How StorageGRID implements S3 REST API

StorageGRID 11.7

NetApp
January 09, 2024

Table of Contents

How StorageGRID implements S3 REST API

Conflicting client requests

Conflicting client requests, such as two clients writing to the same key, are resolved on a "latest-wins" basis.

The timing for the "latest-wins" evaluation is based on when the StorageGRID system completes a given request, and not on when S3 clients begin an operation.

Consistency controls

Consistency controls provide a balance between the availability of the objects and the consistency of those objects across different Storage Nodes and sites, as required by your application.

By default, StorageGRID guarantees read-after-write consistency for newly created objects. Any GET following a successfully completed PUT will be able to read the newly written data. Overwrites of existing objects, metadata updates, and deletes are eventually consistent. Overwrites generally take seconds or minutes to propagate, but can take up to 15 days.

If you want to perform object operations at a different consistency level, you can specify a consistency control for each bucket or for each API operation.

Consistency controls

The consistency control affects how the metadata that StorageGRID uses to track objects is distributed between nodes, and therefore the availability of objects for client requests.

You can set the consistency control for a bucket or an API operation to one of the following values:

- **all**: All nodes receive the data immediately, or the request will fail.
- **strong-global**: Guarantees read-after-write consistency for all client requests across all sites.
- **strong-site**: Guarantees read-after-write consistency for all client requests within a site.
- **read-after-new-write**: (Default) Provides read-after-write consistency for new objects and eventual consistency for object updates. Offers high availability and data protection guarantees. Recommended for most cases.
- **available**: Provides eventual consistency for both new objects and object updates. For S3 buckets, use only as required (for example, for a bucket that contains log values that are rarely read, or for HEAD or GET operations on keys that don’t exist). Not supported for S3 FabricPool buckets.

Use “read-after-new-write” and “available” consistency controls

When a HEAD or GET operation uses the “read-after-new-write” consistency control, StorageGRID performs the lookup in multiple steps, as follows:

- It first looks up the object using a low consistency.
- If that lookup fails, it repeats the lookup at the next consistency level until it reaches a consistency level equivalent to the behavior for strong-global.

If a HEAD or GET operation uses the “read-after-new-write” consistency control but the object does not exist, the object lookup will always reach a consistency level equivalent to the behavior for strong-global. Because this consistency level requires multiple copies of the object metadata to be available at each site, you can receive a high number of 500 Internal Server errors if two or more Storage Nodes at the same site are unavailable.

Unless you require consistency guarantees similar to Amazon S3, you can prevent these errors for HEAD and GET operations by setting the consistency control to “available.” When a HEAD or GET operation uses the “available” consistency control, StorageGRID provides eventual consistency only. It does not retry a failed operation at increasing consistency levels, so it does not require that multiple copies of the object metadata be available.

Specify consistency control for API operation

To set the consistency control for an individual API operation, consistency controls must be supported for the operation, and you must specify the consistency control in the request header. This example sets the consistency control to “strong-site” for a GET Object operation.

```
GET /bucket/object HTTP/1.1
Date: date
Authorization: authorization name
Host: host
Consistency-Control: strong-site
```

> You must use the same consistency control for both the PUT Object and GET Object operations.

Specify consistency control for bucket

To set the consistency control for bucket, you can use the StorageGRID PUT Bucket consistency request and the GET Bucket consistency request. Or you can use the Tenant Manager or the Tenant Management API.

When setting the consistency controls for a bucket, be aware of the following:

- Setting the consistency control for a bucket determines which consistency control is used for S3 operations performed on the objects in the bucket or on the bucket configuration. It does not affect operations on the bucket itself.
- The consistency control for an individual API operation overrides the consistency control for the bucket.
- In general, buckets should use the default consistency control, “read-after-new-write.” If requests aren’t working correctly, change the application client behavior if possible. Or, configure the client to specify the consistency control for each API request. Set the consistency control at the bucket level only as a last resort.

How consistency controls and ILM rules interact to affect data protection

Both your choice of consistency control and your ILM rule affect how objects are protected. These settings can interact.

