Object Lifecycle Management

Archived: 2026-04-05 15:55:34 UTC

Setup Object Lifecycle Management Configuration samples

To support common use cases like setting a Time to Live (TTL) for objects, retaining noncurrent versions of objects, or "downgrading" storage classes of objects to help manage costs, Cloud Storage offers the Object Lifecycle Management feature.

This page describes the feature as well as the options available when using it. For the general format of a lifecycle configuration file, see the bucket resource representation for JSON or the lifecycle configuration format for XML.

Introduction

In order to use Object Lifecycle Management, you define a lifecycle configuration, which must be set on a bucket. The configuration contains a set of rules which apply to current and future objects in the bucket. When an object meets the criteria of one of the rules, Cloud Storage automatically performs a specified action on the object. Here are some example use cases:

- Downgrade the storage class of objects older than 365 days to Coldline storage.
- Delete objects created before January 1, 2019.
- Keep only the 3 most recent versions of each object in a bucket with versioning enabled.

Lifecycle configuration

Each lifecycle management configuration contains a set of rules. Each rule contains one action and one or more conditions.

- An object has to match *all* of the conditions specified in a rule for the action in the rule to be taken.
- If you specify multiple rules that contain the same action, the action is taken on an object when that object matches the conditions in *any* of the rules.
- If multiple rules have their conditions satisfied simultaneously for a single object, Cloud Storage performs the action associated with only one of the rules, based on the following considerations:
  - The `Delete` action takes precedence over any `SetStorageClass` action.
  - The `SetStorageClass` action that switches the object to the storage class with the lowest at-rest storage pricing takes precedence.

  For example, if you have one rule that changes the object's class to Nearline storage and another rule that changes the object's class to Coldline storage, but both rules use the exact same condition, the object's class always changes to Coldline storage when the condition is met.

- You should test your lifecycle rules on development data before applying to production to ensure your rules don't perform actions under unintended sets of conditions. If that's not possible, you should test on a small subset of your production data by using the `matchesPrefix` or `matchesSuffix` conditions in your rules.
- Changes to a bucket's lifecycle configuration can take up to 24 hours to go into effect, and Object Lifecycle Management might still perform actions based on the old configuration during this time.

  For example, if you change an `age` condition from 10 days to 20 days, an object that is 11 days old could be deleted by Object Lifecycle Management up to 24 hours later, due to the criteria of the old configuration.

For use cases, see Configuration examples for Object Lifecycle Management.

Lifecycle actions

A lifecycle rule specifies exactly one of the following actions:

- `Delete`
- `SetStorageClass`
- `AbortIncompleteMultipartUpload`

Delete

The `Delete` action deletes an object when the object meets all conditions specified in the lifecycle rule. By default, when you delete a live object, it becomes soft-deleted, and Cloud Storage retains it for a duration of seven days. You can restore this soft-deleted object within the soft delete retention duration.

Exception: In buckets with Object Versioning enabled, deleting the live version of an object causes it to become a *noncurrent* version, while deleting a noncurrent version deletes that version from the bucket. See the configuration for deleting objects for an example of using the `Delete` action along with Object Versioning.

The `Delete` action does not take effect on an object while the object has an object hold placed on it or a retention policy that it has not yet fulfilled. As long as the conditions in the `Delete` action remain satisfied for the object, the `Delete` action occurs after any object hold is removed and any retention policy is fulfilled.

SetStorageClass

The `SetStorageClass` action changes the storage class of an object and updates the object's modification time when the object meets all conditions specified in the lifecycle rule.

`SetStorageClass` supports the following storage class transitions:

Original storage class	New storage class
Durable Reduced Availability (DRA) storage	Nearline storage Coldline storage

Original storage class	New storage class
	Archive storage Multi-Regional storage/Regional storage[1]
Standard storage, Multi-Regional storage, or Regional storage	Nearline storage Coldline storage Archive storage
Nearline storage	Coldline storage Archive storage
Coldline storage	Archive storage

[1] For buckets in a region, the new storage class cannot be Multi-Regional storage. For buckets in a multi-region or dual-region, the new storage class cannot be Regional storage.

