To support the retrieval of objects that are deleted or overwritten, Cloud Storage offers the Object Versioning feature. This page describes the feature and the options available when using it. To learn how to enable and use Object Versioning see Using Object Versioning (/storage/docs/using-object-versioning).

Enable Object Versioning to protect your Cloud Storage data from being overwritten or accidentally deleted. Enabling Object Versioning increases storage costs (/storage/pricing), which can be partially mitigated by configuring Object Lifecycle Management (/storage/docs/lifecycle) to delete older object versions.

n: Objects cannot be recovered from a deleted bucket, even if the bucket used Object Versioning.

You enable Object Versioning for a *bucket*. Once enabled:

- Cloud Storage creates a *noncurrent* version of an object each time you perform an overwrite or delete of the *live* version, as long as you do not specify the generation number (/storage/docs/generations-preconditions#_Generations) of the live version.
    - Noncurrent versions retain the name of the object, but are uniquely identified by their generation number.
    - Noncurrent versions only appear in requests that explicitly call for object versions to be included.
- You permanently delete versions of objects by including the generation number in the deletion request or by using Object Lifecycle Management (/storage/docs/lifecycle).

Caution: Specifying the generation number in a deletion request permanently deletes the version, even if the version is live.

- Noncurrent versions of objects exist independently of any live version.

You can turn versioning on or off for a bucket at any time. Turning off versioning leaves existing object versions in place and causes the bucket to stop accumulating new noncurrent versions of objects.

ant: If you change the versioning configuration for a bucket, the change may take time to propagate, and object dele rwrites in the bucket may behave according to the previous setting for a short period of time. In order to ensure objec ns and overwrites behave according to your change, wait at least 30 seconds between changing the versioning uration and deleting or overwriting objects.

Cloud Storage uses two properties that together identify the version of an object. One property identifies the version of the object's data; the other property identifies the version of the object's metadata. These properties are always present with every version of the object, even if Object Versioning is not enabled. These properties can be used as preconditions for conditional updates (/storage/docs/generations-preconditions#_Preconditions) to enforce ordering of updates.

Cloud Storage marks every object using the following properties:

Property	Description
`generation`	Identifies the content (data) generation, and updates when the content of an object is overwritten. There is no relationship between the generation numbers of unrelated objects, even if the objects are in the same bucket.
`metageneration`	Identifies the metadata generation, and increases every time the metadata (/storage/docs/metadata#editable) **for a given content generation** is updated. **`metageneration`** is reset to **1** for each new **`generation`** of an object. The **`metageneration`** property has no meaning without the **`generation`** property and should be used only in conjunction with it. In other words, it is meaningless to compare metadata generations of two versions that have different data generations.

Object Versioning cannot be enabled on a bucket that currently has a retention policy (/storage/docs/bucket-lock).

n: There is no default limit on the number of object versions you can create. Each noncurrent version of an object is d at the same rate as the live version of the object. If you enable versioning, consider using Object Lifecycle Manage age/docs/lifecycle), which can remove the oldest versions of an object as newer versions become noncurrent.

Noncurrent versions of objects have their own metadata, which may differ from the metadata of the live version. Most importantly, a noncurrent version retains its ACLs (/storage/docs/access-control/lists)

and does not necessarily have the same permissions as the live version.

Each version, whether live or noncurrent, has one set of metadata; only the latest metageneration number refers to metadata. Older metageneration numbers cannot be used to access metadata that has since been changed.

You can update metadata for a noncurrent version of an object by specifying its `generation` in your request. To ensure safe read-modify-write semantics, you can use a metageneration-match precondition (/storage/docs/generations-preconditions#_Preconditions). Using this precondition causes the update to fail if the metadata you are attempting to update was changed between the time you read the metadata and sent the update.

This example shows what happens to the file `cat.jpg` in a bucket with Object Versioning enabled as you overwrite, update, and delete the file.

You upload a new image

When you first upload `cat.jpg` to Cloud Storage (/storage/docs/uploading-objects), it receives a `generation` number and a `metageneration` number. In this example, the generation number is `1360887697105000`. Because the object is new, the `metageneration` number is `1`.

`cat.jpg` receives `generation` and `metageneration` numbers even though Object Versioning is not enabled. You can view these numbers by using the `stat` command in gsutil. For instructions, see viewing the object metadata (/storage/docs/viewing-editing-metadata#view).

You enable Object Versioning

At this point, you decide to enable Object Versioning (/storage/docs/using-object-versioning#enable) for your bucket. Doing so does not affect the `generation` or `metageneration` numbers of `cat.jpg`.

You change the metadata of the image

You update the metadata for `cat.jpg` by adding custom metadata (/storage/docs/viewing-editing-metadata#edit): `color:black`. Updating metadata causes the `metageneration` value of `cat.jpg` to increase, in this case from `1` to `2`. However, the object itself remains unchanged, so Cloud Storage continues to store only one version of `cat.jpg`, and the version continues to have a `generation` number of `1360887697105000`.