For example, the consistency control used when an object is stored affects the initial placement of object metadata, while the ingest behavior selected for the ILM rule affects the initial placement of object copies. Because StorageGRID requires access to both an object’s metadata and its data to fulfill client requests,

selecting matching levels of protection for the consistency level and ingest behavior can provide better initial data protection and more predictable system responses.

The following ingest behaviors are available for ILM rules:

- **Dual commit**: StorageGRID immediately makes interim copies of the object and returns success to the client. Copies specified in the ILM rule are made when possible.
- **Strict**: All copies specified in the ILM rule must be made before success is returned to the client.
- **Balanced**: StorageGRID attempts to make all copies specified in the ILM rule at ingest; if this is not possible, interim copies are made and success is returned to the client. The copies specified in the ILM rule are made when possible.

> Before selecting the ingest behavior for an ILM rule, read the full description of these settings in the instructions for managing objects with information lifecycle management.

Example of how the consistency control and ILM rule can interact

Suppose you have a two-site grid with the following ILM rule and the following consistency level setting:

- **ILM rule**: Create two object copies, one at the local site and one at a remote site. The Strict ingest behavior is selected.
- **Consistency level**: “strong-global” (Object metadata is immediately distributed to all sites.)

When a client stores an object to the grid, StorageGRID makes both object copies and distributes metadata to both sites before returning success to the client.

The object is fully protected against loss at the time of the ingest successful message. For example, if the local site is lost shortly after ingest, copies of both the object data and the object metadata still exist at the remote site. The object is fully retrievable.

If you instead used the same ILM rule and the “strong-site” consistency level, the client might receive a success message after object data is replicated to the remote site but before object metadata is distributed there. In this case, the level of protection of object metadata does not match the level of protection for object data. If the local site is lost shortly after ingest, object metadata is lost. The object can’t be retrieved.

The inter-relationship between consistency levels and ILM rules can be complex. Contact NetApp if you require assistance.

Related information

Manage objects with ILM

GET Bucket consistency

PUT Bucket consistency

How StorageGRID ILM rules manage objects

The grid administrator creates information lifecycle management (ILM) rules to manage object data ingested into the StorageGRID system from S3 REST API client applications. These rules are then added to the ILM policy to determine how and where object data is stored over time.

ILM settings determine the following aspects of an object:

- **Geography**

  The location of an object's data, either within the StorageGRID system (storage pool) or in a Cloud Storage Pool.

- **Storage grade**

  The type of storage used to store object data: for example flash or spinning disk.

- **Loss protection**

  How many copies are made and the types of copies that are created: replication, erasure coding, or both.

- **Retention**

  The changes over time to how an object's data is managed, where it is stored, and how it is protected from loss.

- **Protection during ingest**

  The method used to protect object data during ingest: synchronous placement (using the Balanced or Strict options for Ingest Behavior), or making interim copies (using the Dual commit option).

ILM rules can filter and select objects. For objects ingested using S3, ILM rules can filter objects based on the following metadata:

- Tenant account
- Bucket name
- Ingest time
- Key
- Last access time

> By default, updates to last access time are disabled for all S3 buckets. If your StorageGRID system includes an ILM rule that uses the Last access time option, you must enable updates to last access time for the S3 buckets specified in that rule. Use the PUT Bucket last access time request, the Tenant Manager (see Enable or disable last access time updates), or the Tenant Management API. When enabling last access time updates, be aware that StorageGRID performance might be reduced, especially in systems with small objects.

- Location constraint
- Object size
- User metadata
- Object tag

Related information

Use a tenant account

Manage objects with ILM

PUT Bucket last access time

Object versioning

You can use versioning to retain multiple versions of an object, which protects against accidental deletion of objects, and enables you to retrieve and restore earlier versions of an object.

The StorageGRID system implements versioning with support for most features, and with some limitations. StorageGRID supports up to 1,000 versions of each object.