Cloud Storage does not validate correctness of the storage class transition. This means that you can specify a storage class transition not listed in the above table, but the transition will not occur. You should verify that your lifecycle rules use one of the listed storage class transitions.

Abort incomplete multipart uploads

The `AbortIncompleteMultipartUpload` action aborts an incomplete multipart upload and deletes the associated parts when the multipart upload meets the conditions specified in the lifecycle rule.

Only the following lifecycle conditions can be used with this action:

- `age`
- `matchesPrefix`
- `matchesSuffix`

Attempting to create a rule that uses the `AbortIncompleteMultipartUpload` action in combination with other conditions results in an error.

Lifecycle conditions

A lifecycle rule includes conditions which an object must meet before the action defined in the rule occurs on the object. Lifecycle rules support the following conditions:

- `age`
- `createdBefore`
- `customTimeBefore`
- `daysSinceCustomTime`
- `daysSinceNoncurrentTime`
- `isLive`

- `matchesStorageClass`
- `matchesPrefix` and `matchesSuffix`
- `noncurrentTimeBefore`
- `numNewerVersions`

All conditions are optional, but at least one condition is required. If you attempt to set an invalid lifecycle configuration, such as by using an action or condition that does not exist, you receive a `400 Bad request` error response, and any existing lifecycle configuration remains in place.

`age`

The `age` condition is satisfied when a resource reaches the specified age (in days). Age is measured from the resource's creation time.

- For objects, the creation time is the time when the object is successfully written to the bucket, such as when an upload completes.
    - The age of an object is unaffected by the object becoming a noncurrent version.
- For multipart uploads, the creation time is the time when the upload is initiated.
- If the `age` condition is set to a value of `0`, the condition is satisfied at midnight UTC after the object is created.

For example, if a resource is created at 2022/01/10 10:00 UTC and the `age` condition is 10 days, then the condition is satisfied for the resource on and after 2022/01/20 10:00 UTC.

`createdBefore`

The `createdBefore` condition is satisfied when an object is created before midnight of the specified date in UTC.

`customTimeBefore`

The `customTimeBefore` condition is satisfied when the date portion of an object's `Custom-Time` metadata is earlier than the date specified in this condition. This condition is set using the date format `YYYY-MM-DD`. `customTimeBefore` is never satisfied for an object with no `Custom-Time` metadata set.

`daysSinceCustomTime`

The `daysSinceCustomTime` condition is satisfied when the specified number of days have passed since the date and time specified in an object's `Custom-Time` metadata field. For example, if an object's `Custom-Time` is `2020-05-16T10:00:00Z` and the `daysSinceCustomTime` condition is 10 days, then the condition is satisfied for the object on and after 2020/05/26 10:00 UTC.

`daysSinceCustomTime` is never satisfied for an object with no `Custom-Time` metadata set.

`daysSinceNoncurrentTime`

The `daysSinceNoncurrentTime` condition is typically only used in conjunction with Object Versioning. The condition is satisfied when the specified number of days have passed since the object became noncurrent, either because the live version was deleted or replaced. For example, if an object became noncurrent at 2020/07/08 15:00 UTC and the `daysSinceNoncurrentTime` condition is 10 days, then the condition is satisfied for the object on and after 2020/07/18 15:00 UTC.

`isLive`

The `isLive` condition is typically only used in conjunction with Object Versioning. When set to `false`, this condition is satisfied for any noncurrent version of an object. When set to `true`, this condition is satisfied for the live version of an object. If you don't use versioning, all your objects are considered live and match when `isLive` is `true`.

`matchesPrefix` **and** `matchesSuffix`

The `matchesPrefix` and `matchesSuffix` conditions are satisfied when the beginning or end of an object's name is an exact case-sensitive match with the specified prefix or suffix. You can specify multiple strings as a list (for example, `"matchesSuffix": [".jpg", ".png"]`).

