceeded, NoSuchEntity, ServiceFailure	Write	user*
DeleteUserPermissionsBoundary	Deletes the permissions boundary for the specified IAM user. URL: /iam/?Action=DeleteUserPermissionsBoundary Query Parameters: UserName* Error: NoSuchEntity, ServiceFailure	Permissions management	user*
DeleteUserPolicy	Deletes the specified inline policy from its user. URL: /iam/?Action=DeleteUserPolicy Query Parameters: UserName*, PolicyName* Error: LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	user*
DetachGroupPolicy	Detach a specified managed policy from the specified IAM group. URL: /iam/?Action=DetachGroupPolicy Query Parameters: GroupName*, PolicyArn* Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	group*
DetachRolePolicy	Detach a specified managed policy from the specified IAM role. URL: /iam/?Action=DetachRolePolicy Query Parameters: RoleName*, PolicyArn*	Permissions management	role*

Actions	Description (*required)	Access Level	Resource Types (*required)
	Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure		
DetachUserPolicy	Detach a specified managed policy from the specified IAM user. URL: /iam/?Action=DetachUserPolicy Query Parameters: UserName*, PolicyArn* Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	user*
GetAccessKeyLastUsed	Retrieves best effort information about when specified access key was last used. URL: /iam/?Action=GetAccessKeyLastUsed Query Parameters: AccessKeyId* Error: ServiceFailure	Read	user*
GetContextKeysForCustomPolicy	Retrieves list of all of the context keys referenced in the input policies. URL: /iam/?Action=GetContextKeysForCustomPolicy Query Parameters: PolicyInputList* Error: InvalidInput	Read	-
GetContextKeysForPrincipalPolicy	Detach a specified managed policy from the specified IAM user. URL: /iam/?Action=GetContextKeysForPrincipalPolicy Query Parameters: PolicyInputList, PolicySourceArn* Error : InvalidInput, NoSuchEntity	Read	user, group, role
GetGroup	Returns a list of IAM users that are in the specified IAM group. You can paginate the results using the MaxItems and Marker parameters. URL: /iam/?Action=GetGroup Query Parameters: GroupName*, Marker, MaxItems Error: NoSuchEntity, ServiceFailure	Read	group*
GetGroupPolicy	Gets the specified inline policy document from the specified IAM group. URL: /iam/?Action=GetGroupPolicy Query Parameters: GroupName*, PolicyName* Error: NoSuchEntity, ServiceFailure	Read	group*
GetPolicy	Retrieve information about the specified managed policy.	Read	policy*

Actions	Description (*required)	Access Level	Resource Types (*required)
	URL: /iam/?Action=GetPolicy Query Parameters: PolicyArn* Error: InvalidInput, NoSuchEntity, ServiceFailure		
GetPolicyVersion	Retrieve information about a version of the specified managed policy. URL: /iam/?Action=GetPolicyVersion Query Parameters: PolicyArn*, VersionId* Error: InvalidInput, NoSuchEntity, ServiceFailure	Read	policy*
GetRole	Retrieves information about the specified role. URL: /iam/?Action=GetRole Query Parameters: RoleName* Error: NoSuchEntity, ServiceFailure	Read	role*
GetPolicy	Retrieves information about specified managed policy. URL: /iam/?Action=GetPolicy Query Parameters: PolicyArn* Error: InvalidInput, NoSuchEntity, ServiceFailure	Read	policy*
GetPolicyVersion	Retrieves information about specified version of the managed policy. URL: /iam/?Action=GetPolicyVersion Query Parameters: PolicyArn*, VersionId* Error: InvalidInput, NoSuchEntity, ServiceFailure	Read	policy*
GetRolePolicy	Retrieves the specified inline policy document that is embedded with the specified IAM role. URL: /iam/?Action=GetRolePolicy Query Parameters: RoleName*, PolicyName* Error: NoSuchEntity, ServiceFailure	Read	role*
GetSAMLProvider	Retrieves the SAML provider metadata document that is associated with the IAM SAML provider resource. URL: /iam/?Action=GetSAMLProvider Query Parameters: SAMLProviderArn* Error: InvalidInput, NoSuchEntity, ServiceFailure	Read	saml-provider*
GetUser	Retrieves information about the specified IAM user URL: /iam/?Action=GetUser	Read	user*

Actions	Description (*required)	Access Level	Resource Types (*required)
	Query Parameters: UserName Error: NoSuchEntity, ServiceFailure		
GetUserPolicy	Retrieves the specified inline policy document that of the specified IAM user. URL: /iam/?Action=GetUserPolicy Query Parameters: UserName*, PolicyName* Error: NoSuchEntity, ServiceFailure	Read	user*
ListAccessKeys	Lists information about the access key IDs that are associated with the specified IAM user. URL: /iam/?Action=ListAccessKeys Query Parameters: UserName* Error: NoSuchEntity, ServiceFailure	List	user*
ListAttachedGroupPolicies	List all managed policies that are attached to the specified IAM group. URL: /iam/?Action=ListAttachedGroupPolicies Query Parameters: GroupName*, Marker, MaxItems, PathPrefix (only '/' supported) Error: InvalidInput, NoSuchEntity, ServiceFailure	List	group*
ListAttachedRolePolicies	List all managed policies that are attached to the specified IAM role. URL: /iam/?Action=ListAttachedRolePolicies Query Parameters: RoleName*, Marker, MaxItems, PathPrefix (only '/' supported) Error: InvalidInput, NoSuchEntity, ServiceFailure	List	role*
ListAttachedUserPolicies	List all managed policies that are attached to the specified IAM user URL: /iam/?Action=ListAttachedUserPolicies Query Parameters: UserName*, Marker, MaxItems, PathPrefix (only '/' supported) Error: InvalidInput, NoSuchEntity, ServiceFailure	List	user*
ListEntitiesForPolicy	Lists all entities (IAM users, groups, and roles) that are attached to the specified managed policy. URL: /iam/?Action=ListEntitiesForPolicy Query Parameters: EntityFilter, Marker, MaxItems, PathPrefix (only '/' supported), PolicyArn*, PolicyUsageFilter Error: InvalidInput, NoSuchEntity, ServiceFailure	List	policy*

Actions	Description (*required)	Access Level	Resource Types (*required)
ListGroupPolicies	List the names of the inline policies that are in the specified IAM group. URL: /iam/?Action=ListGroupPolicies Query Parameters: GroupName*, Marker, MaxItems Error: NoSuchEntity, ServiceFailure	List	group*
ListGroups	List the IAM groups that have the specified path prefix. URL: /iam/?Action=ListGroups Query Parameters: Marker, MaxItems, PathPrefix (only '/' supported) Error: ServiceFailure	List	-
ListGroupsForUser	List the IAM groups that the provided IAM user belongs to. URL: /iam/?Action=ListGroupsForUser Query Parameters: Marker, MaxItems, UserName* Error: NoSuchEntity, ServiceFailure	List	user*
ListMulitpartUploads			
ListPolicies	Lists all managed policies that are associated with the namespace. URL: /iam/?Action=ListPolicies Query Parameters: Marker, MaxItems, OnlyAttached, PathPrefix (only '/' supported), PolicyUsageFilter, Scope Error: InvalidInput, NoSuchEntity, ServiceFailure	List	-
ListPolicyVersions	Lists information about the versions of the requested managed policy. URL: /iam/?Action=ListPolicyVersions Query Parameters: Marker, MaxItems, PolicyArn* Error: InvalidInput, NoSuchEntity, ServiceFailure	List	policy*
ListRolePolicies	List the names of the inline policies that are in the specified IAM role. URL: /iam/?Action=ListRolePolicies Query Parameters: RoleName*, Marker, MaxItems Error: NoSuchEntity, ServiceFailure	List	role*
ListRoles	List the IAM roles that have the specified path prefix. URL: /iam/?Action=ListRoles Query Parameters: Marker, MaxItems, PathPrefix (only '/' supported)	List	-

Actions	Description (*required)	Access Level	Resource Types (*required)
	Error: ServiceFailure		
ListRoleTags	Lists the tags that are attached to the specified role. URL: /iam/?Action=ListRoleTags Query Parameters: Marker, MaxItems, RoleName* Error: NoSuchEntity, ServiceFailure	List	role*
ListSAMLProviders	List the SAML providers in the namespace. URL: /iam/?Action=ListSAMLProviders Error: ServiceFailure	List	-
ListUserPolicies	List the names of the inline policies that are in the specified IAM user. URL: /iam/?Action=ListUserPolicies Query Parameters: UserName*, Marker, MaxItems Error: NoSuchEntity, ServiceFailure	List	user*
ListUsers	List the IAM users that have the specified path prefix. URL: /iam/?Action=ListUsers Query Parameters: Marker, MaxItems, PathPrefix (only '/' supported) Error: ServiceFailure	List	-
ListUserTags	Lists the tags that are attached to the specified user. URL: /iam/?Action=ListUserTags Query Parameters: Marker, MaxItems, UserName* Error: NoSuchEntity, ServiceFailure	List	user*
PutGroupPolicy	Adds or updates an inline policy document to the specified IAM group. URL: /iam/?Action=PutGroupPolicy Query Parameters: GroupName*, PolicyDocument*, PolicyName* Error: LimitExceeded, MalformedPolicyDocument, NoSuchEntity, ServiceFailure	Permissions management	group*
PutRolePermissionsBoundary	Sets or updates the provided managed policy as the roles permissions boundary. URL: /iam/?Action=PutRolePermissionsBoundary Query Parameters: RoleName*, PermissionsBoundary*	Permissions management	role*

Actions	Description (*required)	Access Level	Resource Types (*required)
	Error: InvalidInput, PolicyNotAttachable, NoSuchEntity, ServiceFailure		
PutRolePolicy	Adds or updates an inline policy document to the specified IAM role. URL: /iam/?Action=PutRolePolicy Query Parameters: RoleName*, PolicyDocument*, PolicyName* Error: LimitExceeded, MalformedPolicyDocument, NoSuchEntity, ServiceFailure	Permissions management	role*
PutUserPermissionsBoundary	Sets or updates the provided managed policy as the users permissions boundary. URL: /iam/?Action=PutUserPermissionsBoundary Query Parameters: UserName*, PermissionsBoundary* Error: InvalidInput, PolicyNotAttachable, NoSuchEntity, ServiceFailure	Permissions management	user*
PutUserPolicy	Adds or updates an inline policy document to the specified IAM user. URL: /iam/?Action=PutUserPolicy Query Parameters: UserName*, PolicyDocument*, PolicyName* Error: LimitExceeded, MalformedPolicyDocument, NoSuchEntity, ServiceFailure	Permissions management	user*
RemoveUserFromGroup	Remove an IAM user from the specified group. URL: /iam/?Action=RemoveUserFromGroup Query Parameters: UserName*, GroupName* Error: LimitExceeded, NoSuchEntity, ServiceFailure	Write	group*
SetDefaultPolicyVersion	Sets the specified version of the policy as default URL: /iam/?Action= SetDefaultPolicyVersion Query Parameters: PolicyArn*, VersionId* Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Permissions management	policy*
SimulateCustomPolicy	Simulate how a set of IAM policies and optionally a resource-based policy works with a list of API operations and RCS resources to determine the policies effective permissions.	Read	-

Actions	Description (*required)	Access Level	Resource Types (*required)
	URL: /iam/?Action= SimulateCustomPolicy Query Parameters: ActionNames*, CallerArn, ContextEntries, Marker, MaxItems, PolicyInputList*, ResourceArns, ResourceOwner, ResourcePolicy Error: InvalidInput, PolicyEvaluation		
SimulatePrincipalPolicy	Simulate how a set of IAM policies attached to an IAM entity (user, group, or role) works with a list of API operations and ECS resources to determine the policies effective permissions. URL: /iam/?Action= SimulatePrincipalPolicy Query Parameters: ActionNames*, CallerArn, ContextEntries, Marker, MaxItems, PolicyInputList, PolicySourceArn*, ResourceArns, ResourceOwner, ResourcePolicy Error : InvalidInput, NoSuchEntity, PolicyEvaluation	Read	user, group, role
TagRole	Add tags to an IAM role. URL: /iam/?Action=TagRole Query Parameters: RoleName*, Tags Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Tagging	role*
TagUser	Add tags to an IAM user . URL: /iam/?Action=TagUser Query Parameters: UserName*, Tags Error: InvalidInput, LimitExceeded, NoSuchEntity, ServiceFailure	Tagging	user*
UntagRole	Remove tags from specified IAM role. URL: /iam/?Action=UntagRole Query Parameters: RoleName*, Tags Error: NoSuchEntity, ServiceFailure	Tagging	role*
UntagUser	Remove tags from specified IAM user. URL: /iam/?Action=UntagUser Query Parameters: UserName*, Tags Error: NoSuchEntity, ServiceFailure	Tagging	user*
UpdateAccessKey	Update the status of the specified access key as Active or Inactive. URL: /iam/?Action=UpdateAccessKey Query Parameters: AccessKeyId*, Status*, UserName	Write	user*

Actions	Description (*required)	Access Level	Resource Types (*required)
	Error: LimitExceeded, NoSuchEntity, ServiceFailure		
UpdateAssumeRolePolicy	Update the policy that grants an IAM entity permission to assume a role. URL: /iam/?Action=UpdateAssumeRolePolicy Query Parameters: PolicyDocument*, RoleName* Error: LimitExceeded, MalformedPolicyDocument, NoSuchEntity, ServiceFailure	Permissions management	role*
UpdateRole	Updates the description or maximum session duration setting of an IAM role. URL: /iam/?Action=UpdateRole Query Parameters: Description, MaxSessionDuration, RoleName* Error: NoSuchEntity, ServiceFailure	Write	role*
UpdateSAMLProvider	Updates the metadata document for an existing SAML provider. URL: /iam/?Action=UpdateSAMLProvider Query Parameters: SAMLMetadataDocument*, SAMLProviderArn* Error: Invalidinput, LimitExceeded, NoSuchEntity, ServiceFailure	Write	saml-provider*

ECS S3 and IAM API access

API	Access method
S3	• Legacy users with appropriate access key credentials and relevant bucket policy/ACLs can access S3 API. • IAM users with appropriate valid access key credentials and appropriate permissions can access S3 API. • IAM roles with appropriate temporary credentials and appropriate permissions can access S3 API. **NOTE:** ECS management users must create legacy users or IAM users or IAM roles with the required permissions to access S3 API.
IAM	• Legacy users cannot access IAM API. • IAM users with valid credentials and appropriate permissions can access IAM API. • IAM roles with valid temporary credentials and appropriate permissions can access IAM API. • ECS management users can obtain X-SDS-AUTH-TOKEN from auth service to access IAM API. **NOTE:** ECS management users can also create IAM users or IAM roles with the required permissions to access IAM API.
Other management APIs	• Legacy users, IAM users, and IAM roles cannot access other management APIs.

API	Access method
	• ECS management users can obtain X-SDS-AUTH-TOKEN from auth service to access other management APIs.
Other data head APIs (except S3)	• Legacy users with valid access key credentials and appropriate permissions can access other data head APIs. • IAM users or roles cannot access other data head APIs. **NOTE:** ECS management users must create legacy users with the required permissions to access other data head APIs.

AWS SDK APIs not supported in ECS IAM

The below table lists the AWS SDK APIs that are not supported in ECS IAM.

NOTE: All these APIs are certified against AWS Java SDK version 1.11.769.

Feature	API
User/Account	UpdateUser
User/Account	UploadSSHPublicKey
User/Account	UpdateSSHPublicKey
User/Account	DeleteSSHPublicKey
User/Account	ListSSHPublicKeys
User/Account	GetSSHPublicKey
User/Account	ChangePassword
User/Account	CreateAccountAlias
User/Account	DeleteAccountAlias
User/Account	ListAccountAliases
User/Account	CreateLoginProfile
User/Account	GetLoginProfile
User/Account	UpdateLoginProfile
User/Account	DeleteLoginProfile
OIDC Support	CreateOpenIDConnectProvider
OIDC Support	AddClientIDToOpenIDConnectProvider
OIDC Support	GetOpenIDConnectProvider
OIDC Support	ListOpenIDConnectProviders
OIDC Support	DeleteOpenIDConnectProvider
OIDC Support	RemoveClientIDFromOpenIDConnectProvider
OIDC Support	UpdateOpenIDConnectProviderThumbprint
Policy	DeleteAccountPasswordPolicy
Policy	GetAccountPasswordPolicy
Policy	UpdateAccountPasswordPolicy
Policy	ListPoliciesGrantingServiceAccess
Group	UpdateGroup

Feature	API
Role	UpdateRoleDescription
Role	CreateServiceLinkedRole
Role	GetServiceLinkedRoleDeletionStatus
Role	DeleteServiceLinkedRole
STS	SetSecurityTokenServicePreferences
MFA	CreateVirtualMFADevice
MFA	DeactivateMFADevice
MFA	DeleteVirtualMFADevice
MFA	EnableMFADevice
MFA	ListMFADevices
MFA	ListVirtualMFADevices
MFA	ResyncMFADevice
Report/Audit	GenerateCredentialReport
Report/Audit	GenerateOrganizationsAccessReport
Report/Audit	GenerateServiceLastAccessedDetails
Report/Audit	GetAccountAuthorizationDetails
Report/Audit	GetAccountSummary
Report/Audit	GetCredentialReport
Report/Audit	GetOrganizationsAccessReport
Report/Audit	GetServiceLastAccessedDetails
Report/Audit	GetServiceLastAccessedDetailsWithEntities
Report/Audit	GetServiceLinkedRoleDeletionStatus

ECS IAM error codes

The below table lists the ECS IAM error codes.

Error type	HTTP status code	Description
AccessDeniedException	400	Indicates that you do not have the required access to perform the action.
ConcurrentModification	409	Indicates that multiple requests are submitted simultaneously to modify the object. You need to wait for a few minutes and submit the request again.
DeleteConflict	409	Indicates that the request is raised to delete a resource that is attached with another entity.
DeleteBucket	400	Indicates that background delete is active, that the bucket is not empty, or that empty bucke is in progress.
EntityAlreadyExists	409	Indicates that the request is raised to create a resource that already exists.
ExpiredToken	400	Indicates that the Web identity token that is used to perform the action is expired or not valid.