Object versioning can be combined with StorageGRID information lifecycle management (ILM) or with S3 bucket lifecycle configuration. You must explicitly enable versioning for each bucket to turn on this functionality for the bucket. Each object in your bucket is assigned a version ID, which is generated by the StorageGRID system.

Using MFA (multi-factor authentication) Delete is not supported.

> Versioning can be enabled only on buckets created with StorageGRID version 10.3 or later.

ILM and versioning

ILM policies are applied to each version of an object. An ILM scanning process continuously scans all objects and re-evaluates them against the current ILM policy. Any changes you make to ILM policies are applied to all previously ingested objects. This includes previously ingested versions if versioning is enabled. ILM scanning applies new ILM changes to previously ingested objects.

For S3 objects in versioning-enabled buckets, versioning support allows you to create ILM rules that use “Noncurrent time” as the Reference time (select **Yes** for the question, “Apply this rule to older object versions only?” in Step 1 of the Create an ILM rule wizard). When an object is updated, its previous versions become noncurrent. Using a “Noncurrent time” filter allows you to create policies that reduce the storage impact of previous versions of objects.

> When you upload a new version of an object using a multipart upload operation, the noncurrent time for the original version of the object reflects when the multipart upload was created for the new version, not when the multipart upload was completed. In limited cases, the noncurrent time for the original version might be hours or days earlier than the time for the current version.

See ILM rules and policies for S3 versioned objects (Example 4).

Use S3 REST API to configure S3 Object Lock

If the global S3 Object Lock setting is enabled for your StorageGRID system, you can create buckets with S3 Object Lock enabled. You can specify default retention for each bucket or retention settings for each object version.

How to enable S3 Object Lock for a bucket

If the global S3 Object Lock setting is enabled for your StorageGRID system, you can optionally enable S3 Object Lock when you create each bucket.

S3 Object Lock is a permanent setting that can only be enabled when you create a bucket. You can't add or disable S3 Object Lock after a bucket is created.

To enable S3 Object Lock for a bucket, use either of these methods:

- Create the bucket using the Tenant Manager. See Create S3 bucket.
- Create the bucket using a PUT Bucket request with the `x-amz-bucket-object-lock-enabled` request header. See Operations on buckets.

S3 Object Lock requires bucket versioning, which is enabled automatically when the bucket is created. You can't suspend versioning for the bucket. See Object versioning.

Default retention settings for a bucket

When S3 Object Lock is enabled for a bucket, you can optionally enable default retention for the bucket and specify a default retention mode and default retention period.

Default retention mode

- In COMPLIANCE mode:
  - The object can't be deleted until its retain-until-date is reached.
  - The object's retain-until-date can be increased, but it can't be decreased.
  - The object's retain-until-date can't be removed until that date is reached.
- In GOVERNANCE mode:
  - Users with the `s3:BypassGovernanceRetention` permission can use the `x-amz-bypass-governance-retention: true` request header to bypass retention settings.
  - These users can delete an object version before its retain-until-date is reached.
  - These users can increase, decrease, or remove an object's retain-until-date.

Default retention period

Each bucket can have a default retention period specified in years or days.

How to set default retention for a bucket

To set the default retention for a bucket, use either of these methods:

- Manage bucket settings from the Tenant Manager. See Create an S3 bucket and Update S3 Object Lock default retention.
- Issue a PUT Object Lock Configuration request for the bucket to specify the default mode and default number of days or years.

PUT Object Lock Configuration

The PUT Object Lock Configuration request allows you to set and modify the default retention mode and default retention period for a bucket that has S3 Object Lock enabled. You can also remove previously configured default retention settings.

When new object versions are ingested to the bucket, the default retention mode is applied if `x-amz-object-lock-mode` and `x-amz-object-lock-retain-until-date` aren't specified. The default retention period

is used to calculate the retain-until-date if `x-amz-object-lock-retain-until-date` is not specified.

If the default retention period is modified after ingest of an object version, the retain-until-date of the object version remains the same and is not recalculated using the new default retention period.

You must have the `s3:PutBucketObjectLockConfiguration` permission, or be account root, to complete this operation.