When using `matchesPrefix`, don't include the bucket name or the `/` that precedes object names in most request paths. For example, in the Google Cloud CLI, the path to an object in a bucket named `my_bucket` has a format similar to `gs://my_bucket/pictures/paris_2022.jpg`. To match the object, you would use a condition such as `"matchesPrefix":["pictures/paris_"]`.

You can have up to 1000 prefixes and suffixes in total specified across all rules. In the Google Cloud console, you can copy and paste up to 1000 prefixes or suffixes in total. A prefix or suffix cannot be used twice in a single condition.

`matchesStorageClass`

The `matchesStorageClass` condition is satisfied when an object in the bucket is stored as the specified storage class. You can use the following values for `matchesStorageClass`: `STANDARD`, `NEARLINE`, `COLDLINE`, `ARCHIVE`, `MULTI_REGIONAL`, `REGIONAL`, and `DURABLE_REDUCED_AVAILABILITY`.

Generally, if you intend to use the `matchesStorageClass` condition on Standard storage objects, you should also include the following:

- If the bucket is in a region, include `REGIONAL` and `DURABLE_REDUCED_AVAILABILITY` in the condition.
- If the bucket is in a multi-region or dual-region, include `MULTI_REGIONAL` and `DURABLE_REDUCED_AVAILABILITY` in the condition.

Including these additional classes ensures the lifecycle rule covers older objects in your buckets which might be set to legacy storage classes.

`noncurrentTimeBefore`

The `noncurrentTimeBefore` condition is typically only used in conjunction with Object Versioning. The condition is satisfied for objects that became noncurrent on a date prior to the one specified in this condition. The condition is set using the date format `YYYY-MM-DD`. `noncurrentTimeBefore` is never satisfied for a live object.

`numNewerVersions`

The `numNewerVersions` condition is typically only used in conjunction with Object Versioning. If the value of this condition is set to *N*, an object version satisfies the condition when there are at least *N* versions (including the live version) newer than it. For a live object version, the number of newer versions is considered to be 0. For the most recent noncurrent version, the number of newer versions is 1 (or 0 if there is no live object version), and so on.

Object lifecycle behavior

When Object Lifecycle Management is configured for Cloud Storage buckets, Cloud Storage regularly inspects all the objects and performs all actions applicable according to the bucket's rules. Cloud Storage performs an action asynchronously, so there can be a lag between when the conditions are satisfied and when the action is taken. Your applications shouldn't rely on lifecycle actions occurring within a certain amount of time after a lifecycle condition is met.

For example, if an object meets the conditions for deletion, the object might not be deleted right away, and you see the object until the lifecycle action is executed on the object.

Applicable charges still apply while the object remains in its original state, with the exception of at-rest storage costs, which are waived if the object meets all of the following criteria:

- The object is in a bucket with soft delete disabled.
- The object is subject to a rule with a `Delete` action.
- The only condition for the rule is either an `age` condition or a combination of the `age` and `matchesStorageClass` conditions.
- The `age` condition is satisfied for the object in rules with only age condition. Both the `age` and `matchesStorageClass` conditions should be satisfied for the object if the delete rule has both conditions.
- The object does not have any object holds.

Object lifecycle behavior on versioned objects

In buckets with Object Versioning enabled, a live object that gets deleted according to lifecycle rules will exist in a noncurrent state for some amount of time. If the noncurrent version of the object also satisfies the delete rule's conditions, the noncurrent version of the object will also get deleted after the time elapses.

For example, say there's a lifecycle rule that deletes objects older than 180 days. If a live object is 200 days old, it gets deleted and becomes noncurrent. The now noncurrent object is still older than 180 days, so the noncurrent object also gets deleted after some time passes.

`SetStorageClass` cost considerations

Similar to changing an object's storage class manually, using `SetStorageClass` counts as a Class A operation and is billed at the rate determined by the destination storage class.