Error type	HTTP status code	Description
IDPRejectedClaim	403	Indicates that the identity provider (IdP) reported that authentication failed.
InternalFailure	500	Indicates that the request failed due to an unknown error, exception, or failure.
InvalidAction	400	Indicates that the requested action is not valid.
InvalidInput	400	Indicates that an invalid or an out-of-range value is provided for an input.
InvalidParameterValue	400	Indicates that an invalid or an out-of-range value is provided for an input parameter.
LimitExceeded	409	Indicates that the request is rejected because an attempt is made to create resources beyond the current account limits.
MalformedPolicyDocument	400	Indicates that the provided policy document is malformed.
MissingAction	400	Indicates that the action or a required parameter is missed in the request.
MissingParameter	400	Indicates that the required parameter is missed in the request.
NoSuchEntity	404	Indicates that the referenced entity does not exist.
NotImplemented	501	Indicates that the mentioned functionality is not implemented yet.
PackedPolicyTooLarge	400	Indicates that the total packed size of the session policies and session tags combined is too large. See ECS IAM limitations on entities and objects.
PermissionDenied	403	Indicates that the principal does not have the required permission to perform the action.
ServiceFailure	500	Indicates that the request is failed because of an unknown error, exception, or failure.
ServiceUnavailable	503	Indicates that the request is failed due to a temporary failure of the server.
ValidationError	400	Indicates that the input fails to satisfy the constraints specified by the specific API.
Various Bucket Errors	400	Indicates that empty bucket is in progress, or that writes are not allowed when empty bucket is in progress.

ECS IAM supported condition keys

ECS IAM supports the following condition keys:

Global condition keys	Type	Description
`aws:CurrentTime`	Date	To check for date and time conditions
`aws:EpochTime`	Date	To check for date and time conditions using a date in epoch or UNIX time
`aws:PrincipalArn`	ARN	Checks the ARN of the IAM user or role that made the request.
`aws:UserAgent`	String	To check the client application of the requestor.
`aws:PrincipalTag/` *`tag-key`*	String	Checks that the tag attached to the principal making the request matches the specified key name and value.

Global condition keys	Type	Description
`aws:RequestTag/ tag-key`	String	Checks that the tag key-value pair is present in an AWS request.
`aws:ResourceTag/ tag-key`	String	Checks that the tag key-value pair is attached to the resource.
`aws:SourceIp`	IpAddr	To check the IP address of the requester
`aws:TagKeys`	String, ForAllValues:String ForAnyValue: String	This context key is a list of tag keys without values
`aws:TokenIssueTime`	Date	Checks the date and time that temporary security credentials were issued.
`aws:principaltype`	String	Indicates the type of principal making the request. ● Root user is Account. ● IAM user is User. ● Legacy object user is ECSUser. ● SAML or Assumed role user is AssumedRole.
`aws:userid`	String	Based on authorized user access is set to the following: ● Root user ARN if root user is requester. ● IAM user unique id IAM user is requester. ● If SAML federated user is requester, it is set to the role-id:caller-specified-role-name ● If assumed role user is requester, it is set to the role-id:caller-specified-role-name `role-id:` is the unique id of role `caller-specified-role-name:` is the `RoleSessionName` in `AssumeRole` request or the name attribute value in SAML assertion passed to `AssumeRoleWithSAML` request.
`aws:username`	String	Based on authorized user access, if requester is an IAM user, it is set to the IAM username otherwise it is not set.

IAM condition keys	Type	Description
`iam:PermissionsBoundary`	String	Checks that the specified policy is attached as permissions boundary on the IAM principal resource.
`iam:PolicyARN`	ARN	Checks the ARN of a managed policy in requests that involve a managed policy.
`iam:ResourceTag/` *`key-name`*	String	Checks that the tag attached to the IAM entity (user or role) matches the specified key name and value.

STS and SAML condition keys	Type	Description
`saml:aud`	String	An endpoint URL to which SAML assertions are presented. The value for this key comes from the `SAML Recipient` field in the assertion, not the `Audience` field.
`saml:edupersonorgdn`	String	This is an `eduPerson` attribute in SAML assertion.
`saml:iss`	String	The issuer, which is represented by a URN.
`saml:namequalifier`	String	This contains a hash value that represents the combination of the `saml:doc` and `saml:iss` values. It is used as a namespace qualifier; the combination

STS and SAML condition keys	Type	Description
		of `saml:namequalifier` and `saml:sub` uniquely identifies a user.
`saml:sub`	String	This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization.
`saml:sub_type`	String	This key can have the value `persistent` , `transient` , or consist of the full `Format` URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of `persistent` indicates that the value in `saml:sub` is the same for a user between sessions. If the value is transient , the user has a different `saml:sub` value for each session.

S3 condition keys	Description
`s3:x-amz-acl`	Specifies the canned ACL in the request.
`s3:x-amz-grant-` *`permission`*	Specifies *permission* for the following access. ● read ● write ● read-acp ● write-acp ● full-control
`s3:x-amz-copy-source`	Enables restricting copy source to a specific bucket, folder, or object.
`s3:x-amz-metadata-directive`	Specifies certain behavior to be enforced during object uploads (COPY vs REPLACE).
`s3:x-amz-server-side-encryption`	Specifies that the request should contain this header to ensure that the uploads are stored encrypted.
`s3:VersionId`	Limits access to specific versions of object.
`s3:LocationConstraint`	Using this condition key, you can restrict a user to create a bucket in a specific AWS Region.
`s3:delimiter`	Used to require the requester to specify delimiter parameter.
`s3:max-keys`	Limits ListBucket requests to the set s3:max-keys value.
`s3:prefix`	Limits ListBucket and ListBucketVersions to specific prefix.
`s3:ExistingObjectTag/` *`<tag-key>`*	Using this condition key, you can limit the permission for the `s3:PutObjectAcl` action to only on objects that have a specific tag key and value.
`s3:RequestObjectTagKeys`	Using this condition key, you can limit permission for the `s3:PutObject` action by restricting the object tags allowed in the request.
`s3:RequestObjectTag/` *`<tag-key>`*	Using this condition key, you can limit permission for the `s3:PutObject` action by restricting the object tags allowed in the request.

ECS IAM limitations on entities and objects

ECS IAM has certain limitations on its resources such as naming the entities, characters to be used for the identities, number of policies to be attached to an entity, and the number of resources that can be linked to an entity.

NOTE: Paths are not supported for IAM entities.

ECS IAM entity name limits

Resource	Limits
Names of users, groups, roles, and managed policies	• Must be unique within the namespace. • Must be alphanumeric and it may include any of these special characters: Plus (+), equal (=), comma (,), period (.), at (@), underscore (_), and hyphen (-). **NOTE:** These names are case insensitive.
Inline policy names	• Must be unique to the user, group, or to the role that they are embedded in. • Can contain any Basic Latin (ASCII) characters except these special characters: Backward slash (\), forward slash (/), asterisk (*), question mark (?), and space. These characters are reserved according to the RFC (Request for Comments) 3986 Internet standard.
Policy documents	Can contain these Unicode characters: horizontal tab (U+0009), linefeed (U+000A), carriage return (U+000D), and characters in the range from U+0020 to U+00FF.

ECS IAM entity object limits

Resource	Limit
Users in a namespace	500
Groups in a namespace	100
Roles in a namespace	200
Customer-managed policies in a namespace	500
ECS IAM users in a group	Equal to user quota in namespace
Managed policies that are attached to an ECS IAM group	10
Managed policies that are attached to an ECS IAM role	10
Managed policies that are attached to an ECS IAM user	10

ECS IAM entities limits

Resource	Limit
Access keys that are assigned to an ECS IAM user	2
Access keys that are assigned to the namespace root user	2
Groups an ECS IAM user can be a member of	10
Identity providers (IdPs) associated with an ECS IAM SAML provider object	1
Keys per SAML provider	1
Permissions boundaries for an ECS IAM user	1
Permissions boundaries for an ECS IAM role	1
SAML providers in an AWS account	10
Tags that can be attached to an ECS IAM user	50
Tags that can be attached to an ECS IAM role	50
Versions of a managed policy that can be stored	5

ECS IAM entity character limits

Description	Limit
Path	Only the character slash (/) is supported.
User name	64 characters
Group name	128 characters
Role name	64 characters
Tag key	128 characters
Tag value	256 characters **NOTE:** Tag values can be empty. That is, tag values can have a length of 0 characters.
Unique IDs created by ECS IAM	128 characters
Policy name	128 characters
Role trust policy JSON text (the policy that determines who is allowed to assume the role)	2,048 characters
Role session name	64 characters
Max role session duration	12 hours
For inline policies	You can add as many inline policies as you want to an IAM user, role, or group. But the total aggregate policy size (the sum size of all inline policies) per entity cannot exceed the following limits: ● User policy size cannot exceed 2,048 characters. ● Role policy size cannot exceed 10,240 characters. ● Group policy size cannot exceed 5,120 characters. **NOTE:** IAM does not count white space when calculating the size of a policy against these limitations.
For managed policies	● You can add up to 10 managed policies to an IAM user, role, or group. ● The size of each managed policy cannot exceed 6,144 characters. **NOTE:** IAM does not count white space when calculating the size of a policy against these limitations.
For session policies	● You can pass only one inline policy or specify up to 10 managed policy ARNs when assuming a role. ● The size of each session policy cannot exceed 2,048 characters.

ECS IAM access management

ECS IAM access is managed by creating policies and ACLs, and associating them with ECS resources and identities.

ECS IAM Policies

Policies specify what permissions are granted to an ECS entity which needs to access a resource.

For example, policies can:

- Specify actions on a resource.
- Identify resources.

- Identify principals that are applicable for the policies.
- Specify conditions that are applicable.

ECS IAM supports the following policy types:

Policies	Description
Identity-based policies	Policies that are assigned to users, groups, and roles which grant permissions to an identity. ● Inline Policies ● Managed Policies (Both ECS and Customer managed)
Resource-based policies	These are inline policies that are assigned to an ECS resource that grants specified principal permission to perform specific action on the resource. ● Bucket Policy ● Trust Policy - Is a resource-based policy that is attached to an IAM role. Trust policies identify the principal entities that can assume the role.
Permission Boundaries	Use a managed policy as the permissions boundary for an IAM entity (user or role). That policy defines the maximum permissions that the identity-based policies can grant to an entity, but does not grant permissions. Permissions boundaries do not define the maximum permissions that a resource-based policy can grant to an entity.
Session policies	Session policies are used with AssumeRole and AssumeRoleWithSAML APIs. Session policies limit the permissions that the identity-based policies of a role or user grant to the session. Session policies limit permissions for a created session, but do not grant permissions.
Access Control Lists (ACLs)	ACLs are cross-account permissions policies that grant permissions to the specified principal.

NOTE: If there is an explicit deny in any policy, then the request is denied otherwise there must be a policy that explicitly allows the request. If neither then by default the request is denied.

ACLs

This section describes the differences between the ECS S3 user ACL access with the ECS IAM S3 user ACL access.

S3 non-ECS IAM access	S3 ECS IAM access
Users own buckets and objects.	Buckets are owned by the namespace to which they belong and objects are owned by the namespace to which the user that created the object belongs.
Bucket and object owners can be changed.	Buckets and object owners can never be changed.
Any user can be a non-group grantee in an ACL.	Only a namespace can be a non-group grantee in an ACL.

S3 Request authorization

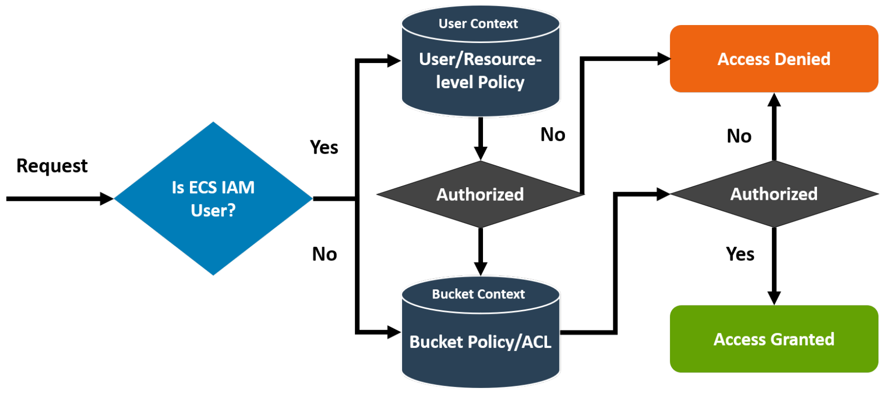
During the S3 request authorization process, the system evaluates permission using user, bucket, and object contexts as needed.

Context	Description
User	In this context, if the requester is an ECS IAM principal, the principal must have permission from the parent namespace to which it belongs. In this step, the subset of policies that are owned by the parent account (also referred as the context authority) is evaluated. This subset of policies includes the user policy that the parent attaches to the principal. If the parent also owns the resource in the request (bucket, object), then the corresponding resource policies (bucket policy, bucket ACL, and object ACL) are also evaluated at the same time.
Bucket	In this context, ECS evaluates policies that are owned by the namespace that owns the bucket. If the namespace that owns the object in the request is not same as the bucket owner, in the bucket context the policies are checked to verify that the bucket owner has not explicitly denied access to the object. If there is an explicit deny set on the object, then the request is not authorized.

Context	Description
Object	In this context, the requester must have permissions from the object owner to perform a specific object operation. In this step, the object ACL is evaluated if required.

S3 bucket operation authorization

The below diagram describes how the system evaluates the authorization request for an S3 bucket operation process:



In the S3 bucket operation authorization process, at first the system evaluates whether the requester is an ECS IAM user. If yes, then the request is evaluated against the user context and the bucket contexts. If both verifications are authorized, the access is granted. Else, it is denied.

The below table describes the summary of access details for the same and cross account bucket operation:

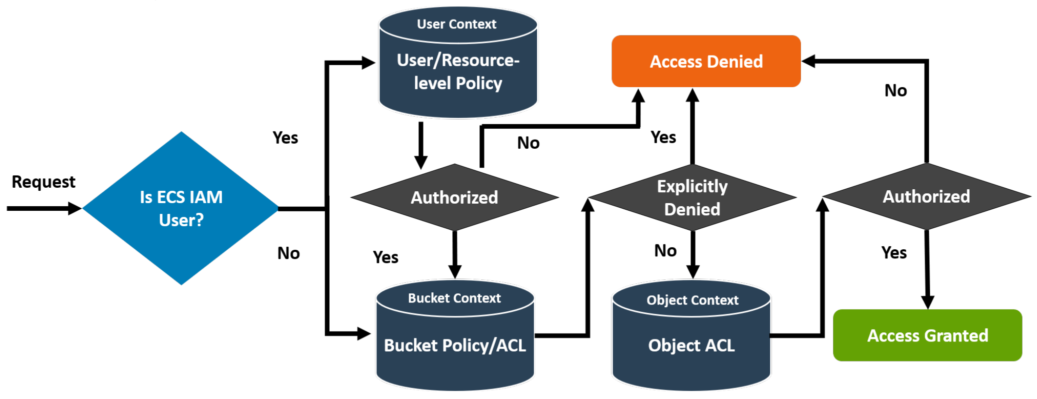
Bucket owner (account)	Requestor (account, user)	Comments
A1	U1	The user or the bucket policy determines the access. There is no bucket ACL check.
A1	U2	U2 needs IAM policy from A2, if A1 bucket policy does not a make a determination, then the system checks the bucket ACL.
A1	R1	IAM policy is not relevant for root user (R1). If A1 bucket policy does not a make a determination, then the system checks the bucket ACL.
A1	R2	IAM policy is not relevant for root user (R2). If A1 bucket policy does not a make a determination, then the system checks the bucket ACL.

NOTE: In this table, the following legends are used:

A1 = first account, A2 = second account, U1 = user from the first account, U2 = user from the second account, R1 = root user from the first account, and R2 = root user from the second account.

S3 object operation authorization

The below diagram describes how the system evaluates the authorization request for an S3 object operation process:



In the S3 object operation authorization process, at first the system evaluates whether the requester is an ECS IAM user. If yes, then the request is evaluated against the user, bucket, and object contexts. If these three contexts verifications are authorized, the access is granted. Else, it is denied.

The below table describes the summary of access details for the same and cross account bucket operation:

Bucket owner (account)	Object owner (account)	Requestor	Comments
A1	A1	U1	Access is determined by the user and/or by the bucket policy. No object ACL check
A1	A1	U2	U2 needs IAM policy from A2 and if A1 bucket policy does not a make a determination, then the system checks the object ACL
A1	A1	R1	IAM policy not relevant for R1. If A1 bucket policy does not a make a determination, then the system checks the object ACL
A1	A1	R2	IAM policy not relevant for R2. If A1 bucket policy does not a make a determination, then the system checks the object ACL
A1	A2	U1	U1 needs IAM policy or bucket policy allow. Object ACL must allow A1 access.
A1	A2	U2	U2 needs IAM policy allow. Bucket policy should not deny. **NOTE:** Bucket policy cannot allow access.
A1	A2	U3	U3 needs IAM policy allow. Bucket policy should not deny. Object ACL must allow A3 access. **NOTE:** Bucket policy cannot allow access.
A1	A2	R1	IAM policy not relevant. Bucket policy should not be deny. Object ACL needs to allow A1 access. **NOTE:** Bucket policy cannot allow access.
A1	A2	R2	IAM policy not relevant. Bucket policy should not be deny. Object ACL must allow A2 access. **NOTE:** Bucket policy cannot allow access.
A1	A2	R3	IAM policy not relevant. Bucket policy should not be deny. Object ACL must allow A3 access. **NOTE:** Bucket policy cannot allow access.

NOTE: In this table, the following legends are used:

Bucket owner (account)	Object owner (account)	Requestor	Comments
A1 = first account, A2 = second account, A3 = third account, U1 = user from the first account, U2 = user from the second account, U3 = user from the third account, R1 = root user from the first account, R2 = root user from the second account, and R3 = root user from the third account.			