The `Content-MD5` request header must be specified in the PUT request.

Request example

This example enables S3 Object Lock for a bucket and sets the default retention mode to COMPLIANCE and the default retention period to 6 years.

```
PUT /bucket?object-lock HTTP/1.1
Accept-Encoding: identity
Content-Length: 308
Host: host
Content-MD5: request header
User-Agent: s3sign/1.0.0 requests/2.24.0 python/3.8.2
X-Amz-Date: date
X-Amz-Content-SHA256: authorization-string
Authorization: authorization-string

<ObjectLockConfiguration>
    <ObjectLockEnabled>Enabled</ObjectLockEnabled>
    <Rule>
        <DefaultRetention>
            <Mode>COMPLIANCE</Mode>
            <Years>6</Years>
        </DefaultRetention>
    </Rule>
</ObjectLockConfiguration>
```

How to determine the default retention for a bucket

To determine if S3 Object Lock is enabled for a bucket and to see the default retention mode and retention period, use either of these methods:

- View the bucket in the Tenant Manager. See View S3 buckets.
- Issue a GET Object Lock Configuration request.

GET Object Lock Configuration

The GET Object Lock Configuration request allows you to determine if S3 Object Lock is enabled for a bucket and, if it is enabled, see if there is a default retention mode and retention period configured for the bucket.

When new object versions are ingested to the bucket, the default retention mode is applied if `x-amz-object-`

`lock-mode` is not specified. The default retention period is used to calculate the retain-until-date if `x-amz-object-lock-retain-until-date` is not specified.

You must have the `s3:GetBucketObjectLockConfiguration` permission, or be account root, to complete this operation.

Request example

```
GET /bucket?object-lock HTTP/1.1
Host: host
Accept-Encoding: identity
User-Agent: aws-cli/1.18.106 Python/3.8.2 Linux/4.4.0-18362-Microsoft
botocore/1.17.29
x-amz-date: date
x-amz-content-sha256: authorization-string
Authorization: authorization-string
```

Response example

```
HTTP/1.1 200 OK
x-amz-id-2:
iVmcB7OXXJRkRH1FiVq1151/T24gRfpwpuZrEG11Bb9ImOMAAe98oxSpXlknabA0LTvBYJpSIX
k=
x-amz-request-id: B34E94CACB2CEF6D
Date: Fri, 04 Sep 2020 22:47:09 GMT
Transfer-Encoding: chunked
Server: AmazonS3

<?xml version="1.0" encoding="UTF-8"?>
<ObjectLockConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
    <ObjectLockEnabled>Enabled</ObjectLockEnabled>
    <Rule>
        <DefaultRetention>
            <Mode>COMPLIANCE</Mode>
            <Years>6</Years>
        </DefaultRetention>
    </Rule>
</ObjectLockConfiguration>
```

How to specify retention settings for an object

A bucket with S3 Object Lock enabled can contain a combination of objects with and without S3 Object Lock retention settings.

Object-level retention settings are specified using the S3 REST API. The retention settings for an object override any default retention settings for the bucket.

You can specify the following settings for each object:

- **Retention mode**: Either COMPLIANCE or GOVERNANCE.
- **Retain-until-date**: A date specifying how long the object version must be retained by StorageGRID.
  - In COMPLIANCE mode, if the retain-until-date is in the future, the object can be retrieved, but it can't be modified or deleted. The retain-until-date can be increased, but this date can't be decreased or removed.
  - In GOVERNANCE mode, users with special permission can bypass the retain-until-date setting. They can delete an object version before its retention period has elapsed. They can also increase, decrease, or even remove the retain-until-date.
- **Legal hold**: Applying a legal hold to an object version immediately locks that object. For example, you might need to put a legal hold on an object that is related to an investigation or legal dispute. A legal hold has no expiration date, but remains in place until it is explicitly removed.

  The legal hold setting for an object is independent of the retention mode and the retain-until-date. If an object version is under a legal hold, no one can delete that version.

