



# **How StorageGRID implements S3 REST API**

## **StorageGRID**

NetApp  
March 02, 2022

# Table of Contents

- How StorageGRID implements S3 REST API ..... 1
  - Conflicting client requests ..... 1
  - Consistency controls ..... 1
- How StorageGRID ILM rules manage objects ..... 4
- Object versioning ..... 5
- Recommendations for implementing S3 REST API ..... 6

# How StorageGRID implements S3 REST API

A client application can use S3 REST API calls to connect to StorageGRID to create, delete, and modify buckets, as well as storing and retrieving objects.

## 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 trade-off 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:

Consistency control	Description
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.

Consistency control	Description
read-after-new-write	<p>(Default) Provides read-after-write consistency for new objects and eventual consistency for object updates. Offers high availability and data protection guarantees. Most closely matches Amazon S3 consistency guarantees.</p> <p><b>Note:</b> If your application uses HEAD or GET requests on objects that do not exist, you might receive a high number of 500 Internal Server errors if one or more Storage Nodes are unavailable. To prevent these errors, set the consistency control to “available” unless you require consistency guarantees similar to Amazon S3.</p>
available (eventual consistency for HEAD or GET operations)	<p>Behaves the same as the “read-after-new-write” consistency level, but provides eventual consistency for HEAD and GET operations. Offers higher availability for HEAD and GET operations than “read-after-new-write” if Storage Nodes are unavailable. Differs from Amazon S3 consistency guarantees for HEAD and GET operations.</p>

## 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 the highest consistency level, “all,” which requires all copies of the object metadata to be available.

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 the “all” consistency level. Because this consistency level requires all copies of the object metadata to be available, you can receive a high number of 500 Internal Server errors if one or more Storage Nodes 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 until it reaches the “all” consistency level, so it does not require that all 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 are not 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:

- **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.
- **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.



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 cannot 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 request](#)

[PUT Bucket consistency request](#)

## 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. You can enable last access time updates using the PUT Bucket last access time request, the **S3 > Buckets > Configure Last Access Time** check box in the Tenant Manager, or using 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

For more information about ILM, see the instructions for managing objects with information lifecycle management.

#### **Related information**

[Use tenant account](#)

[Manage objects with ILM](#)

[PUT Bucket last access time request](#)

## **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. 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 the instructions for managing objects with information lifecycle management for an example ILM policy for S3 versioned objects.

### Related information

[Manage objects with ILM](#)

## 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 do not 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

For buckets that are created in StorageGRID 11.4 or later, restricting object key names to meet performance best practices is no longer required. For example, you can now use random values for the first four characters of object key names.

For buckets that were created in releases earlier than StorageGRID 11.4, continue to follow these recommendations for object key names:

- You should not use random values as the first four characters of object keys. This is in contrast to the former AWS recommendation for key prefixes. Instead, you should 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, you should 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
```

## Recommendations for “range reads”

If the **Compress Stored Objects** option is selected (**CONFIGURATION > System > Grid options**), 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 very 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 request](#)
- [Administer StorageGRID](#)

## Copyright Information

Copyright © 2022 NetApp, Inc. All rights reserved. Printed in the U.S. No part of this document covered by copyright may be reproduced in any form or by any means-graphic, electronic, or mechanical, including photocopying, recording, taping, or storage in an electronic retrieval system- without prior written permission of the copyright owner.

Software derived from copyrighted NetApp material is subject to the following license and disclaimer:

THIS SOFTWARE IS PROVIDED BY NETAPP "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, WHICH ARE HEREBY DISCLAIMED. IN NO EVENT SHALL NETAPP BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NetApp reserves the right to change any products described herein at any time, and without notice. NetApp assumes no responsibility or liability arising from the use of products described herein, except as expressly agreed to in writing by NetApp. The use or purchase of this product does not convey a license under any patent rights, trademark rights, or any other intellectual property rights of NetApp.

The product described in this manual may be protected by one or more U.S. patents, foreign patents, or pending applications.

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7103 (October 1988) and FAR 52-227-19 (June 1987).

## Trademark Information

NETAPP, the NETAPP logo, and the marks listed at <http://www.netapp.com/TM> are trademarks of NetApp, Inc. Other company and product names may be trademarks of their respective owners.