ECS IAM and STS resources requests

The following procedure describes how the system evaluates the authorization requests on ECS IAM and STS resources within one namespace:

1. Deny evaluation - By default, all requests are denied (implicit deny). PEM evaluates all policies within the account that apply to the request. These include resource-based policies, permissions boundaries, role session policies, and identity-based policies. In all these policies, enforcement code looks for a `Deny` statement that applies to the request (explicit deny). If the code finds even one explicit deny that applies, the code returns a final decision of `Deny`. If there is no explicit deny, the evaluation continues.
2. Resource-based policies - If the requested resource has a resource-based policy that allows the principal entity to perform the requested action, then the code returns a final decision of `Allow`. If there is no resource-based policy, or if the policy does not include an Allow statement, then the code continues. This logic can behave differently if you specify the ARN of an IECS AM role or user as the principal of the resource-based policy. Someone can use session policies to create a temporary credential session for that role or federated user. In that case, the effective permissions for the session might not exceed those allowed by the identity-based policy of the user or role.
3. IAM permissions boundaries - The enforcement code then checks whether the IAM entity that is used by the principal has a permissions boundary. If the policy that is used to set the permissions boundary does not allow the requested action, then the request is implicitly denied. The code returns a final decision of `Deny`. If there is no permissions boundary, or if the permissions boundary allows the requested action, the evaluation continues.
4. Session policies - The code then checks whether the principal entity is using a session that was assumed by passing a session policy. You can pass a session policy while using temporary credentials for a role or federated user. If the session policy is present and does not allow the requested action, then the request is implicitly denied. The code returns a final decision of `Deny`. If there is no session policy, or if the policy allows the requested action, the code continues.
5. Identity-based policies - The code then checks the identity-based policies for the principal entity. For an IAM user, these include user policies and policies from groups to which the user belongs. If any statement in any applicable identity-based policies allows the requested action, then the PEM evaluation returns a final decision of `Allow`. If there are no statements that allow the requested action, then the request is implicitly denied, and the code returns a final decision of `DenyErrors` that is any errors that are encountered by PEM during the evaluation will throw an exception and stops evaluation.

Secure Token Service

The Security Token Service (STS) enables you to request temporary credentials, for IAM users or for other users that are externally authenticated (SAML).

ECS IAM supports the following three STS APIs:

- `AssumeRole` - Provides a way for a trusted IAM user to assume a role and get temporary credentials for accessing ECS S3 and IAM resources.

  NOTE: The `AssumeRole` API requires the standard AWS S3 authentication header for an IAM user to get authenticated.
- `AssumeRoleWithSAML` - Provides a way for an external user to assume a role and get temporary credentials using SAML assertions generated by a trusted external identity provider.
- `GetFederationToken` - Provides a way for the IAM user, who has permission, to get temporary security credential for federated users.

NOTE: The temporary credentials from the `AssumeRole`, `AssumeRoleWithSAML`, and `GetFederationToken` APIs consist of an access key ID, secret access key, and a session token. These temporary credentials cannot be revoked.

Accessing accounts using AssumeRole

`AssumeRole` returns a set of temporary security credentials that you can use to access IAM and S3 resources.

NOTE: The role trust relationship should grant permission to an entity to assume the role.

Same account access with AssumeRole

You can access the same account using `AssumeRole` by attaching a policy to the user (identical to the previous user in a different account) or by adding the user as a principal directly in the role trust policy.

Method	Example

Attaching a policy to the user	1. Trust policy for Role assumeRoleSameAccount in ns1:
	`{`
	`  "Version": "2012-10-17",`
	`  "Statement": [`
	`    {`
	`      "Effect": "Allow",`
	`      "Principal": {`
	`        "AWS": "urn:ecs:iam::ns1:root"`
	`      },`
	`      "Action": "sts:AssumeRole"`
	`    }`
	`  ]`
	`}`
	2. Policy is attached to the user1 in ns1 to `AssumeRole`:
	`{`
	`  "Version": "2012-10-17",`
	`  "Statement": [`
	`    {`
	`      "Action": [`
	`        "sts:AssumeRole"`
	`      ],`
	`      "Resource": "urn:ecs:iam::ns1:role/`
	`assumeRoleSameAccount",`
	`      "Effect": "Allow",`
	`      "Sid": "VisualEditor0"`
	`    }`
	`  ]`
	`}`

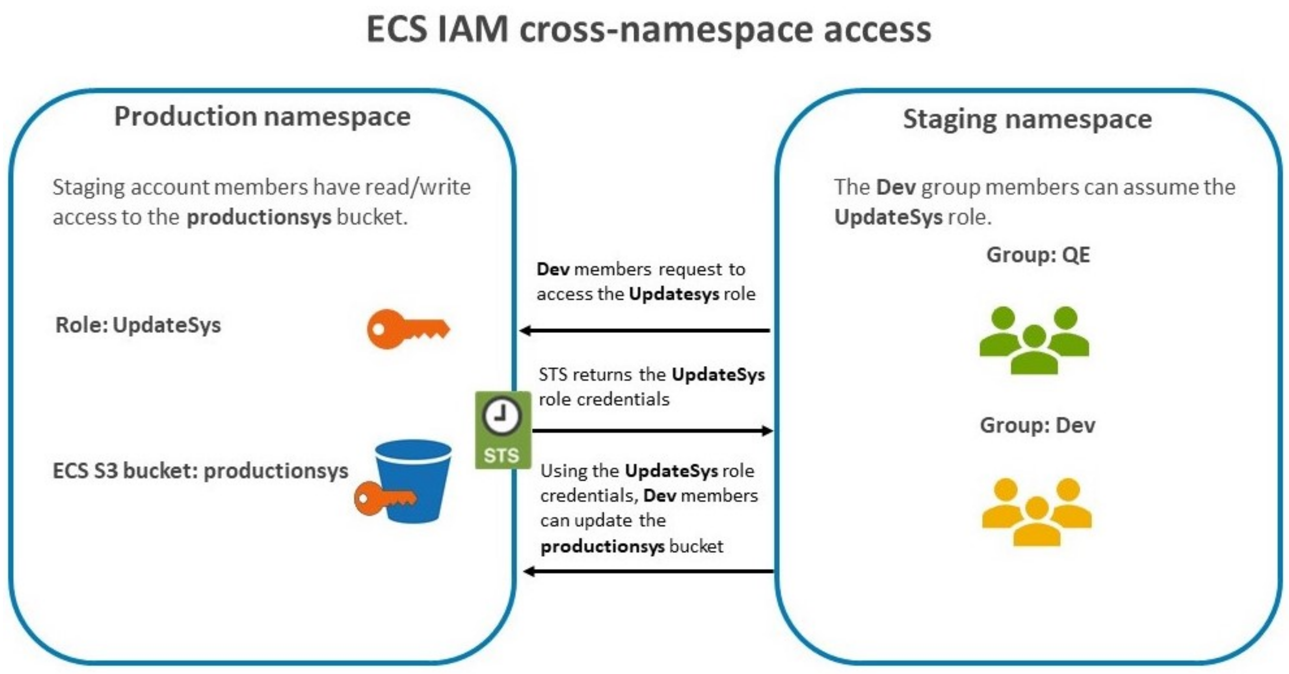
Adding the user to the role trust policy	Trust policy for Role in ns1 with an ECS IAM user:
	`{`
	`  "Version": "2012-10-17",`
	`  "Statement": [`
	`    {`
	`      "Effect": "Allow",`
	`      "Principal": {`
	`        "AWS": "urn:ecs:iam::ns1:user/user1"`
	`      },`
	`      "Action": "sts:AssumeRole"`
	`    }`
	`  ]`
	`}`

Cross account access with AssumeRole

By default, an ECS IAM user in one namespace has no access to buckets in another namespace. However, you can access different accounts using the role trust policy through `AssumeRole`.

Your account must be trusted by the role to assume a role from a different account. The trust relationship is defined in the role trust policy when the role is created. That trust policy states which accounts are allowed to delegate that access to users in the account. Also, ensure that you have permissions that are delegated from the user account administrator. The administrator must attach a policy that allows you to call `AssumeRole` for the Amazon Resource Name (ARN) of the role in the other account.

For example, your organization has multiple namespaces. From which, you segregate a staging environment from a production environment. Certain users such as developers from the staging namespace may also want to access the production namespace when you move the staging environment to the production.


ECS IAM cross-namespace access
Production namespace
Staging account members have read/write access to the productionsys bucket.
Role: UpdateSys
ECS S3 bucket: productionsys
STS
Dev members request to access the Updatesys role
STS returns the UpdateSys role credentials
Using the UpdateSys role credentials, Dev members can update the productionsys bucket
Staging namespace
The Dev group members can assume the UpdateSys role.
Group: QE
Group: Dev

- For this scenario, the admin creates two groups for the staging account namely Dev and QE, and each group has its own policy.
- In the production namespace, the administrator performs the following:
  - Specifies a trust policy to the role to state that the staging account as a Principal. So that the authorized users from the staging account can use that role.
  - Specifies which role users have read and write permissions to the productionsys bucket through a permissions policy.
  - Shares the namespace and role information with the users who need to assume the role.
- In the staging namespace, the administrator grants permission to the Dev group to assume the UpdateSys role. By doing this, the Dev group members can switch their role to the required and permitted role. For example, the Dev group members can switch their role to the UpdateSys role in the production namespace. Other users such as QE group members cannot switch their role. Hence, they cannot access the productionsys bucket.

In this process, STS verifies whether the requester is a trusted entity. After verifying, it returns temporary credentials to the authorized users to perform the required actions.

Example

1. Trust policy for Role in ns1:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "urn:ecs:iam::ns2:root"
      },
```

Example

```
          "Action": "sts:AssumeRole"
        }
      ]
    }
```

2. Policy that is attached to the user in ns2 to `AssumeRole`:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "urn:ecs:iam::ns1:role/assumeRoleCrossAccount",
      "Effect": "Allow",
      "Sid": "VisualEditor0"
    }
  ]
}
```

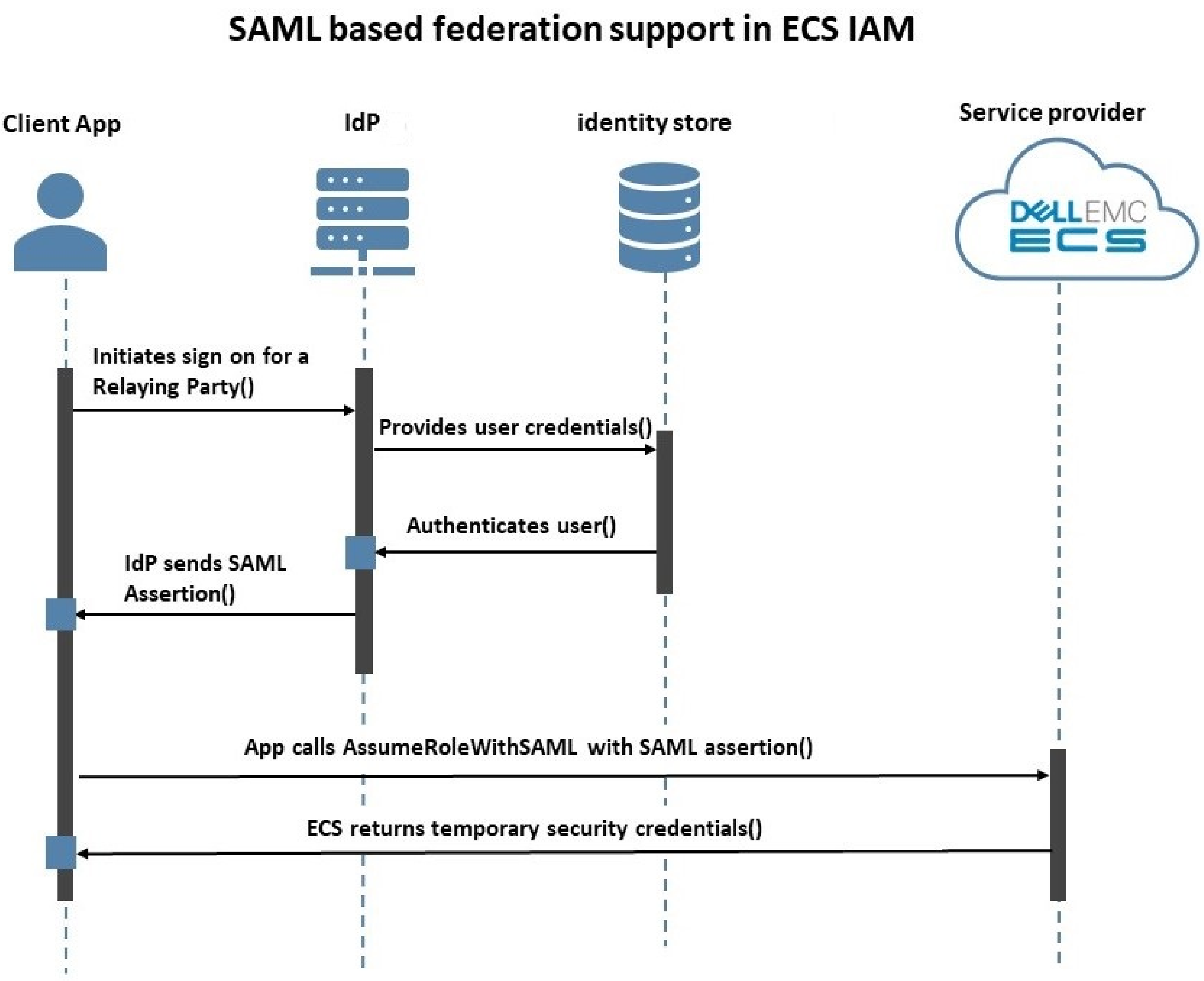
ECS IAM SAML support

Security Assertion Markup Language (SAML) is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider (IdP) and a service provider.

ECS supports integration with SAML 2.0 compliant Identity Providers (IdPs). Though other SAML 2.0 compliant Identity Providers should work, ECS has qualified Active Directory Federation Services and Azure AD as IdP. Contact ECS Remote Support for assistance on SAML supported IdPs.

The IdP must be able to generate SAML 2.0 Assertions with the following:

- The referral must be SAML 2.0 compliant as set out in https://datatracker.ietf.org/doc/html/rfc7522.
- The following claim attributes must be mapped in the IdP generated SAML assertion:
  - **NameId**: Must be formatted as a username.
  - **RoleSessionName**: The *https://aws.amazon.com/SAML/Attributes/RoleSessionName* must be mapped to the email address of the user. Example: bloke@dell.com.
  - **Roles**: The *https://aws.amazon.com/SAML/Attributes/Role* must be mapped to the SAML Provider URN, Role URN. Example: urn:ecs:iam::s3:saml-provider/provider1,urn:ecs:iam::s3:role/SAML-RW-Access.


SAML based federation support in ECS IAM
Client App
IdP
identity store
Service provider
DELL EMC ECS
Initiates sign on for a Relaying Party()
Provides user credentials()
Authenticates user()
IdP sends SAML Assertion()
App calls AssumeRoleWithSAML with SAML assertion()
ECS returns temporary security credentials()

SAML-compliant provider setup

ECS supports integration with SAML 2.0 compliant Identity Providers (IdPs). The IdP must be able to generate SAML 2.0. Here, an example with ADFS is used for demonstration purpose.

- Download the Identity Provider (ADFS) metadata file. The default URL to download ADFS metadata is **https://[server-name]/FederationMetadata/2007-06/FederationMetadata.xml**.
- Upload the downloaded metadata xml file when creating Identity provider for a namespace.
- To create Identity provider in the ECS Portal, perform the following:
  - Go to **Manage** > **Identity and Access (S3)** > **Identity Provider**.
  - Select a namespace.
  - Click **NEW IDENTITY PROVIDER**.
- In order to establish trust relationship between ECS and ADFS, ECS metadata xml file is required.
- To create ECS metadata file, base64 encoded Java keystore, alias that is used for the key and password is required.
- To create ECS metadata file, go to **Manage** > **Identity and Access (S3)** > **SAML Service Provider Metadata**. Provide the required information as mentioned above and download the metadata file.
- Establish trust relationship between ECS and ADFS using the downloaded ECS metadata file.
- Add claim rules in ADFS to add the required elements such as `NameId`, `RoleSessionName`, and Roles to the SAML authentication process.

AssumeRoleWithSAML

In order to use `AssumeRoleWithSAML`, you must configure your SAML identity provider (IdP) to issue the claims required by ECS.