To specify S3 Object Lock settings when adding an object version to a bucket, issue a PUT Object, PUT Object - Copy, or Initiate Multipart Upload request.

You can use the following:

- `x-amz-object-lock-mode`, which can be COMPLIANCE or GOVERNANCE (case sensitive).

  > If you specify `x-amz-object-lock-mode`, you must also specify `x-amz-object-lock-retain-until-date`.

- `x-amz-object-lock-retain-until-date`
  - The retain-until-date value must be in the format `2020-08-10T21:46:00Z`. Fractional seconds are allowed, but only 3 decimal digits are preserved (milliseconds precision). Other ISO 8601 formats aren't allowed.
  - The retain-until-date must be in the future.
- `x-amz-object-lock-legal-hold`

  If legal hold is ON (case-sensitive), the object is placed under a legal hold. If legal hold is OFF, no legal hold is placed. Any other value results in a 400 Bad Request (InvalidArgument) error.

If you use any of these request headers, be aware of these restrictions:

- The `Content-MD5` request header is required if any `x-amz-object-lock-*` request header is present in the PUT Object request. `Content-MD5` is not required for PUT Object - Copy or Initiate Multipart Upload.
- If the bucket does not have S3 Object Lock enabled and a `x-amz-object-lock-*` request header is present, a 400 Bad Request (InvalidRequest) error is returned.
- The PUT Object request supports the use of `x-amz-storage-class: REDUCED_REDUNDANCY` to match AWS behavior. However, when an object is ingested into a bucket with S3 Object Lock enabled, StorageGRID will always perform a dual-commit ingest.
- A subsequent GET or HEAD Object version response will include the headers `x-amz-object-lock-`

`mode`, `x-amz-object-lock-retain-until-date`, and `x-amz-object-lock-legal-hold`, if configured and if the request sender has the correct `s3:Get*` permissions.

You can use the `s3:object-lock-remaining-retention-days` policy condition key to limit the minimum and maximum allowable retention periods for your objects.

How to update retention settings for an object

If you need to update the legal hold or retention settings for an existing object version, you can perform the following object subresource operations:

- `PUT Object legal-hold`

  If the new legal-hold value is ON, the object is placed under a legal hold. If the legal-hold value is OFF, the legal hold is lifted.

- `PUT Object retention`
  - The mode value can be COMPLIANCE or GOVERNANCE (case sensitive).
  - The retain-until-date value must be in the format `2020-08-10T21:46:00Z`. Fractional seconds are allowed, but only 3 decimal digits are preserved (milliseconds precision). Other ISO 8601 formats aren't allowed.
  - If an object version has an existing retain-until-date, you can only increase it. The new value must be in the future.

How to use GOVERNANCE mode

Users who have the `s3:BypassGovernanceRetention` permission can bypass the active retention settings of an object that uses GOVERNANCE mode. Any DELETE or PUT Object retention operations must include the `x-amz-bypass-governance-retention:true` request header. These users can perform these additional operations:

- Perform DELETE Object or DELETE Multiple Objects operations to delete an object version before its retention period has elapsed.

  Objects that are under a legal hold can't be deleted. Legal hold must be OFF.

- Perform PUT Object retention operations that change an object version's mode from GOVERNANCE to COMPLIANCE before the object's retention period has elapsed.

  Changing the mode from COMPLIANCE to GOVERNANCE is never allowed.

- Perform PUT Object retention operations to increase, decrease, or remove an object version's retention period.

Related information

- Manage objects with S3 Object Lock
- Use S3 Object Lock to retain objects
- Amazon Simple Storage Service User Guide: Using S3 Object Lock

Create S3 lifecycle configuration

You can create an S3 lifecycle configuration to control when specific objects are deleted from the StorageGRID system.

The simple example in this section illustrates how an S3 lifecycle configuration can control when certain objects are deleted (expired) from specific S3 buckets. The example in this section is for illustration purposes only. For complete details on creating S3 lifecycle configurations, see Amazon Simple Storage Service Developer Guide: Object lifecycle management. Note that StorageGRID only supports Expiration actions; it does not support Transition actions.