Unlike changing an object's storage class manually, using `SetStorageClass` does not rewrite an object. This gives Object Lifecycle Management certain pricing advantages:

- There are never inter-region replication charges, retrieval fees, or early deletion fees associated with the storage class change. However, early deletion fees apply if objects in a storage class with a specified minimum storage duration are deleted before the duration ends.
- The object's time spent set at the original storage class counts towards any minimum storage duration that applies for the new storage class.

For example, say you upload an object as Nearline storage, and 20 days later your lifecycle configuration changes the storage class of the object to Coldline storage. This change incurs no retrieval or early deletion fees. If you then delete the object 60 days after the storage class change, there is only a 10-day early deletion charge, since Coldline storage has a 90-day minimum storage duration, and the object existed for a total of 80 days.

In comparison, say you upload an object as Nearline storage, and 20 days later change the storage class using a rewrite (again to Coldline storage). This change incurs both a retrieval fee and a 10-day early deletion charge. If you then delete the object 60 days after the rewrite, there is a 30-day early deletion charge.

In both these examples, if soft delete is enabled on the bucket, the storage charges increase, but the early deletion charges reduce based on the length of the soft delete retention period.

Object creation time

In many cases, an object's upload completes soon after it begins; however, for uploads that occur over multiple requests, such as resumable uploads, there can be days between when the initial upload request is sent and when the final upload request is sent. In such cases, you should keep in mind the following:

- An object is not subject to lifecycle rules until after its upload completes.
- An object's creation time is based on when the upload completes. This impacts the `age` and `createdBefore` lifecycle conditions.
- When you set a `Custom-Time` for the object, you do so at the beginning of the upload. If you set a `Custom-Time` based on the time of the request, the `Custom-Time` could be much earlier than the object's creation time. This impacts the `customTimeBefore` and `daysSinceCustomTime` lifecycle conditions.

Expiration time metadata

If a `Delete` action is specified for a bucket with the `age` condition (and no other conditions besides `matchesStorageClass`), then some objects might be tagged with expiration time metadata. An object's expiration time indicates the time at which the object becomes (or became) eligible for deletion by Object Lifecycle Management. The expiration time might change as the bucket's lifecycle configuration or retention policy change.

Note that the absence of expiration time metadata does not necessarily mean the object will not be deleted, but rather that not enough information is available to determine when or if it will be deleted. For example, if the object creation time is 2020/01/10 10:00 UTC and the `age` condition is set to 10 days, then the object expiration time is 2020/01/20 10:00 UTC. However, the expiration time is not available for the object if:

- There are other conditions specified in the `Delete` rule, with the exception of `matchesStorageClass`.
- You use a `matchesStorageClass` condition that does not include the object's storage class.
- The object is under a hold, because Cloud Storage cannot know when the hold will be removed.
- Soft delete is enabled on your bucket.

You are not charged for storage after the object expiration time even if the object is not deleted immediately. You can continue to access the object before it is deleted and are responsible for other charges (request, network bandwidth). If the expiration time is not available for an object, the object is charged for storage until the time it is deleted.

When working with expiration times, keep in mind the following:

- If the bucket has a retention policy, the expiration time is the later of the Object Lifecycle Management `age` condition and the time the object satisfies the retention period specified by the retention policy.
- If there are multiple conflicting expiration times applicable for an object due to different lifecycle management rules, then the earliest applicable expiration time is used.

Options for tracking Lifecycle actions

To track the lifecycle management actions that Cloud Storage takes, use one of the following options:

- Use Cloud Storage usage logs. This feature logs both the action and who performed the action. A value of `GCS Lifecycle Management` in the `cs_user_agent` field of the log entry indicates the action was taken by Cloud Storage in accordance with a lifecycle configuration.
- Enable Pub/Sub Notifications for Cloud Storage for your bucket. This feature sends notifications to a Pub/Sub topic of your choice when specified actions occur. Note that this feature does not record who performed the actions.

What's next

- Enable Object Lifecycle Management.
- Explore lifecycle configuration examples.

- Review the general format of a lifecycle configuration in JSON API requests and XML API requests.