- IAM role must be created that specifies this SAML Provider in the trust policy.
- `AssumeRoleWithSAML` returns a set of temporary security credentials for users who have been authenticated through a SAML authentication response.
- This operation provides a mechanism for tying an enterprise identity store or directory to role-based access without user-specific credentials or configuration.
- Calling `AssumeRoleWithSAML` does not require the use of ECS security credentials. The identity of the caller is validated by the claims that are provided in the SAML Assertions by the identity provider.
- Temporary credentials consist of an access key ID, a secret access key, and a security token.
- Following condition keys are supported in the `AssumeRolePolicyDocument`.
    - `saml:aud`
    - `saml:iss`
    - `saml:sub`
    - `saml:sub_type`
    - `saml:edupersonorgdn`
    - `saml:namequalifier`

Example role trust policy

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Federated":"urn:aws:iam::s3:saml-provider/provider1"
         },
         "Action":"sts:AssumeRoleWithSAML",
         "Condition":{
            "StringEquals":{
               "SAML:sub":"<Idp>\\Bob",
               "SAML:aud":"https://10.247.179.105/saml",
               "SAML:eduPersonOrgDN":[
                  "ECS",
                  "Atmos"
               ],
               "SAML:iss":"http://AD.<Idp>.emc.com/<Idp>/services/trust"
            }
         }
      }
   ]
}
```

Attributes in SAML assertion

The following attributes are required in SAML assertion.

- https://aws.amazon.com/SAML/Attributes/RoleSessionName
- https://aws.amazon.com/SAML/Attributes/Role

NOTE:

- The `Role` attribute must be of the format `SAML Provider URN, Role URN` to be used from ECS for an AD Group.
- If you must use `saml:edupersonorgdn`, then `oid` attribute must also be present in the SAML assertion as `urn:oid:1.3.6.1.4.1.5923.1.1.1.3`. However, it is optional to use this attribute.

For example:

```
<AttributeStatement>
         <Attribute Name="https://aws.amazon.com/SAML/Attributes/RoleSessionName">
            <AttributeValue>Bob@emc.com</AttributeValue>
```

```
          </Attribute>
          <Attribute Name="https://aws.amazon.com/SAML/Attributes/Role">
             <AttributeValue>urn:ecs:iam::s3:saml-provider/provider1,urn:ecs:iam::s3:role/
<Idp>-Dev</AttributeValue>
             <AttributeValue>urn:ecs:iam::s3:saml-provider/provider1,urn:ecs:iam::s3:role/
<Idp>-Production</AttributeValue>
          </Attribute>
          <Attribute Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.3">
             <AttributeValue>ECS</AttributeValue>
          </Attribute>
       </AttributeStatement>
```

User-specific access using SAML keys

It is recommended to specify permissions based on the users identity when creating access policies in ECS IAM.

As to create policies that contain user-specific information, the user identity should be available in SAML keys. The following SAML keys can be used in policy conditions to create unique user identifiers.

SAML keys	Description
saml:namequalifier	A hash value based on the concatenation of the Issuer response value (`saml:iss`) and a string with the ECS namespace (account ID) and the friendly name (the last part of the ARN) of the SAML provider in IAM. The namespace (account ID) and provider name must be separated by a '/' as in "123456789012/provider_name". The combination of `NameQualifier` and `Subject` can be used to uniquely identify a federated user. The following pseudocode shows how this value is calculated. In this pseudocode, "+" indicates concatenation, SHA1 represents a function that produces a message digest using SHA-1, and Base64 represents a function that produces Base-64 encoded version of the hash output. Base64 = (SHA1 ("https://example.com/saml" + "ECSNamespace" + "/SamlProvider"))
saml:sub	This is the subject of the claim, which includes a value that uniquely identifies an individual user within an organization. For example, `_3e52ef03414f3464d2461c00ebae0152c25fb88bbc`.
saml:sub_type	This key can be persistent, transient, or the full Format URI from the `Subject` and `NameID` elements used in your SAML assertion. A value of persistent indicates that the value in `saml:sub` is the same for a user across all sessions. If the value is transient, the user has a different `saml:sub` value for each session.

IAM Policy

The following example shows a permission policy that uses the preceding keys to grant permissions to a user-specific folder in Amazon S3. The policy assumes that the Amazon S3 objects are identified using a prefix that includes both `saml:namequalifier` and `saml:sub`. Notice that the `Condition` element includes a test to be sure that `saml:sub_type` is set to persistent. If it is set to transient, the `saml:sub` value for the user can be different for each session, and the combination of values should not be used to identify user-specific folders.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3:::exampleECSBucket/backup/${saml:namequalifier}/${saml:sub}",
      "arn:aws:s3:::exampleECSBucket/backup/${saml:namequalifier}/${saml:sub}/*"
    ],
    "Condition": {"StringEquals": {"saml:sub_type": "persistent"}}
  }
}
```

Example with sample values

- Create a role using `AssumeRoleWithSAML`. See AssumeRoleWithSAML for more information.
- Attach an IAM policy to this role as below.

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:PutObject",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3:::exampleECSBucket/backup/${saml:namequalifier}/${saml:sub}",
      "arn:aws:s3:::exampleECSBucket/backup/${saml:namequalifier}/${saml:sub}/*"
    ],
    "Condition": {"StringEquals": {"saml:sub_type": "persistent"}}
  }
}
```

The values in the above example are as follows:

- saml:iss = http://AD.adfs.emc.com/adfs/services/trust. See ECS IAM supported condition keys for the list of SAML condition keys.
- account = `s3`
- providername = `provider1`
- saml:sub = `ADFS\Bob`
- Base64 = `SHA1 ("http://AD.adfs.emc.com/adfs/services/trust " + "s3" + "/provider1")`
- SHA1 = `BB9445BB2D9C57D519ACEBD08EFD428076522D5B`
- Base64 of `BB9445BB2D9C57D519ACEBD08EFD428076522D5B` is `u5RFuy2cV9UZrOvQjv1CgHZSLVs=`.

GetFederationToken

`GetFederationToken` provides a set of temporary security credentials (consisting of an access key ID, a secret access key, and a security token) to a federated IAM user for use only in S3 service. It is a part of STS along with `AssumeRole` and `AssumeRoleWithSAML`.

GetFederationToken permissions

You can specify maximum of 10 managed policies for permissions for `GetFederationToken` API. If you have not specified policies, the temporary security credentials become ineffective. However, if the resource policy has permission to access the resource, user can access the resource using the temporary security credentials.

When you pass session policy (inline or managed policy) the system restricts permissions that are available to the IAM user by allowing only a subset of the permissions that are granted to the IAM user. The passed policy cannot grant more permissions than those granted to the IAM user. The final permissions for the federated user are the most restrictive set based on the intersection of the passed policy and the IAM user policy.

The following figures show a visual representation of how the policies interact to determine permissions for the temporary security credentials.

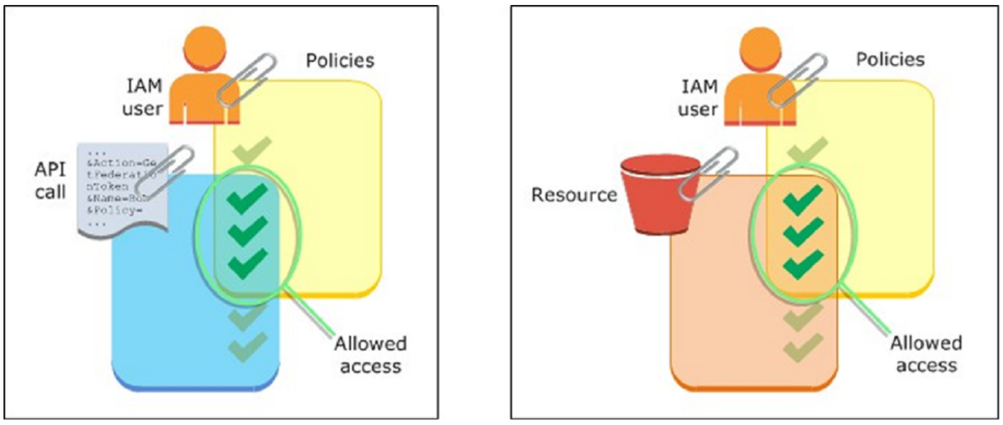


Figure 2. GetFederationToken - determining the policy permissions

NOTE: The IAM user needs the policy attached to get permission to use the `GetFederationToken` API. The IAM user also needs allow action on `sts:TagSession` to get permission to add tags on STS session. You cannot use these credentials in IAM operations and STS operations.

GetFederationToken request parameters

This section lists summary of `GetFederationToken` API request parameters:

Table 29. Request parameters of GetFederationToken API

Name	Description
DurationSeconds	The duration in seconds that the session should last. Acceptable durations for federation sessions range from 900 seconds (15 minutes) to 129,600 seconds (36 hours) with 43,200 seconds (12 hours) as the default. ● Type: Integer ● Valid Range: Minimum value of 900 and maximum value of 129600 ● Required: No
Name	The name of the federated user that is used as an identifier for the temporary security credentials. ● Type: String ● Length Constraints: Minimum length of 2 and maximum length of 32 ● Pattern: [\w+=,.@-]* ● Required: Yes
Tags	● A list of session tags and each session tag consists of a key name and an associated value. ● You can pass a session tag with the same key as a tag that is already attached to the user you are federating. When you do so, session tags override a user tag with the same key. ● Tag key value pairs are not case-sensitive, but case is preserved. You cannot have separate Department and department tag keys. For example, the user has the Department=Marketing tag and you pass the department=engineering session tag. Department and department are not saved as separate tags, and the session tag that is passed in the request takes precedence over the user tag. ● When you use the temporary credentials that the `GetFederationToken` operation returned, the session principal tags include the user tags and the passed session tags. ● You can pass up to 50 session tags.
Policy	An IAM Policy is passed with the `GetFederationToken` call and evaluated along with the policy or policies that are attached to the IAM user whose credentials are used to call `GetFederationToken`. ● Type: String

Table 29. Request parameters of GetFederationToken API (continued)

Name	Description
	● Length Constraints: Minimum length of 1 and maximum length of 2048 ● Pattern: [\u0009\u000A\u000D\u0020-\u00FF]+ ● Required: No
PolicyArns	The URNs of the IAM-managed policies that you want to use as a managed session policy. The policies must exist in the same account as the IAM user that is requesting federated access. You can pass up to 10 managed policies to use as managed session policies. The plaintext that you use for both inline and managed session policies should not exceed 2,048 characters. ● Type: Array of objects ● Required: No

Example usages

- A sample inline or managed policy that can be attached to an IAM user to grant GetFederationToken and TagSession permissions:

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
       "Sid": "VisualEditor0",
       "Effect": "Allow",
       "Action": [
         "sts:TagSession",
         "sts:GetFederationToken"
        ],
        "Resource": "*"
     }
    ]
}
```

- GetFederationToken using AWS CLI:

```
./aws sts --profile ecsiamuser1 get-federation-token --name BobTemp --policy file://
s3ReadOnlyPolicy --endpoint-
url=https://IP:4443/sts --no-verify-ssl
{
    "Credentials": {
        "AccessKeyId": "ASIA20B9DB02921B9D93",
        "SecretAccessKey": "PXqhhj5gMMFY0aSBNXoaP_xWgfJXFlkpdMySmqnY8Fk",
        "SessionToken":
"CgJzMxIUQUlEQUU5QUIwNzY5NzExMkEzMTEqFEFTSUEyMEI5REIwMjkyMUI5RDkzMlBNYXN0ZXJLZXlSZWNvc
mQtM2RhNGUyZ
TZjMjBjYjM4NjQ1ZWUyZWI5ZDVlMWM1MTgyYmEwYWI0NzViMTA4OGFhOTQwZjMyMmUwMjVhM2NkNTipg4D59C9
C5wF7CiAgICAiVmVyc2lvbiI6ICIyMDEyLTE
wLTE3IiwKICAgICJTdGF0ZW1lbnQiOiBbCiAgICAgICAgewogICAgICAgICAgICAiRWZmZWN0IjogIkFsbG93I
iwKICAgICAgICAgICAgIkFjdGlvbiI6IFsKI
CAgICAgICAgICAgICAgICJzMzpHZXQqIiwKICAgICAgICAgICAgICAgICJzMzpMaXN0KiIKICAgICAgICAgICA
gXSwKICAgICAgICAgICAgIlJlc291cmNlIjo
gIioiCiAgICAgICAgfQogICAgXQp9CgpohMX_kAZyHXVybjplY3M6aWFtOjpzMzp1c2VyL2lhbXVzZXIxegdCb
2JUZW1w",
        "Expiration": "2022-03-03T09:32:52+00:00"
    },
        "FederatedUser": { "FederatedUserId": "s3:BobTemp",
  "Arn": "urn:ecs:sts::s3:federated-user/BobTemp"
        },
        "PackedPolicySize": 11
}
```

- A sample usage of temporary credentials to access S3 API:

```
    lrmk # export AWS_ACCESS_KEY_ID=ASIA20B9DB02921B9D93
    lrmk # export AWS_SECRET_ACCESS_KEY=PXqhhj5gMMFY0aSBNXoaP_xWgfJXFlkpdMySmqnY8Fk
    lrmk # export
AWS_SESSION_TOKEN=CgJzMxIUQUlEQUU5QUIwNzY5NzExMkEzMTEqFEFTSUEyMEI5REIwMjkyMUI5RDkzMlBN
YXN0ZXJLZXlSZ
```

```
WNvcmQtM2RhNGUyZTZjMjBjYjM4NjQ1ZWUyZWI5ZDVlMWM1MTgyYmEwYWI0NzViMTA4OGFhOTQwZjMyMmUwMjV
hM2NkNTipg4D5
9C9C5wF7CiAgICAiVmVyc2lvbiI6ICIyMDEyLTEwLTE3IiwKICAgICJTdGF0ZW1lbnQiOiBbCiAgICAgICAgew
ogICAgICAgICA
gICAiRWZmZWN0IjogIkFsbG93IiwKICAgICAgICAgICAgIkFjdGlvbiI6IFsKICAgICAgICAgICAgICAgICJzM
zpHZXQqIiwKIC
AgICAgICAgICAgICAgICJzMzpMaXN0KiIKICAgICAgICAgICAgXSwKICAgICAgICAgIlJlc291cmNlIjog
IioiCiAgICAgI
CAgfQogICAgXQp9CgpohMX_kAZyHXVybjplY3M6aWFtOjpzMzp1c2VyL2lhbXVzZXIxegdCb2JUZW1w
    lrmk # ./aws s3api get-object --bucket bucket1 --key obj1 outfile --endpoint-
url=http://$IP:9020
    -
-no-verify-ssl
    {
    "LastModified": "2022-09-12T20:27:32+00:00",
    "ContentLength": 11,
    "ETag": "\"093ec021131c823e8ab89c78e458f72a\"",
    "ContentType": "application/octet-stream",
    "ServerSideEncryption": "AES256",
    "Metadata": {}
}
```

- A sample resource policy with ARN for the federated user:

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
          "Sid": "Statement1",
          "Effect": "Allow",
          "Principal": {
              "AWS": [
                  "urn:ecs:sts::s3:federated-user/BobTemp"
              ]
          },
          "Action": [
             "s3:GetObject",
             "s3:GetObjectAcl",
             "s3:ListBucket"
          ],
          "Resource": [
             "arn:aws:s3:::seetbucket/*",
             "arn:aws:s3:::seetbucket"
          ],
          "Condition": {
             "StringEquals": {
                 "aws:PrincipalTag/department": "Engineering"
             }
         }
      }
   ]
}
```

- A sample resource policy with federated ID in condition:

```
{
   "Version": "2012-10-17",
   "Statement":[
      {
           "Sid": "Statement3",
           "Effect": "Allow",
           "Principal": {
               "AWS": "urn:ecs:iam::s3:root"
         },
         "Action": [
             "s3:GetObject",
             "s3:GetObjectAcl"
         ],
         "Resource": [
           "arn:aws:s3:::bucket1/*",
           "arn:aws:s3:::bucket1"
         ],
         "Condition": {
                "StringEquals": {
                      "aws:userid": "s3:iamu1temp"
                }
            }
        }
    ]
}
```

4

OpenStack Swift

This section describes the supported methods, the ECS extensions, and the mechanism for authentication.

ECS supports the OpenStack Swift API and can be used with applications that support that API. This section describes supported methods, the ECS extensions, and the mechanism for authentication.

Topics:

OpenStack Swift support in ECS

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

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

Table 30. Port details

Protocol	Ports
HTTP	9024
HTTPS	9025

Examples showing the use of the OpenStack Swift API can be found in OpenStack API Examples.

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

Using OpenStack with ECS requires you to configure ECS so that it can authenticate OpenStack users. You can see Authentication using ECS Keystone V3 integration for information about configuring authentication.

OpenStack Swift supported operations

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

This information is taken from the Object Storage API V1 section of the OpenStack API Reference documentation.

Supported OpenStack Swift calls

Table 31. OpenStack Swift supported calls

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

Table 32. Additional features

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

Unsupported OpenStack Swift calls

Table 33. OpenStack Swift unsupported calls

Method	Path	Description
COPY	v1/{account}/{container}/{object}	Copy operation can be achieved using PUT v1/{account}/{container}/{object} with X-Copy-From header.

Table 33. OpenStack Swift unsupported calls (continued)

Method	Path	Description
HEAD	v1/{account}	Retrieve the account metadata. Not fully supported as returns zero for the bytes stored (X-Account-Bytes-Used).

Swift extensions

ECS supports a number of extensions to the Swift API.

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

- Swift byte range extensions
- Retention
- File system enabled

Swift byte range extensions

The following ECS extensions are provided for performing the following operations on Swift byte ranges:

- Updating a byte range within an object
- Overwriting part of an object
- Appending data to an object
- Reading multiple byte ranges within an object

Updating a byte range within an object

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

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

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

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

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

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

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

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

```
PUT /container1/object1 HTTP/1.1
Content-Length: 5
```

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

green

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

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

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

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

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

Overwriting part of an object

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

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

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

```
PUT /container1/object1 HTTP/1.1
Content-Length: 9
Range: bytes=10-
ACCEPT: application/json,application/xml,text/html,application/octet-stream
Date: Mon, 12 Mar 2018 20:51:41 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:uwPjDAgmazCP5lu77Zvbo+CiT4Q=
Accept-Encoding: gzip, deflate, compress

brown cat

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

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

```
GET /container1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

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

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

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

Appending data to an object

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

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

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

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

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

and cat

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

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

```
GET /container1/object1 HTTP/1.1
ACCEPT: application/json,application/xml,text/html,application/octet-stream
Date: Mon, 12 Mar 2018 20:46:56 -0000
x-emc-namespace: emc
Content-Type: application/octet-stream
Authorization: AWS wuser1:D8FSE8JoLl0MTQcFmd4nG1gMDTg=
Accept-Encoding: gzip, deflate, compress
```

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

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

Reading multiple byte ranges within an object

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

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

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

```
GET /container1/object1 HTTP/1.1
Date: Mon, 12 Mar 2018 20:51:55 -0000
x-emc-namespace: emc
Range: bytes==4-8,41-44
Content-Type: application/octet-stream
Authorization: AWS wuser1:/UQpdxNqZtyDkzGbK169GzhZmt4=
Accept-Encoding: gzip, deflate, compress

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

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

Retention

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

Retention can be set in the following ways:

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

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

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

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

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

File system enabled

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

Enabling FS access

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

Cross-head support for FS

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

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

You can find more information on the cross-head support with file systems in the *ECS Administration Guide* which is available from the https://www.dell.com/support/.