What lifecycle configuration is

A lifecycle configuration is a set of rules that are applied to the objects in specific S3 buckets. Each rule specifies which objects are affected and when those objects will expire (on a specific date or after some number of days).

StorageGRID supports up to 1,000 lifecycle rules in a lifecycle configuration. Each rule can include the following XML elements:

- Expiration: Delete an object when a specified date is reached or when a specified number of days is reached, starting from when the object was ingested.
- NoncurrentVersionExpiration: Delete an object when a specified number of days is reached, starting from when the object became noncurrent.
- Filter (Prefix, Tag)
- Status
- ID

If you apply a lifecycle configuration to a bucket, the lifecycle settings for the bucket always override StorageGRID ILM settings. StorageGRID uses the Expiration settings for the bucket, not ILM, to determine whether to delete or retain specific objects.

As a result, an object might be removed from the grid even though the placement instructions in an ILM rule still apply to the object. Or, an object might be retained on the grid even after any ILM placement instructions for the object have lapsed. For details, see How ILM operates throughout an object's life.

> Bucket lifecycle configuration can be used with buckets that have S3 Object Lock enabled, but bucket lifecycle configuration is not supported for legacy Compliant buckets.

StorageGRID supports the use of the following bucket operations to manage lifecycle configurations:

- DELETE Bucket lifecycle
- GET Bucket lifecycle
- PUT Bucket lifecycle

Create lifecycle configuration

As the first step in creating a lifecycle configuration, you create a JSON file that includes one or more rules. For example, this JSON file includes three rules, as follows:

1. Rule 1 applies only to objects that match the prefix `category1/` and that have a `key2` value of `tag2`. The `Expiration` parameter specifies that objects matching the filter will expire at midnight on 22 August 2020.
2. Rule 2 applies only to objects that match the prefix `category2/`. The `Expiration` parameter specifies that objects matching the filter will expire 100 days after they are ingested.

> Rules that specify a number of days are relative to when the object was ingested. If the current date exceeds the ingest date plus the number of days, some objects might be removed from the bucket as soon as the lifecycle configuration is applied.

3. Rule 3 applies only to objects that match the prefix `category3/`. The `Expiration` parameter specifies that any noncurrent versions of matching objects will expire 50 days after they become noncurrent.

```
{
    "Rules": [
        {
            "ID": "rule1",
            "Filter": {
                "And": {
                    "Prefix": "category1/",
                    "Tags": [
                        {
                            "Key": "key2",
                            "Value": "tag2"
                        }
                    ]
                }
            },
            "Expiration": {
                "Date": "2020-08-22T00:00:00Z"
            },
            "Status": "Enabled"
        },
        {
            "ID": "rule2",
            "Filter": {
                "Prefix": "category2/"
            },
            "Expiration": {
                "Days": 100
            },
            "Status": "Enabled"
        },
        {
            "ID": "rule3",
            "Filter": {
                "Prefix": "category3/"
            },
            "NoncurrentVersionExpiration": {
                "NoncurrentDays": 50
            },
            "Status": "Enabled"
        }
    ]
}
```

Apply lifecycle configuration to bucket

After you have created the lifecycle configuration file, you apply it to a bucket by issuing a PUT Bucket lifecycle request.

This request applies the lifecycle configuration in the example file to objects in a bucket named `testbucket`.

```
aws s3api --endpoint-url <StorageGRID endpoint> put-bucket-lifecycle-
configuration
--bucket testbucket --lifecycle-configuration file://bktjson.json
```

To validate that a lifecycle configuration was successfully applied to the bucket, issue a GET Bucket lifecycle request. For example:

```
aws s3api --endpoint-url <StorageGRID endpoint> get-bucket-lifecycle-
configuration
 --bucket testbucket
```

A successful response lists the lifecycle configuration you just applied.

Validate that bucket lifecycle expiration applies to object

You can determine if an expiration rule in the lifecycle configuration applies to a specific object when issuing a PUT Object, HEAD Object, or GET Object request. If a rule applies, the response includes an `Expiration` parameter that indicates when the object expires and which expiration rule was matched.