S3 and Swift interoperability

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

For details, see S3 and Swift interoperability.

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

OpenStack Swift authentication

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

v1	ECS enables object users to authenticate with the ECS Swift service and obtain an authentication token that can be used when making subsequent API calls to the ECS Swift service. See OpenStack Version 1 authentication.
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
v3	ECS validates Keystone V3 users that present tokens scoped to a Keystone project. See Authentication using ECS Keystone V3 integration.

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

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

Assigning Swift credentials to ECS object users is described in Create Swift users in the ECS Portal.

Create Swift users in the ECS Portal

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

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

1. In the ECS Portal, select **Manage** > **Users**.
2. On the **User Management** page, you can create a new object user who will access the ECS object store through the Swift object protocol in one of two ways:
   a. Click **New Object User** to create a new object user.
      - On the **New Object User** page, in the **Name** field, type a name for the object user.
      - In the **Namespace** field, select the namespace to which the user belongs.
      - Click **Next to Add Passwords**.

   b. Click **Edit** in the **Actions** column beside an existing user and add a Swift password to the existing user.
3. On the **Update Passwords for User <*username*>** page, in the **Swift Groups** field, enter the Swift group to which the user belongs.

   If you specify the `admin` group, users will automatically be able to perform all container operations. If you specify a different group, that group must be given permissions on the container. Refer to Authorization on Container for more information on container authorization.
4. In the **Swift password** field, type a password for the Swift user.
5. Click **Set Groups & Password**.

OpenStack Version 1 authentication

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

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

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

   Request:

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

   Response:

   ```
   HTTP 200
   ```

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

   Request:

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

Response:

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

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

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.

ECS provides an implementation of the OpenStack Swift V2 identity service which enables a Swift application that uses V2 authentication to authenticate users. Users must be ECS object users who have been assigned OpenStack Swift credentials which enable them to access the ECS object store using the Swift protocol.

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

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

The two articles listed below provide important background information.

- OpenStack Keystone Workflow and Token Scoping
- Authenticate for Admin API

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

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

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

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

2. Retrieve tenant information associated with the unscoped token.

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

   The response looks like the following.

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

3. Retrieve the scoped token along with the storageUrl.

   ```
   curl -v -X POST -H 'ACCEPT: application/json' -H "Content-Type: application/json" -d
   '{"auth": {"tenantName" : "s3",
                             "token":{"id" : ecs_d668b72a011c4edf960324ab2e87438b"}}}'
   http://203.0.113.10:9024/v2.0/tokens
   ```

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

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

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

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

Authentication using ECS Keystone V3 integration

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

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

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

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

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

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

Domains

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

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

Configure OpenStack Swift and ECS integration

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

The following prerequisites apply:

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

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

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

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

   See Add a Keystone authentication provider.
4. Create an ECS namespace that has the same ID as the Keystone project/account that the users want to authenticate belong to.

   Obtain the Keystone project ID.

   a. In the ECS Portal, select **Manage** > **Namespace** > **New Namespace**
   b. Enter the name of the namespace.

      This should be the name of the Swift project.
   c. Enter the namespace administrator account as the **User Admin**.

      This should be the name of a management user that has previously been created.
   d. Configure any other namespace settings that you require.

      For more information about Namespace settings and about creating users in ECS, see the *ECS Administration Guide* which is available from the https://www.dell.com/support/.

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

Add a Keystone authentication provider

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

- This operation requires the Security Administrator role in ECS.
- You can add only one Keystone authentication provider.
- Obtain the authentication provider information listed in Keystone authentication provider settings.

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

   The required fields are displayed.
4. Type values in the **Name**, **Description**, **Server URL**, **Keystone Administrator**, and **Admin Password** fields. For more information about these fields, see Keystone authentication provider settings.
5. Click **Save**.

Keystone authentication provider settings

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

Table 34. Keystone authentication provider settings

Field	Description
Name	The name of the Keystone authentication provider. This name is used to identify the provider in ECS.
Description	Free text description of the authentication provider.
Type	Keystone V3.
Server URL	URl of the Keystone system that ECS connects to in order to validate Swift users.
Keystone Administrator	Username for an administrator of the Keystone system. ECS connects to the Keystone system using this username.
Admin Password	Password of the specified Keystone administrator.

Authorization on Container

OpenStack Swift authorization targets only containers.

Swift currently supports two types of authorization:

- Referral style authorization
- Group style authorization

ECS supports only group-based authorization.

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

Admin assigns read access to the container

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

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

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

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

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

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

Admin assigns write access to the container

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

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

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

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

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

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

Additional information about authorization can be found in: Container Operations.

ECS Swift error codes

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

Table 35. Error Codes

Error Code	HTTP Status Code	HTTP Status	Description
ERROR_NAMESPACE_NOT_FOUND	400	BAD_REQUEST	Namespace not found.
ERROR_KEYPOOL_NOT_FOUND	404	NOT_FOUND	Keypool not found.
ERROR_KEYPOOL_NOT_EMPTY	409	CONFLICT	Keypool not empty.
ERROR_OBJECT_NOT_FOUND	404	NOT_FOUND	Object not found.
ERROR_VERSION_NOT_FOUND	404	NOT_FOUND	Version not found.
ERROR_ACCESS_DENIED	403	FORBIDDEN	null
ERROR_SERVICE_BUSY	503	SERVICE_UNAVAILABLE	null
ERROR_PRECONDITION_FAILED	412	PRECONDITION_FAILED	null
ERROR_INVALID_ARGUMENT	400	BAD_REQUEST	Invalid argument.
ERROR_BAD_ETAG	422	SC_UNPROCESSABLE_ENTITY	Bad etag.
ERROR_PROJECT_NOT_FOUND	404	NOT_FOUND	SwiftException. NO_PROJECT_FOUND.
ERROR_NO_DEVICE	404	NOT_FOUND	SwiftException. NO_DATA_STORE_FOUND. //add 416- Requested Range Not Satisfiable to errorMap.
ERROR_INVALID_RANGE	422	SC_REQUESTED_RANGE_NOT_ SATISFIABLE	Requested range cannot be satisfied.
ERROR_INSUFFICIENT_STORAGE	507	SC_INSUFFICIENT_STORAGE	The server cannot process the request because there is not enough space on disk.
ERROR_RETENTION_INCORRECT	404	SC_NOT_FOUND	The specified retention does not exist.
ERROR_OBJECT_UNDER_RETENTION	409	SC_CONFLICT	The object is under retention and cannot be deleted or modified.
ERROR_METHOD_NOT_ALLOWED	403	SC_FORBIDDEN	Quota may have been exceeded.
ERROR_BUCKET_NOT_FOUND	404	NOT_FOUND	Bucket not found.
ERROR_KEYPOOL_OPERATION_NOT _SUPPORTED	400	BAD_REQUEST	VersionEnabled and FileSystemEnabled functionality is not supported.
ERROR_REP_GROUP_NOT_FOUND	400	BAD_REQUEST	Specified Replication Group is Invalid.
ERROR_OBJECT_METADATA_REACH _MAXIMUM	400	BAD_REQUEST	Metadata exceeds max allowed length.
ERROR_KEYPOOL_LOCKED	409	CONFLICT	Bucket may be locked.
ERROR_INVALID_PART	409	CONFLICT	Segment eTag differs from that of the manifest.

Table 35. Error Codes (continued)

Error Code	HTTP Status Code	HTTP Status	Description
`ERROR_DELETE_DIRECTORY_NOT_EMPTY`	409	`CONFLICT`	Directory is not empty.
`ERROR_API_INVALID`	400	`BAD_REQUEST`	Cross head access is not supported.
`ERROR_DIRECTORYTABLE_TABLE_FULL`	503	`SERVICE_UNAVAILABLE`	Unable to handle request due to temporary overloading. Reduce request rate.

5

EMC Atmos

This section describes the support that ECS provides for EMC Atmos.

Topics:

EMC Atmos API support in ECS

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

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

Table 36. Port Details

Protocol	Ports
HTTP	9022
HTTPS	9023

The *Atmos Programmer's Guide* provides more information about the supported operations such as:

- Wire format compatibility for all supported operations, which also applies to the API operations exposed by ECS.
- Authenticating with the Atmos API and provides comprehensive examples for many of the supported features.

The *Atmos Programmer’s Guide* is available from http://support.emc.com.

Supported EMC Atmos REST API Calls

ECS supports a subset of the EMC Atmos API.

Table 37. 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		

Table 37. Supported Atmos REST API calls (continued)

Method	Path	Description
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> /rest/namespace/<path>	Get all object metadata
Object-space Operations		
GET	/rest/objects	List objects
GET	/rest/objects?listabletags	Get listable tags
Anonymous Access		
GET	/rest/objects/<ObjectId>?uid=<uid>&expires=<exp>&signature=<sig> /rest/namespace/<path>?uid=<uid>&expires=<exp>&signature=<sig>	Shareable URL

NOTE:
- The x-emc-wschecksum header is supported in ECS.
- The Atmos objects do not inherit ACL from the group ACL that is set at a bucket level.
  - If there is no user ACL provided, the ACL is inherited from the x-emc-useracl header.
  - If there is no group ACL provided, the Read ACL is used by default.
- `GET /rest/objects` does not support different response types with x-emc-accept. For example, text/plain is not supported.
- Read, Write, and Delete ACLs work in ECS the same as Atmos.
- POST /rest/objects supports the x-emc-object-id header to enable legacy (44 character) object Ids.

Atmos listable tags

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

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

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

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

Object ID length

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

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

Unsupported EMC Atmos REST API Calls

The following Atmos REST API calls are not supported.

Table 38. Unsupported Atmos REST API calls

Method	Path	Description
Object Versioning		
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.

Table 39. Call Details

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

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

API Extensions

ECS supports a number of extensions to the Atmos API.

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

Appending data to an object

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

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

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

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

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

and cat

HTTP/1.1 200 OK
x-emc-mtime: 1431626712933
Date: Mon, 17 Jun 2013 20:46:01 GMT
x-emc-policy: default
x-emc-utf8: true
x-emc-request-id: 0af9ed8d:14cc314a9bc:112f6:9
x-emc-delta: 8
x-emc-append-offset: 24
Content-Length: 0
Server: Jetty(7.6.4.v20120524)
```

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

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

ECS support for retention and retention expiration periods for Atmos objects

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

Retention periods

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

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

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

When a retention policy is set on the ECS namespace, set the retention period directly on the object. The object does not inherit the retention policy in the namespace.

Table 40. Atmos retention periods

Retention set on the	Using the	Notes
Object	Atmos API through the ● Header retention period in seconds: `'x-emc-retention-period:60'` ● User meta data (UMD), end date: `'x-emc-meta:user.maui.retentionEnable=true,user.maui.retentionEnd=2016-10-21:10:00Z'` ● Both header, and UMD: `'x-emc-meta:user.maui.retentionEnable = true,user.maui.retentionEnd=2016-10-21T18:14:30Z' -header 'x-emc-retention-period:60'`	● Retention can be set on the object while creating, or updating the object settings. ● Header retention period is defined in seconds. ● End date defines the UMD retention. ● If retention period is set from both the header and the UMD, the UMD attribute is checked first and takes precedence over the setting in the header. ● You cannot modify the retention period after it has been set on the object until the period has expired. ● When using the `x-emc` header to set retention ○ If one is defined, -1 sets an infinite retention period and disable the expiration period. ○ -2 disables the retention period set on the object.
ECS namespace	ECS Portal from the **New Namespace** or **Edit Namespace** page.	● If you want to set a retention period for an object, and a retention policy has been defined on the object user's namespace, you must still define a retention period directly on the object as described earlier. ● If a retention policy is set on the ECS namespace, and/or a retention period is set on a bucket within the namespace, and an object is created within the bucket, ECS retains the namespace, bucket, and object for the longest retention periods set for either the namespace, or bucket. ● If a retention period has been set on the object itself through the object header, ECS retains the object for the longest time set on the namespace, bucket, or object. ● If a retention end date is defined on an object through the Atmos API, ECS uses the Atmos API retention end date set on the object, and ignores the namespace retention policy, and bucket retention periods when creating the object. ● While applying a retention policy on a subtenant (bucket) containing Atmos objects, the retention policy is applied to both objects created in the subtenant after the retention policy was set, and objects that were created in the subtenant before the retention policy was set.
	ECS REST API `POST /object/namespaces/namespace/{namespace}/retention`	
ECS bucket	ECS Portal from the **New Bucket**, or **Edit Bucket** page.	
	ECS REST API `PUT /object/bucket/{bucketName}/retention`	

NOTE: For further details about Namespace Retention Policies and Bucket Retention Periods, see the ECS Administration Guide that is available on https://www.dell.com/support/.

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

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

Response
```

```
HTTP/1.1 201 Created
Date: Thu, 16 Feb 2017 19:28:17 GMT
x-emc-policy: default
x-emc-utf8: true
x-emc-request-id: 0af7b3e4:15a4849d95e:37c:0
x-emc-delta: 774
Location: /rest/objects/
0a40bd045f7373d367639f095d1db0d15acadb82d5d2cd108e2142f4be04635c-59bdb9b6-20c0-4f55-
bc91-9db728a58854
x-emc-mtime: 1487273295379
Content-Length: 0
Server: ViPR/1.0
```

Example: Request and response to get object metadata:

```
curl --head  -H "x-emc-date:Mon, 30 Jan 2017 16:56:35 GMT"
-H "x-emc-uid:7a2593be81374744adbf8e3983e7bd84/u1"
-H "x-emc-signature:CQgfoiIQ/DCif7TafcIskWyVpME="
http://10.247.179.228:9022/rest/objects/
d1bced53f2ebbcbc51af1d84747bd198d123d3b8585293a5bf0d32bb73c6cf4b-365f4482-c24a-4eca-
b24a-070efe29bf63

Response

HTTP/1.1 200 OK
Date: Mon, 30 Jan 2017 16:56:35 GMT
x-emc-mtime: 1485795387838
x-emc-retention-period: 21798212
x-emc-meta: user.maui.retentionEnd=2017-10-10T00:00:00Z,user.maui.retentionEnable=true,allow-inline-
update=false,atime=2017-01-30T16:45:48Z,ctime=2017-01-30T16:56:27Z,ctype=plain/text,data-
range=CAAQgFA=,dek=kq/W1Rg/
7qbmaCcLF8pFvqlDJ8+suPTdVddBBZFwZA86muG3P0Pb7w==,dekAlgo=AESKeyWrapRFC5649,etag=0-,fs-
mtime-
millisec=1485795387838,itime=2017-01-30T16:45:48Z,kekId=s3.7a2593be81374744adbf8e3983e7bd
843cdda755061bac6c12c06eb02800a7fee4b11ac2e03f62bb01eee02995068e56,keypoolid=s3.7a2593be8
1374744adbf8e3983e7bd84,keypoolname=7a2593be81374744adbf8e3983e7bd84,keyversion=0,mtime=2
017-01-30T16:56:27Z,namespace=s3,nlink=1,object-
name=,objectid=d1bced53f2ebbcbc51af1d84747bd198d123d3b8585293a5bf0d32bb73c6cf4b-365f4482-
c24a-4eca-
b24a-070efe29bf63,objname=file,parentOid=53ae036bfcfb46f5580b912222f3026835e3ef972c7e3e53
2ba4a5de30b1946e,parentZone=urn:storageos:VirtualDataCenterData:365f4482-c24a-4eca-
b24a-070efe29bf63,policyname=default,retention=CgYIoKOZmlE=,size=0,type=regular,uid=u1,pa
rent=apache,gid=apache
x-emc-useracl: u1=FULL_CONTROL
x-emc-groupacl: other=READ
x-emc-policy: default
x-emc-request-id: 0af7b3e4:159f0185cf7:957:4
Content-Type: plain/text
Content-Length: 0
Server: ViPR/1.0
```

Example: Update an object with retention values.

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

Response

HTTP/1.1 200 OK

Date: Thu, 16 Feb 2017 19:37:16 GMT
x-emc-policy: _int
x-emc-utf8: true
```

```
x-emc-request-id: 0af7b3e4:15a4849d95e:582:0
Content-Length: 0
Server: ViPR/1.0
```

Expiration period

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

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

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

```
POST /rest/namespace/file2 HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Tue, 31 Jan 2017 19:38:00 GMT
x-emc-expiration-period:300
x-emc-uid:a2b85977fd08488b80e646ea875e990b/u1
Content-Type:plain/text
x-emc-signature:krhYBfKSiM3mFOT6FtRB+2/xZnw=
Content-Length: 10240
Expect: 100-continue
```

Example: Request and response using the Atmos API:

```
POST /rest/namespace/file2 HTTP/1.1
User-Agent: curl/7.37.0
Host: 10.247.179.228:9022
Accept: */*
x-emc-date:Thu, 02 Feb 2017 02:47:32 GMT
x-emc-meta:user.maui.expirationEnable=true,user.maui.expirationEnd=2017-03-30T20:20:00Z
x-emc-uid:239e20dec7a54301a0b02f6090edcace/u1
Content-Type:plain/text
x-emc-signature:5tGEyK/9qUZCPSnQ9OPOdktN+Zo=
Content-Length: 10240
Expect: 100-continue

Response

HTTP/1.1 100 Continue
HTTP/1.1 201 Created
Date: Thu, 02 Feb 2017 02:47:33 GMT
x-emc-policy: default
x-emc-request-id: 0af7b3e4:159fb81ddae:345e:0
x-emc-delta: 10240
Location: /rest/objects/5c3abaf60e0e207abec96baf0618c0461b7cd716898f8a12ee236aed1ec94bea-
c86ee0e9-8709-4897-898e-c3d1895e1d93
x-emc-mtime: 1486003652813
Content-Length: 0
Server ViPR/1.0 is not blacklisted
Server: ViPR/1.0
```

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

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

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

Retention start delay window

Atmos enables you to specify a start delay window when creating a retention period, which enables you to migrate to ECS. Also, this feature prevents the objects from getting into retention after initial upload of an object.

Atmos creates subtenant request header, `x-emc-retention-start-delay` that captures the autocommit interval.

```
 ./atmoscurl.pl -user USER1 -action PUT -pmode TID -path / -header "x-emc-retention-
period:300" -header "x-emc-retention-start-delay:120" -include
```

Retention start delay applied on object mtime

In Atmos object creation, if retention start delay is set on the bucket (`x-emc-retention-start-delay`), the start delay for the object is calculated based on `time-since-mtime` of the object.

NOTE: The `time-since-mtime` is considered to calculate the start delay as it does not give an exact time to complete an upload and `x-emc-retention-start-delay` could be shorter even as a few minutes.

Override bucket-level retention for migrated objects

- If the user decides to migrate data through Atmos API to an ECS bucket in a compliant namespace with maui retention headers and if there are any conflicting retentions, the longest retention wins.
- On noncompliant buckets, for Atmos migrated objects, the `user.maui*headers` specifies the final retention value on an object. If there are no `user.maui*headers` available, the longest retention wins.
- On object creation in ECS through Atmos API, the `user.maui*headers` cannot be combined with any of `x-emc-retention` headers.

Atmos API supports GeoDrive

Atmos API supports GeoDrive on ECS. GeoDrive is a windows application that enables Atmos data to be mirrored to the local Windows file system, and it is the same as CIFS-ECS.

ECS Atmos error codes

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

Table 41. Error Codes

Error Code	Error Message	HTTP Status Code	HTTP Status Description
1001	The server encountered an internal error. Please try again.	500	Internal Server Error
1002	One or more arguments in the request were invalid.	400	Bad Request
1003	The requested object was not found.	404	Not Found

Table 41. Error Codes (continued)

1004	The specified range cannot be satisfied.	416	Requested Range Not Satisfiable
1005	One or more metadata tags were not found for the requested object.	400	Bad Request
1006	Operation aborted because of a conflicting operation in process against the resource. **NOTE:** This error code may indicate that the system is temporarily too busy to process the request. This is a non-fatal error; you can re-try the request later.	409	Conflict
1007	The server encountered an internal error. Please try again.	500	Internal Server Error
1008	The requested resource was not found on the server.	400	Bad Request
1009	The method specified in the Request is not allowed for the resource identified.	405	Method Not Allowed
1010	The requested object size exceeds the maximum allowed upload/download size.	400	Bad Request
1011	The specified object length does not match the actual length of the attached object.	400	Bad Request
1012	There was a mismatch between the attached object size and the specified extent size.	400	Bad Request
1013	The server encountered an internal error. Please try again.	500	Internal Server Error
1014	The maximum allowed metadata entries per object has been exceeded.	400	Bad Request
1015	The request could not be finished due to insufficient access privileges.	401	Unauthorized
1016	The resource you are trying to create already exists.	400	Bad Request
1019	The server encountered an I/O error. Please try again.	500	Internal Server Error
1020	The requested resource is missing or could not be found.	500	Internal Server Error
1021	The requested resource is not a directory.	400	Bad Request
1022	The requested resource is a directory.	400	Bad Request
1023	The directory you are attempting to delete is not empty.	400	Bad Request
1024	The server encountered an internal error. Please try again.	500	Internal Server Error
1025	The server encountered an internal error. Please try again.	500	Internal Server Error
1026	The server encountered an internal error. Please try again.	500	Internal Server Error
1027	The server encountered an internal error. Please try again.	500	Internal Server Error
1028	The server encountered an internal error. Please try again.	500	Internal Server Error

Table 41. Error Codes (continued)

1029	The server encountered an internal error. Please try again.	500	Internal Server Error
1031	The request timestamp was outside the valid time window.	403	Forbidden
1032	There was a mismatch between the signature in the request and the signature as computed by the server.	403	Forbidden
1033	Unable to retrieve the secret key for the specified user.	403	Forbidden
1034	Unable to read the contents of the HTTP body.	400	Bad Request
1037	The specified token is invalid.	400	Bad Request
1040	The server is busy. Please try again	500	Internal Server Error
1041	The requested filename length exceeds the maximum length allowed.	400	Bad Request
1042	The requested operation is not supported.	400	Bad Request
1043	The object has the maximum number of links	400	Bad Request
1044	The specified parent does not exist.	400	Bad Request
1045	The specified parent is not a directory.	400	Bad Request
1046	The specified object is not in the namespace.	400	Bad Request
1047	Source and target are the same file.	400	Bad Request
1048	The target directory is not empty and may not be overwritten	400	Bad Request
1049	The checksum sent with the request did not match the checksum as computed by the server	400	Bad Request
1050	The requested checksum algorithm is different than the one previously used for this object.	400	Bad Request
1051	Checksum verification may only be used with append update requests	400	Bad Request
1052	The specified checksum algorithm is not implemented.	400	Bad Request
1053	Checksum cannot be computed for an object on update for which one wasn't computed at create time.	400	Bad Request
1054	The checksum input parameter was missing from the request.	400	Bad Request
1056	The requested operation is not supported for symlinks.	400	Bad Request
1057	If-Match precondition failed.	412	Precondition failed
1058	If-None-Match precondition failed.	412	Precondition failed
1059	The key you are trying to create already exists.	400	Bad Request
1060	The requested key was not found.	404	Not found
1061	The requested pool already exists.	400	Bad Request
1062	The requested pool was not found.	404	Not found
1063	The maximum number of pools has been reached.	400	Bad request
1064	The request could not be completed because the subtenant is over quota	403	Forbidden

Table 41. Error Codes (continued)

1065	The request could not be completed because the UID is over quota	403	Forbidden
1070	Did not receive the expected amount of data.	400	Bad request
1071	Client closed connection before reading all data.	499	Client Closed Request
1072	Could not write all bytes to the client.	499	Client Closed Request
1073	Timeout writing data to the client.	499	Client Closed Request

6

CAS

This section describes the support that ECS provides for CAS.

Topics:

Setting up CAS support in ECS

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

ECS CAS enables 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.

For your storage pools, you might consider setting up a cold archive. See Cold Storage.

Next, set up your namespaces, users, and buckets using the standard documentation. See the *ECS Administration Guide* which is available from the https://www.dell.com/support/ for these steps as well as provisioning steps.

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 two 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 42. 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

Table 42. Requirements for regular and cold archives compared (continued)

Use case	How enabled	Minimum required nodes	Minimum required disks	Recommended disks	EC efficiency	EC scheme
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 storage pool. Once a storage pool has been created, this setting cannot be changed.

Compliance

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

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

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

Compliance has three components:

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

Platform hardening and Compliance

The following ECS security features support Compliance standards.

ECS platform security features:

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

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

Compliance and retention policy

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

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

- You cannot turn Compliance off once it is turned on.
- All buckets in a namespace must have a retention period greater than zero.

  NOTE: If you have an application that assigns object-level retention periods, do not use ECS to assign a retention period greater than the application retention period. This action causes application errors.
- A bucket with data in it cannot be deleted regardless of its retention value.
- Applying the **Infinite** option to a bucket means that objects in the bucket in a Compliance-enabled namespace cannot be deleted permanently.
- The retention period for an object cannot be deleted or shortened. Therefore, the retention period for a bucket cannot be deleted or shortened.
- You can increase object and bucket retention periods.
- No user can delete an object under retention. This includes users with the CAS privileged-delete permission.

Compliance agent

Describes the operation of the Compliance agent.

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

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

CAS retention in ECS

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

Retention periods

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

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

Retention policies (retention classes)

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

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

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

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

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

ECS bucket-level retention and CAS

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

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

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

CAS precedence

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

How to apply CAS retention

You can define retention 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.

Table 43. ECS Management API resources for retention

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

You can find more information about the ECS Management API in ECS Management REST API introduction. The online reference is here: ECS API Reference.

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

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

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

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

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

Min/max governor for CAS bucket-level retention

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

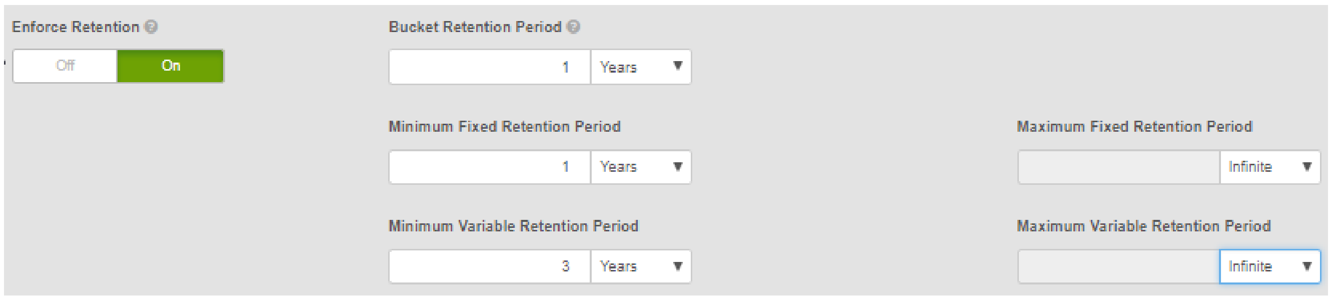

Enforce Retention: Off / On (On selected)
Bucket Retention Period: 1 Years
Minimum Fixed Retention Period: 1 Years
Maximum Fixed Retention Period: Infinite
Minimum Variable Retention Period: 3 Years
Maximum Variable Retention Period: Infinite

Figure 3. Retention options for CAS buckets

Table 44. CAS Bucket

Feature	Description
Enforce Retention	If this feature is turned on, no CAS object can be created without retention information (period or policy). An attempt to save such an object will return an error. If it is turned on, it is possible not to configure **Bucket Retention Period** even in a compliance-enabled environment. **NOTE:** When a CE+ mode Centera is migrated to ECS, **Enforce Retention** is turned on 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	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. 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.
Maximum Fixed Retention Period	

Table 44. CAS Bucket (continued)

Feature	Description
Minimum Variable Retention Period Maximum 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.

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

Event-based retention

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

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

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

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

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

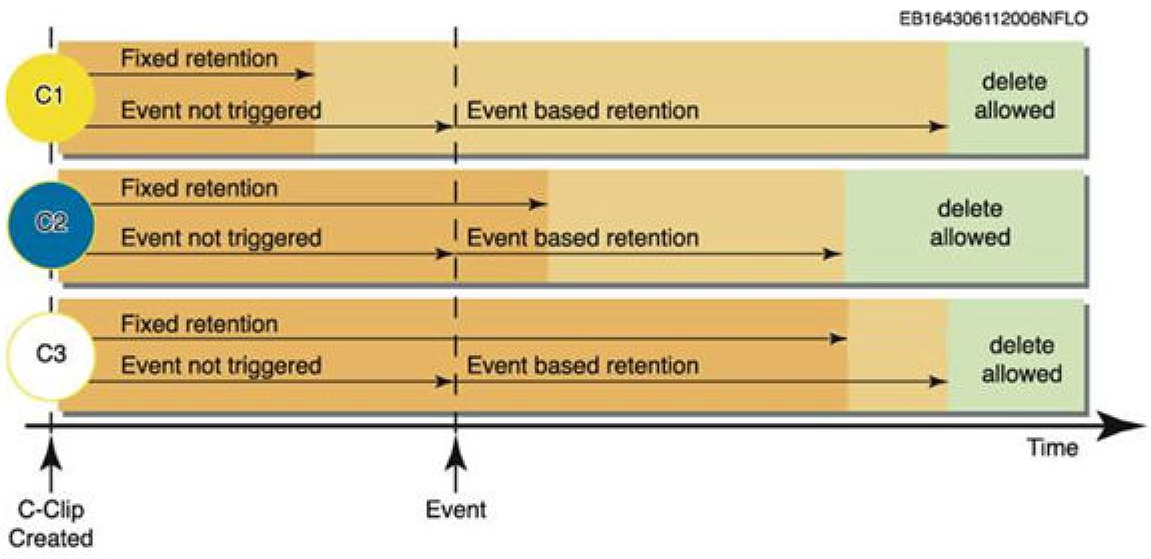


Figure 4. EBR scenarios

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 45. CAS API functions for event-based retention

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

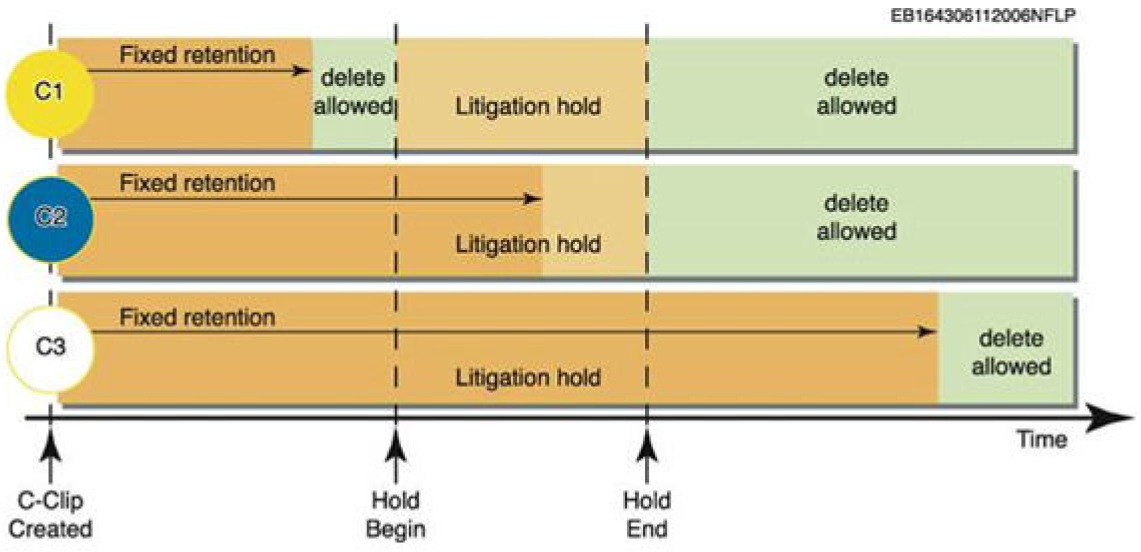


Figure 5. Litigation hold scenarios

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 46. 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.

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.
4. Select **Save**.

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 that the two users and namespace have been set up.

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:

   Table 47. Replication Details

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

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

Set up a CAS object user

Set up an object user to use CAS.

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

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.
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 needs 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 uses 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.

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.
3. Choose the **Edit** associated with the user.
4. Modify the permissions.

Table 48. 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 49. 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.

Table 49. Bucket ACL groups

Bucket ACL group	Description
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:

- `<ip address>?name=<name>,password=<password>` : Authenticates you with the CAS API as an alternative to PEA file.

  NOTE: You need the name and password from the PEA file.
- `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.
- `PUT /object/user-cas/ip-restrictions/{namespace_name}/{user_name}`: Sets IP restrictions for the CAS user.
- `GET /object/user-cas/ip-restrictions/{namespace_name}/{user_name}` : Gets IP restrictions that is set for the CAS user.
- `GET /object/user-cas/ip-restrictions/`: Lists IP restrictions that is set for all the CAS users.

  NOTE: You can restrict client IPs from accessing a CAS bucket. For more information about how to restrict client IPs from accessing a CAS bucket, see *ECS Administration Guide* which is available from https://www.dell.com/support/.

See the ECS API Reference for more information.

Content Addressable Storage (CAS) SDK API support

This section describes about the CAS query support and the CAS SDK APIs that are not supported prior to ECS 3.0 versions.

Supported versions

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

More information about ECS CAS support is provided in Setting up CAS support in ECS.

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

CAS connection string

When the client or the application tries to connect to ECS using the CAS protocol, it connects with the first node. If the first node is not reachable, then the application tries to connect with the other IPs/nodes by traversing left to right in the connection string. The application stops trying when all the nodes in the connection string are down or not reachable.

REST API Call to get IPs - `/object/vdcs/vdc/local`

Use the `svc tool` command to call the API `svc_rest_cmd /object/vdcs/vdc/local`

NOTE: ECS recommends mentioning the IPs of all the nodes in the connection string for an uninterrupted connection. For more information, see KB 000537533.

The `FPPoolOpen` API considers a connection string as an argument. Some examples:

- `10.0.0.1,10.0.0.2?mypeafile.pea`
- `10.0.0.1,10.0.0.2?name=profilename,secret=secret`
- `10.0.0.1,10.0.0.2,secondary=10.1.0.10?name=profilename,secret=secret`

NOTE:

- The connection strings can be of several forms and can be more than a collection of IP addresses. See the SDK Programmers Guide for a complete list.
- In the connection string , you can use alternate VDC IPs such as ECS1_IP1, ECS2_IP1, ECS1_IP2, ECS2_IP2.
- The client or the application user interface should allow the admin to enter a free format string. The free format string allows the user to configure a wide array of connection string possibilities.
- If local object metadata reads are enabled for a CAS bucket, then ECS tries to read local objects first if they were replicated. If the local objects were not replicated, then ECS requests the object metadata from the remote site. No changes are needed in the connection string for buckets with local object metadata reads enabled to compare to other ECS CAS buckets.
- If local object metadata reads are enabled for a CAS bucket, then ECS tries to read object locally first if replicated and, if not, then requests object metadata from a remote site, that is no changes are needed in the connection string for local object metadata reads enabled buckets compare to other ECS CAS buckets.

ECS CAS error codes

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

Table 50. Error Cdes

Value	Error name	Description
10001	FP_INVALID_NAME	The name that you have used is not XML compliant.
10002	FP_UNKNOWN_OPTION	You have used an unknown option name with FPPool_SetIntOption() or FPPool_GetIntOption().
10003	FP_NOT_SEND_REQUEST_ERR	An error occurred when you sent a request to the server. This internal error was generated because the server could not accept the request packet. Verify all LAN connections and try again.
10004	FP_NOT_RECEIVE_REPLY_ERR	No reply was received from the server. This internal error was generated because the server did not send a reply to the request packet. Verify all LAN connections and try again.
10005	FP_SERVER_ERR	The server reports an error. An internal error on the server occurred. Try again.
10006	FP_PARAM_ERR	You have used an incorrect or unknown parameter. Example: Is a string-variable too long, null, or empty when it should not be? Does a parameter have a limited set of values? Check each parameter in your code.
10007	FP_PATH_NOT_FOUND_ERR	This path does not correspond to a file or directory on the client system. The path in one of your parameters does not point to an existing file or directory. Verify the path in your code.
10008	FP_CONTROLFIELD_ERR	The server reports that the operation generated a "Controlfield missing" error. This internal error was generated because the required control field was not found. Try again. (Obsolete fromv2.0.)
10009	FP_SEGDATA_ERR	The server reports that the operation generated a "Segdatafield missing" error. This internal error was generated because the required field containing the blob data was not found in the packet. Try again. (Obsolete fromv2.0.)
10010	FP_DUPLICATE_FILE_ERR	A duplicate CA already exists on the server. If you did not enable duplicate file detection, verify that you have not already stored this data and try again.
10011	FP_OFFSET_FIELD_ERR	The server reports that the operation generated an "Offsetfield missing" error. This internal error was generated because the offset field was not found in the packet. Try again. (Obsolete fromv2.0.)
10012	FP_OPERATION_NOT_SUPPORTED	This operation is not supported. If FPClip_Write(), FPTag_GetSibling(), FPTag_GetPrevSibling(), FPTag_GetFirstChild() or FPTag_Delete() returned this error, then this operation is not supported for C-Clips opened in 'flat' mode. If FPStream returned this error, then you are trying to perform an operation that is not supported by that stream.
10013	FP_ACK_NOT_RCV_ERR	A write acknowledgement was not received. Verify your LAN connections and try again.
10014	FP_FILE_NOT_STORED_ERR	Could not write the blob to the server ORcould not find the blob on the server. This internal error was generated

Table 50. Error Cdes (continued)

		because the store operation of the blob was not successful. Verify that the original data was correctly stored, verify your LAN connections and try again.
10015	FP_NUMLOC_FIELD_ERR	The server reports that the operation generated a "Numlockfield missing" error. This internal error was generated because the numlock field was not found in the packet. Try again. (Obsolete fromv2.0.)
10016	FP_SECTION_NOT_FOUND_ERR	The GetSection request could not retrieve the defined section tag. This internal error was generated because a required section is missing in the CDF. Verify the content of your code and try again. (Obsolete fromv2.0.)
10017	FP_TAG_NOT_FOUND_ERR	The referenced tag could not be found in the CDF. This internal error was generated because information is missing from the description section in the CDF. Verify the content of your code and try again.
10018	FP_ATTR_NOT_FOUND_ERR	Could not find an attribute with that name. If FPTag_GetXXXAttribute() returned this error, then the attribute was not found in the tag. If FPTag_GetIndexAttribute() returned this error, then the index parameter is larger than the number of attributes in the tag.
10019	FP_WRONG_REFERENCE_ERR	The reference that you have used is invalid. The reference was not opened, already closed, or not of the correct type.
10020	FP_NO_POOL_ERR	It was not possible to establish a connection with a cluster. The server could not be located. This means that none of the IP addresses could be used to open a connection to the server or that no cluster could be found that has the required capability. Verify your LAN connections, server settings, and try again.
10021	FP_CLIP_NOT_FOUND_ERR	Could not find the referenced C-Clip in the cluster. Returned by FPClip_Open(), it means the CDF could not be found on the server. Verify that the original data was correctly stored and try again.
10022	FP_TAGTREE_ERR	An error exists in the tag tree. Verify the content of your code and try again.
10023	FP_ISNOT_DIRECTORY_ERR	A path to a file has been given but a path to a directory is expected. Verify the path to the data and try again.
10024	FP_UNEXPECTEDTAG_ERR	Either a "file" or "folder" tag was expected but not given. An unexpected tag was found when retrieving the CDF. The CDF is probably corrupt.
10025	FP_TAG_READONLY_ERR	The tag cannot be changed or deleted (it is probably a top tag). Verify your program logic.
10026	FP_OUT_OF_BOUNDS_ERR	The options parameter is out of bounds. One of the function parameters exceeds its preset limits. Verify each parameter in your code.
10027	FP_FILESYS_ERR	A file system error occurred, for example an incorrect path was given, or you are trying to open an unknown file or a file in the wrong mode. Verify the path and try again.
10029	FP_STACK_DEPTH_ERR	You have exceeded the nested tag limit. Review the structure of your content description and try again. Deprecated.
10030	FP_TAG_HAS_NO_DATA_ERR	You are trying to access blob data of a tag that does not contain blob data.

Table 50. Error Cdes (continued)

10031	FP_VERSION_ERR	The C-Clip has been created using a more recent version of the client software than you are using. Upgrade to the latest version.
10032	FP_MULTI_BLOB_ERR	The tag already has data associated with it. You need to create a new tag to store the new data or delete this tag and recreate it and try again.
10033	FP_PROTOCOL_ERR	You have used an unknown protocol option (Only HPP is supported). Verify the parameters in your code. It is also possible that an internal communication error occurred between the server and client. If you have verified your code and the problem persists then you need to upgrade to the latest client and server versions.
10034	FP_NO_SOCKET_AVAIL_ERR	No new network socket is available for the transaction. Reduce the number of open transactions between the client and the server or use the function FPPool_SetGlobalOption() to increase the number of available sockets with FP_OPTION_MAXCONNECTIONS.
10035	FP_BLOBIDFIELD_ERR	A BlobID field (the Content Address) was expected but not given. Upgrade to the latest client and server versions. (Obsolete fromv2.0.)
10036	FP_BLOBIDMISMATCH_ERR	The blob is corrupt: a BlobID mismatch occurred between the client and server. The Content Address calculation on the client and the server has returned different results. The blob is corrupt. If FPClip_Open() returns this error, it means the blob data or metadata of the C-Clip is corrupt and cannot be decoded.
10037	FP_PROBEPACKET_ERR	The probe packet does not contain valid server addresses. Upgrade to the latest client and server versions. (Obsolete fromv2.0.)
10038	FP_CLIPCLOSED_ERR	(Javaonly.) You tried to perform an operation on a closed C-Clip. This operation requires access to an open C-Clip. Verify your code and try again.
10039	FP_POOLCLOSED_ERR	(Javaonly.) You tried to perform an operation on a closed pool. This operation requires access to an open pool. Verify your code and LAN connections and try again.
10040	FP_BLOBBUSY_ERR	The blob on the cluster is busy and cannot be read from or written to. You tried to read from or write to a blob that is currently busy with another read/write operation. Try again.
10041	FP_SERVER_NOTREADY_ERR	The server is not ready yet. This error can occur when a client tries to connect to the server to execute an operation and the nodes with the access role are running but the nodes with the storage role have not been initialized yet. This error can also occur when not enough mirror groups are found on the server. Allow the SDK to perform the automatic number of configured retries.
10042	FP_SERVER_NO_CAPACITY_ERR	The server has no capacity to store data. Enlarge the server's capacity and try again.
10043	FP_DUPLICATE_ID_ERR	The application passed in a sequence ID that was previously used.
10044	FP_STREAM_VALIDATION_ERR	A generic stream validation error occurred.
10045	FP_STREAM_BYTECOUNT_MISMATCH_ERR	A generic stream byte count mismatch was detected.

Table 50. Error Cdes (continued)

10101	FP_SOCKET_ERR	An error on the network socket occurred. Verify the network.
10102	FP_PACKETDATA_ERR	The data packet contains wrong data. Verify the network, the version of the server or try again later.
10103	FP_ACCESSNODE_ERR	No node with the access role can be found. Verify the IP addresses provided with FPPool_Open().
10151	FP_OPCODE_FIELD_ERR	The Query Opcode field is missing from the packet.
10152	FP_PACKET_FIELD_MISSING_ERR	The packet field is missing.
10153	FP_AUTHENTICATION_FAILED_ERR	Authentication to get access to the server failed. Check the profile name and secret.
10154	FP_UNKNOWN_AUTH_SCHEME_ERR	An unknown authentication scheme has been used.
10155	FP_UNKNOWN_AUTH_PROTOCOL_ERR	An unknown authentication protocol has been used.
10156	FP_TRANSACTION_FAILED_ERR	Transaction on the server failed.
10157	FP_PROFILECLIPID_NOTFOUND_ERR	No profile clip was found.
10158	FP_ADVANCED_RETENTION_DISABLED_ ERR	The Advanced Retention Management feature is not licensed or enabled for event-based retention (EBR) and retention hold.
10159	FP_NON_EBR_CLIP_ERR	An attempt was made to trigger an EBRevent on a C-Clip that is not eligible to receive an event.
10160	FP_EBR_OVERRIDE_ERR	An attempt was made to trigger or enable the event-based retention period/class of a C-Clip a second time. You can set EBR information only once.
10161	FP_NO_EBR_EVENT_ERR	The C-Clip is under event-based retention protection and cannot be deleted.
10162	FP_RETENTION_OUT_OF_BOUNDS_ERR	The event-based retention period being set does not meet the minimum/maximum rule.
10163	FP_RETENTION_HOLD_COUNT_ERR	The number of retention holds exceeds the limit of 100.
10164	FP_METADATA_MISMATCH_ERR	Mutable metadata mismatch found.
10201	FP_OPERATION_REQUIRES_MARK	The application requires marker support but the stream does not provide that.
10202	FP_QUERYCLOSED_ERR	The FP Query for this object is already closed. (Java only).
10203	FP_WRONG_STREAM_ERR	The function expects an input stream and gets an output stream or vice-versa.
10204	FP_OPERATION_NOT_ALLOWED	The use of this operation is restricted or this operation is not allowed because the server capability is false.
10205	FP_SDK_INTERNAL_ERR	An SDK internal programming error has been detected.
10206	FP_OUT_OF_MEMORY_ERR	The system ran out of memory. Check the system's capacity.
10207	FP_OBJECTINUSE_ERR	Cannot close the object because it is in use. Check your code.
10208	FP_NOTYET_OPEN_ERR	The object is not yet opened. Check your code.
10209	FP_STREAM_ERR	An error occurred in the generic stream. Check your code.
10210	FP_TAG_CLOSED_ERR	The FP Tag for this object is already closed. (Java only.)
10211	FP_THREAD_ERR	An error occurred while creating a background thread.

Table 50. Error Cdes (continued)

10212	FP_PROBE_TIME_EXPIRED_ERR	The probe limit time was reached.
10213	FP_PROFILECLIPID_WRITE_ERR	There was an error while storing the profile clip ID.
10214	FP_INVALID_XML_ERR	The specified string is not valid XML.
10215	FP_UNABLE_TO_GET_LAST_ERROR	The call to FPPool_GetLastError() or FPPool_GetLastErrorInfo() failed. The error status of the previous function call is unknown; the previous call may have succeeded.
10216	FP_LOGGING_CALLBACK_ERR	An error occurred in the application-defined FP Logging callback.

Enabling data2 IP in CAS

Data2 IP allows CAS to start on multiple IPs. To enable data2 on CAS, contact ECS remote support.

NOTE: Data2 IP is enabled by default in CAS from ECS 3.1 and later versions.

7

ECS Management REST API

This section describes information about accessing and authenticating with the ECS Management REST API and provides a summary of the API paths.

Topics:

ECS Management REST API introduction

You can configure and manage the object store using the ECS REST Management API. Once the object store is configured, you can perform object create, read, update, and delete operations using the ECS-supported object and file protocols.

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

- Authenticate with the ECS Management REST API
- REST API for Object Control summary

In addition, you can see the ECS REST API REFERENCE Guide which is autogenerated from the source code and provides a reference for the methods available in the API.

Authenticate with the ECS Management REST API

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

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

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

You can retrieve and use authentication tokens by:

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

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

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

Authenticate without cookies

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

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

The default ECS API token lifetime is 8 hours, which means that after 8 hours the token is no longer valid. The default idle time for a token is two hours; after a two hour idle time, the token expires. If you use an expired token, you are redirected to the `/login` URL. You will receive an HTTP status error code 401 upon any subsequent use of the expired token.

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

> GET /login HTTP/1.1
> Authorization: Basic cm9vdDpDaGFuZ2VNZQ==
> User-Agent: curl/7.24.0 (i386-pc-win32) libcurl/7.24.0 OpenSSL/0.9.8t zlib/1.2.5
> Host: 10.247.100.247:4443
> Accept: */*
>
< HTTP/1.1 200 OK
< Date: Tue, 26 Nov 2013 22:18:25 GMT
< Content-Type: application/xml
< Content-Length: 93
< Connection: keep-alive
< X-SDS-AUTH-TOKEN:
BAAcQ0xOd3g0MjRCUG4zT3NJdnNuMlAvQTFYblNrPQMAUAQADTEzODU0OTQ4NzYzNTICAAEABQA5dXJu

OnN0b3JhZ2VvczpUb2tlbjo2MjIxOTcyZS01NGUyLTRmNWQtYWZjOC1kMGE3ZDJmZDU3MmU6AgAC0A8=
<
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<loggedIn>
   <user>root</user>
</loggedIn>
* Connection #0 to host 10.247.100.247 left intact
* Closing connection #0
* SSLv3, TLS alert, Client hello (1):
```