> Because bucket lifecycle overrides ILM, the `expiry-date` shown is the actual date the object will be deleted. For details, see How object retention is determined.

For example, this PUT Object request was issued on 22 Jun 2020 and places an object in the `testbucket` bucket.

```
aws s3api --endpoint-url <StorageGRID endpoint> put-object
--bucket testbucket --key obj2test2 --body bktjson.json
```

The success response indicates that the object will expire in 100 days (01 Oct 2020) and that it matched Rule 2 of the lifecycle configuration.

```
{
      *"Expiration": "expiry-date=\"Thu, 01 Oct 2020 09:07:49 GMT\", rule-
id=\"rule2\"",
      "ETag": "\"9762f8a803bc34f5340579d4446076f7\""
}
```

For example, this HEAD Object request was used to get metadata for the same object in the testbucket bucket.

```
aws s3api --endpoint-url <StorageGRID endpoint> head-object
--bucket testbucket --key obj2test2
```

The success response includes the object's metadata and indicates that the object will expire in 100 days and that it matched Rule 2.

```
{
      "AcceptRanges": "bytes",
      *"Expiration": "expiry-date=\"Thu, 01 Oct 2020 09:07:48 GMT\", rule-
id=\"rule2\"",
      "LastModified": "2020-06-23T09:07:48+00:00",
      "ContentLength": 921,
      "ETag": "\"9762f8a803bc34f5340579d4446076f7\""
      "ContentType": "binary/octet-stream",
      "Metadata": {}
}
```

Recommendations for implementing S3 REST API

You should follow these recommendations when implementing the S3 REST API for use with StorageGRID.

Recommendations for HEADs to non-existent objects

If your application routinely checks to see if an object exists at a path where you don't expect the object to actually exist, you should use the "Available" consistency control. For example, you should use the "Available" consistency control if your application HEADs a location before PUT-ing to it.

Otherwise, if the HEAD operation does not find the object, you might receive a high number of 500 Internal Server errors if one or more Storage Nodes are unavailable.

You can set the "Available" consistency control for each bucket using the PUT Bucket consistency request, or you can specify the consistency control in the request header for an individual API operation.

Recommendations for object keys

Follow these recommendations for object key names, based on when the bucket was first created.

Buckets created in StorageGRID 11.4 or earlier

- Don't use random values as the first four characters of object keys. This is in contrast to the former AWS recommendation for key prefixes. Instead, use non-random, non-unique prefixes, such as `image`.
- If you do follow the former AWS recommendation to use random and unique characters in key prefixes, prefix the object keys with a directory name. That is, use this format:

  `mybucket/mydir/f8e3-image3132.jpg`

  Instead of this format:

```
mybucket/f8e3-image3132.jpg
```

Buckets created in StorageGRID 11.4 or later

Restricting object key names to meet performance best practices is not required. In most cases, you can use random values for the first four characters of object key names.

> An exception to this is an S3 workload that continuously removes all objects after a short period of time. To minimize the performance impact for this use case, vary a leading portion of the key name every several thousand objects with something like the date. For example, suppose an S3 client typically writes 2,000 objects/second and the ILM or bucket lifecycle policy removes all objects after three days. To minimize the performance impact, you might name keys using a pattern like this: `/mybucket/mydir/yyyymmddhhmmss-random_UUID.jpg`

Recommendations for “range reads”

If the global option to compress stored objects is enabled, S3 client applications should avoid performing GET Object operations that specify a range of bytes be returned. These “range read” operations are inefficient because StorageGRID must effectively uncompress the objects to access the requested bytes. GET Object operations that request a small range of bytes from a very large object are especially inefficient; for example, it is inefficient to read a 10 MB range from a 50 GB compressed object.

If ranges are read from compressed objects, client requests can time out.

> If you need to compress objects and your client application must use range reads, increase the read timeout for the application.

Related information

- Consistency controls
- PUT Bucket consistency
- Administer StorageGRID

Copyright information

Trademark information