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

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

OnN0b3JhZ2VvczpUb2tlbjpkYzc3ODU3Mi04NWRmLTQ2YjMtYjgwZi05YTdlNDFkY2QwZDg6AgAC0A8="
```

```
<?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 and passing the cookie into a subsequent request.

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

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

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

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

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

Logout

The logout API ends a session.

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

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

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

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

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

```

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

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

The response should be `HTTP 200`.

ECS Management REST API whoami command

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

Request

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

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

Response

```
HTTP 200

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

```
HTTP 200

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

ECS Management REST API summary

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

Table 51. ECS Management REST API - methods summary

API Area	Description
Data Movement (Copy to Cloud)	
DM Policy	`POST /object/bucket/test-policy` to validate a DM policy. `POST /object/bucket/test-policy-edit` to validate a DM policy edit. `GET /object/bucket/{bucketName}/copypolicy` to return a DM policy for the specified bucket. `PUT /object/bucket/{bucketName}/copypolicy` to update a DM policy for the specified bucket. `DELETE /object/bucket/{bucketName}/copypolicy` to delete a DM policy for the specified bucket. `GET /object/bucket/copypolicy` to return a list of all DM policies for the specified namespace.
Configuration	
Certificate	`/object-cert` API to manage certificates. `/object-cert/keystore` API to specify and rotate the certificate chain used by ECS.
Configuration Properties	`/config/object/properties` API to set the user scope as `GLOBAL` or `NAMESPACE`. This must be set before the first user is created. The default is `GLOBAL`. In `GLOBAL` scope, users are global and are can be shared across namespaces. In this case, the default namespace associated with a user determines the namespace for object operations and there is no need to supply a namespace for an operation. In `NAMESPACE` scope, a user is associated with a namespace. In this case, there might be more than one user with the same name, each associated with a different namespace, and a namespace must be provided for every operation.
Licensing	`/license` API to add a license and retrieve license details.
Feature	`/feature/ServerSideEncryption` API to retrieve the details of the ServerSideEncryption feature.
Syslog	`/vdc/syslog/config` API to manage Syslog configuration and send alerts to the Syslog server for troubleshooting and debugging purposes.
SNMP	`/vdc/snmp/config` API to manage SNMP configuration and send alerts to SNMP server for troubleshooting and debugging purposes.
CAS	
CAS user profile	`/object/user-cas/secret` API to assign secret keys to CAS users and generate the Pool Entry Authorization (PEA) file. `/object/user-cas/bucket` API to retrieve or update the default bucket of a specified CAS user. `/object/user-cas/applications/{namespace}` API to retrieve the CAS registered applications for a specified namespace. `/object/user-cas/metadata/{namespace}/{uid}` API to retrieve or update the CAS user metadata for the specified namespace and CAS user.

Table 51. ECS Management REST API - methods summary (continued)

API Area	Description
File system access	
NFS	`/object/nfs` API to create an NFS export based on an ECS bucket and enable access to the export by UNIX users and groups. `/object/nfs/users` API to manage mapping between ECS user/group and corresponding UNIX user ID. `/object/nfs/exports` API to create and manage NFS exports. For the best practices to apply when you mount ECS NFS exports, see KB 532228.
Metering	
Billing	`/object/billing` API to meter object store usage at the namespace and bucket level.
Migration	
Transformation	`/object/transformation` API to enable data transformation from a Centera cluster.
Monitoring	
Capacity	`/object/capacity` API to retrieve the current managed capacity.
Dashboard	`/dashboard/zones/localzone` API to retrieve the local VDC details, including details on replication groups, storage pools, nodes, and disks. `/dashboard/zones/hostedzone` API to retrieve the hosted VDC details, including details on replication groups. `/dashboard/replicationgroups/{id}` API to retrieve the replication group instance details. `/dashboard/storagepools/{id}` API to retrieve the storage pool details, including details on the storage pool nodes. `/dashboard/nodes/{id}` API to retrieve the node instance details, including node instance disk and process details. `/dashboard/disks/{id}` API to retrieve the disk instance details. `/dashboard/processes/{id}` API to retrieve the process instance details. `/dashboard/rglinks/{id}` API to retrieve the replication group link instance details. `/dashboard/datatables/{id}` API to retrieve the replication group datatables instance details.
Events	`/vdc/events` API to retrieve audit events for a specified namespace.
Alerts	`/vdc/alerts` API to retrieve audit alerts.
Multi-tenancy	
Namespace	`/object/namespaces` API to create and manage a namespace. This API also sets the retention period and quota for the namespace. For more information about retention periods and quotas, see the *ECS Administration Guide* which is available from the https://www.dell.com/support/.
Geo-replication	
Replication Group	`/data/data-service/vpools` API to create and manage replication groups.
Temporary Failed Zone	`/tempfailedzone/` API to retrieve all temporary failed zones, or the temporary failed zones for a specified replication group.
Provisioning	
Base URL	`/object/baseurl` API to create a Base URL that allows existing applications to work with the ECS object store. For more information on Base URL, see the *ECS Administration Guide* which is available from the https://www.dell.com/support/.
Bucket	`/object/bucket` API to provision and manage buckets. `/object/bucket/{bucketName}/lock` API to lock bucket access. `/object/bucket/{bucketName}/tags` API to add tags to a specified bucket. `/object/bucket/{bucketName}/retention` API to set the retention period for a specified bucket. `/object/bucket/{bucketName}/quota` API to set the quota for a specified bucket. `/object/bucket/{bucketName}/policy` API to add a policy for a specified bucket. `/object/bucket/{bucketName}/metadata` API to add metadata for a specified bucket.

Table 51. ECS Management REST API - methods summary (continued)

API Area	Description
	`POST /object/bucket/<bucket>/deactivate?namespace=<namespace>&[emptyBucket=true|false]"` API to delete a bucket. The optional `emptyBucket=true` parameter will empty the bucket as part of the delete. `GET /object/bucket/<bucket>/empty-bucket- status?namespace=<namespace>` API to get bucket status.
Data store	`/vdc/data-stores` API to create datastores on file systems (`/vdc/data-stores/filesystems`) or on commodity nodes (`/vdc/data-stores/commodity`).
Node	`/vdc/nodes` API to retrieve the nodes that are currently configured for the cluster. `/vdc/nodes/{nodename}/lockdown` API to set the locked or unlocked status for a specified node. `/vdc/lockdown` API to retrieve the locked or unlocked status for a VDC.
Storage pool	`/vdc/data-services/varrays` API to create and manage storage pools.
Virtual data center	`/object/vdcs` API to add a VDC and specify the inter-VDC endpoints and secret key for replication of data between ECS sites.
VDC keystore	`/vdc/keystore` API to manage certificates for a VDC.
Support	
Call home	`/vdc/callhome/` API for managing ESRS configuration and sending alerts to ConnectEMC for troubleshooting and debugging purposes.
Task Coordinator	
Task	`GET /object/task?task_id=task_abc` to get single task. `/object/task?[limit=][&marker=]` to list all tasks. `/object/task?[task_status=][task_type=][bucket_id=]` to list tasks with filters. `/object/task/$task_id/priority/3` to update task priority. `DELETE /object/task/$task_id/abort` to delete task.
User Management	
Authentication provider	`/vdc/admin/authnproviders` API to add and manage authentication providers.
Password group (Swift)	`/object/user-password` API to generate a password for use with OpenStack Swift authentication.
Secret key	`/object/user-secret-keys` API to assign secret keys to object users and to manage secret keys.
Secret key self-service	`/object/secret-keys` API to enable 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 to create and manage object users. Object users are always associated with a namespace. The API returns a secret key that can be used for S3 access. An object user assigned an S3 secret key can change it using the REST API. `/object/users/lock`. API to lock user access. `/object/users/{userName}/tags`. API to associate tags with a user ID. Tags are in the form of name=value pairs.
User (management)	`/vdc/users` API to create and manage users. Management users can be assigned to the System Administrator role or to the Namespace Administrator role. You can use this API the change the local management user password.

A

Topics:

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 52. 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.[federation].hosts property to uniquely identify sets of ECS data nodes. The names are used as a component of the authority section of the ECS HDFS file system URI. For example: `<property>` `<name>fs.vipr.installations</name>` `<value><federation>,<site1>,<testsite></value>` `</property>`
fs.vipr.installation.[federation].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 or FQDNs. For example: `<property>` `<name>fs.vipr.installation.<federation>.hosts</name>` `<value>203.0.113.10,203.0.113.11,203.0.113.12</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.<federation>.resolution</name>` `<value>dynamic</value>` `</property>`

Table 52. Hadoop core-site.xml properties (continued)

Property	Description

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.<federation>.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].[federation]`
	
	● *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.
	● *federation*: 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.federation1</value>`
	` </property>`

`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`

Table 52. Hadoop core-site.xml properties (continued)

Property	Description

	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 users and groups are assigned to newly created files.
	NOTE: This property was 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. 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 Hadoop cluster that uses Kerberos. 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

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

core-site.xml

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

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

```
</property>

<property>
  <name>fs.vipr.installations</name>
  <value>federation1</value>
</property>

<property>
  <name>fs.vipr.installation.federation1.hosts</name>
  <value>203.0.113.10,203.0.113.11,203.0.113.12</value>
</property>

<property>
  <name>fs.vipr.installation.federation1.resolution</name>
  <value>dynamic</value>
</property>

<property>
  <name>fs.vipr.installation.federation1.resolution.dynamic.time_to_live_ms</name>
  <value>900000</value>
</property>

<property>
  <name>fs.defaultFS</name>
  <value>viprfs://mybucket.mynamespace.federation1/</value>
</property>

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

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

B

Topics:

External key management

This section provides information about EKM keys, EKM key hierarchy, and storage locations of EKM.

The table lists the supported Key-Trust-Platform (KTP) for different versions of ECS:

Table 53. Supported Key-Trust-Platform (KTP)

Supported KTP	ECS 3.5	ECS 3.6	ECS 3.7	ECS 3.8
IBM Secure Key Lifecycle Manager 3.0	Yes	Yes	Yes	Yes
Gemalto (SafeNet) KeySecure	Yes	Yes	Yes	Yes
Safenet KeySecure 8.11 with Client Certificate Authentication only	No	Yes	Yes	Yes
Thales CipherTrust Manager 2.5.2	No	No	No	Yes

NOTE: If you are using KeySecure, see Migrate KeySecure to Thales CipherTrust Manager for better support for ECS 3.8.

External key management uses a hierarchy. In the externally managed scenario, the Master Key is delegated to the External Key Manager (EKM). Like native key management, Master key is used to derive Virtual Master Key. Each namespace in ECS associates to a Namespace Key, and a Virtual Master Key protects it. Virtual Master Key is a key that is derived from master key and RT Data Encryption key, and the Virtual master key is never persisted to disks. All buckets within a namespace associates to a Bucket Key, and the corresponding namespace key protects the Bucket Key. Data for each object is encrypted using a unique object key, which is protected using a Virtual Bucket Key. Virtual bucket key is derived from bucket and rotation key and is not saved to disks.

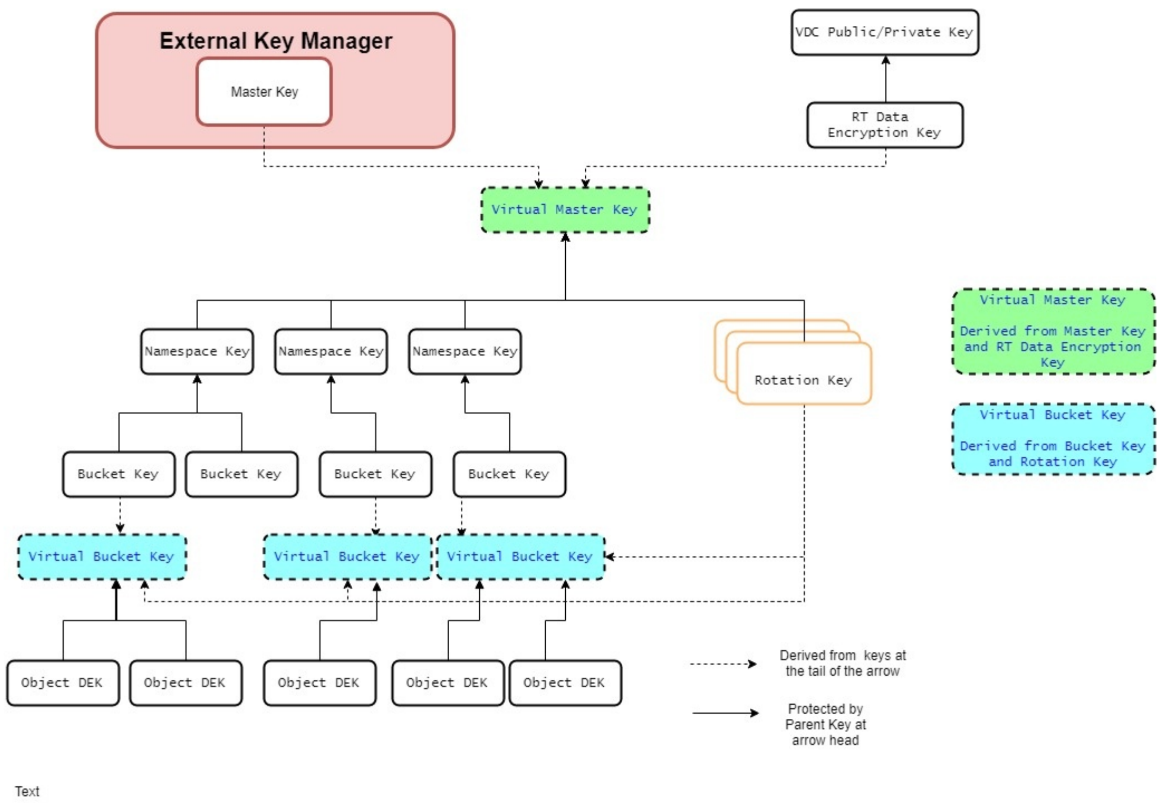


Figure 6. External Key Management Key Hierarchy

The master key is generated and stored securely in the External Key Manager (EKM). Namespace key and Rotation key are wrapped using `AESKeyWrapRFC5649` by virtual master key. Bucket key is wrapped using `AESKeyWrapRFC5649` by namespace key. These wrapped keys are stored in the Resource Table (RT) like the natively managed keys. The virtual bucket key wraps the object keys using `AESKeyWrapRFC5649` and stores in Object Table (OB).

The communication with EKMs is protected by SSL using server and client certificates. When ECS persists or retrieves the keys, only the encrypted data is transported across nodes of a VDC or across VDCs. Decryption of key happens locally at each service that requires a specific key.

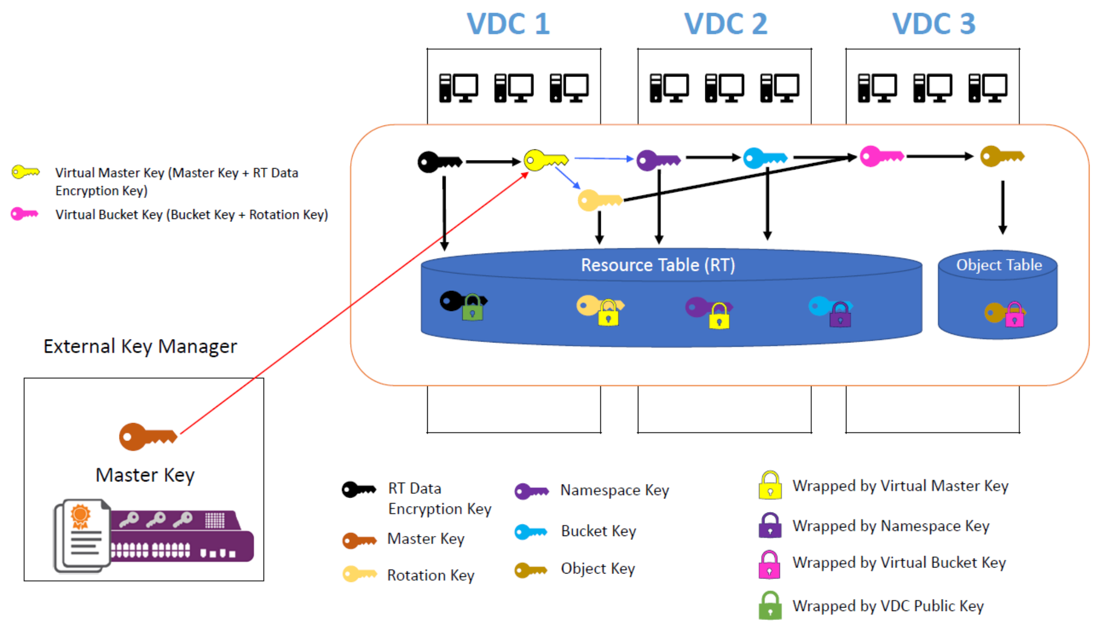

VDC 1
VDC 2
VDC 3
Virtual Master Key (Master Key + RT Data Encryption Key)
Virtual Bucket Key (Bucket Key + Rotation Key)
Resource Table (RT)
Object Table
External Key Manager
Master Key
RT Data Encryption Key
Master Key
Rotation Key
Namespace Key
Bucket Key
Object Key
Wrapped by Virtual Master Key
Wrapped by Namespace Key
Wrapped by Virtual Bucket Key
Wrapped by VDC Public Key

Figure 7. Storage Locations of EKM Managed keys

The table provides a brief explanation of the various keys that are used in key management using EKM.

Table 54. ECS EKM-Managed Keys

Key name	Key type	Protected by	Description	Storage
Master Key	KEK	External Key Manager	AES 256-bit key that is generated by EKM used with RT Data Encryption key to create Virtual Master key that protects rotation and namespace keys.	Unique per ECS Federation is created and stored in EKM. New Master key is generated every time a user requests a key rotation.
Rotation Key	KEK	Virtual Master Key using AESKeyWrapRFC5649	AES 256-bit key that is generated by EKM used to create Virtual Bucket key for wrapping object keys.	New rotation key that is generated and stored in ECS every time a user requests key rotation.
RT Data Encryption Key	KEK	VDC Public Key	Randomly generated AES 256-bit key used to create Virtual Master Key.	Unique per ECS Federation is stored wrapped using each VDC's Public Key in the Resource Table (RT). RT is a KV-store across all VDCs in the federation.
Virtual Master Key	KEK	Does not require protection, as it is not stored.	Is computed from the Master and RT Data Encryption keys when required.	This key is never persisted to disk.
Namespace Key	KEK	Virtual Master Key using `AESKeyWrapRFC5649`	Randomly generated AES 256-bit key per namespace, used to wrap all bucket keys belonging to buckets in the namespace.	Unique to each namespace stored as wrapped key using Virtual Master Key in the Resource Table.
Bucket Key	KEK	Namespace Key using `AESKeyWrapRFC5649`	Randomly generated AES 256-bit key per bucket, used along with Rotation Key to generate Virtual Bucket Key, used to wrap all object keys.	Unique to each bucket stored as wrapped key using namespace key in the Resource Table
Virtual Bucket Key	KEK	Does not require protection, as it is not stored.	Is computed from the Bucket and Rotation keys when required.	This key is never persisted to disk.
Object Data Encryption Key	DEK	Virtual Bucket Key using `AESKeyWrapRFC5649`	Randomly generated AES 256-bit key generated for each object, used to encrypt object data.	Object Key is wrapped using Virtual Bucket KEK in the Object Table (KV-store) in commodity disks.

User-supplied keys with the S3 API headers (ECS 3.2 and later)

With the S3 API, encryption keys can be specified in the header to encrypt objects. When an object is encrypted using user-supplied key, the key is never stored, only the hash of the key is stored in Object Table. The user must supply the encryption key every time an operation is performed on that object. ECS validates that the key provided for update, appends, and reads it as the same used during object creation.

Retrieving master key after Geo-federation

- A system being added to form or extend a federation generates public/private keys locally. These keys are used for encryption or decryption of the federation's master key.
- Upon federation, the new system that does not know the master key, stores the public key in Resource Table.
- A VDC that knows the master key uses this public key to encrypt and share the encrypted key with new system.
- The master key is now global and known to both systems within the federation.

From this point on, the master key is global and known to both systems within the federation. The ECS system that is labeled VDC 2 joins the federation. The master key of VDC 1 (the existing system) is extracted and passed to VDC 2 for encryption with the public-private key randomly generated by VDC 2.

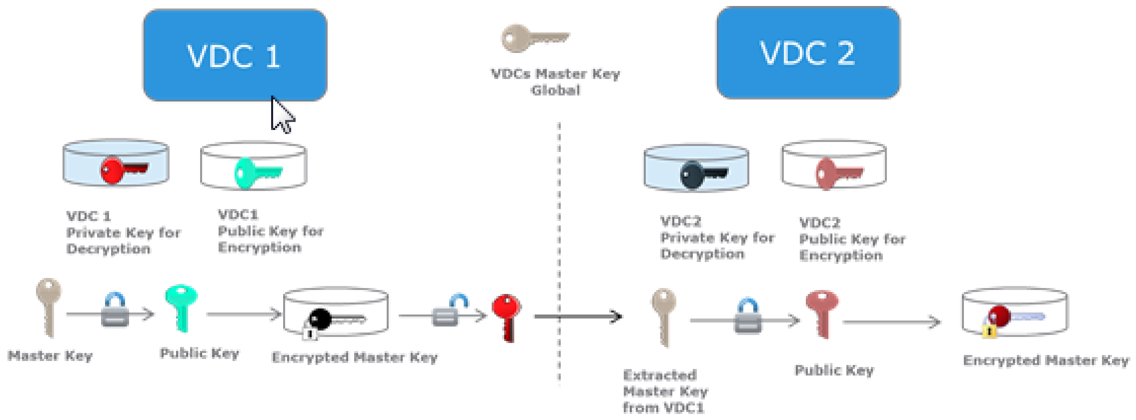

VDC 1
VDCs Master Key Global
VDC 2
VDC 1 Private Key for Decryption
VDC1 Public Key for Encryption
VDC2 Private Key for Decryption
VDC2 Public Key for Encryption
Master Key
Public Key
Encrypted Master Key
Extracted Master Key from VDC1
Public Key
Encrypted Master Key

Figure 8. Encryption of the master key in a geo-replicated environment

Migrate KeySecure to Thales CipherTrust Manager

If you are using KeySecure, perform the following steps to migrate to Thales CipherTrust Manager that ECS 3.8 supports.

1. Contact Customer Support to deactivate the existing KeySecure in ECS.
   When the existing KeySecure is deactivated ECS moves to Local or Native Key Management.
2. Delete the existing deactivated EKM servers and EKM cluster in ECS.
3. Create EKM cluster and select `Thales CipherTrust` as a **External Key Management Type**.
4. Add CTM servers.

   NOTE: You must add minimum of two CTM servers.

5. Create VDC EKM Mapping.
6. Activate EKM Cluster.

   NOTE: See *ECS 3.8 Administration Guide* for steps to create and activate new EKM cluster.

C

Topics:

Document feedback

If you have any feedback or suggestions regarding this document, mailto:ecs.docfeedback@dell.com.

Index