You upload a new version of the image

You upload a new version of `cat.jpg` to your Cloud Storage bucket. When you do so, Object Versioning moves the existing `cat.jpg` object into a noncurrent state. The noncurrent version retains the same storage class and metadata it previously had. The noncurrent version appears only if you perform a versioned listing (/storage/docs/using-object-versioning#list): it does not appear in normal listing commands. The noncurrent version is now referenced as: `cat.jpg#1360887697105000`.

Meanwhile, the newly uploaded `cat.jpg` becomes the live version of the object. This new `cat.jpg` gets its own `generation` number, in this example `1360887759327000`. It also gets its own metadata and a `metageneration` number of `1`, which means it does not contain the `color:black` metadata unless you specify it. When you access or modify `cat.jpg`, this is the version that is used. You can alternatively refer to this version of `cat.jpg` using its `generation` number. For example, when using the gsutil tool you would refer to it as `cat.jpg#1360887759327000`.

You delete the live version of the image

You now delete `cat.jpg`. When you do this, the version that had generation number `1360887759327000` becomes noncurrent. Your bucket now contains two noncurrent versions of `cat.jpg` and no live versions. You can still refer to either noncurrent version by using its `generation` number, but if you try to access `cat.jpg` without a `generation` number, it fails.

Similarly, a normal object listing of the bucket will not show `cat.jpg` as one of the objects in the bucket. For information on listing noncurrent versions of objects, see Listing noncurrent object versions (/storage/docs/using-object-versioning#list).

You disable Object Versioning

You disable Object Versioning (/storage/docs/using-object-versioning#disable), which stops objects from becoming noncurrent. Existing noncurrent versions of objects remain in Cloud Storage. Even though Object Versioning is disabled, `cat.jpg#1360887697105000` and `cat.jpg#1360887759327000` remain stored in your bucket until you delete them, either manually (/storage/docs/using-object-versioning#delete) or through using Object Lifecycle Management (/storage/docs/lifecycle).

You restore one of the noncurrent versions

Even with Object Versioning disabled, you can restore one of the existing noncurrent versions by making a copy of it (/storage/docs/using-object-versioning#copy). To do so, simply name the copy you make

`cat.jpg`. Once you do this, your bucket has three versions of `cat.jpg`: the two noncurrent versions and the live version that came from making a copy.

This reference table shows what happens when you take certain actions with Object Versioning.

Object Versioning Status	Action	Result
Disabled		
	Overwrite **dog.png** with a new version.	The new version replaces the live version and receives a new generation number. The old live version is permanently deleted.
	Copy a noncurrent version of **dog.png** over the live version.[1]	A copy of the noncurrent version replaces the live version and receives a new generation number. The old live version is permanently deleted.
	Delete **dog.png**.	**dog.png** is permanently deleted.
	Delete a noncurrent version of **dog.png** by specifying its generation number.[1]	The noncurrent version is permanently deleted.
Enabled		
	Overwrite **dog.png** with a new version.	The new version replaces the live version and receives a new generation number. The old live version becomes a noncurrent version and keeps the same generation number.
	Copy a noncurrent version of **dog.png** over the live version.	A copy of the noncurrent version replaces the live version and receives a new generation number. The old live version becomes a noncurrent version and keeps the same generation number.
	Delete the live version of **dog.png** without specifying its generation number.	The live version becomes a noncurrent version and keeps the same generation number.
	Delete the live version of **dog.png** by specifying its generation number.	The live version is permanently deleted.

Object VersioningStatus	Action	Result
	Delete a noncurrent version of **dog.png** by specifying its generation number.	The noncurrent version is permanently deleted.

[1] A noncurrent version might exist if the bucket had Object Versioning enabled previously.

This section discusses tips to help you work with Object Versioning more effectively.

- The gsutil tool has comprehensive support for working with versioned objects that makes many tasks involving Object Versioning easier. For example, you can work with noncurrent versions of objects by appending `#` and the `generation` number to the object name. For more information on using gsutil with Object Versioning, see Object Versioning and Concurrency Control (/storage/docs/gsutil/addlhelp/ObjectVersioningandConcurrencyControl).

- Consider using `generation` and `metageneration` numbers for conditional updates instead of ETags. Together, `generation` and `metageneration` numbers keep track of all object updates, including metadata changes, providing a stronger guarantee than ETags.

- You can copy a noncurrent object version to the current live version. See Copying noncurrent object versions (/storage/docs/using-object-versioning#copy) for a step-by-step guide to copying noncurrent versions of objects.

  When you do this with Object Versioning enabled, if there already exists a live version of the object in your bucket, Cloud Storage overwrites it but also creates a new, noncurrent version of the overwritten object. In such a case, your bucket subsequently contains the overwritten object (now noncurrent) and two copies of the object that was previously noncurrent (one live copy and one still-noncurrent copy), all of which incur storage charges. To prevent unnecessary charges, delete the noncurrent version that you used to make the current live copy.

- When you use generation numbers, a request succeeds as long as there is an object with that name and generation number, regardless of whether it is live or noncurrent. If no such object exists, Cloud Storage returns `404 Not Found`.
- When you use generation-match preconditions (/storage/docs/generations-preconditions#_Preconditions), a request succeeds only if the live version of the requested object has the specified generation number. If no such object exists, or is noncurrent, Cloud Storage returns `412 Precondition Failed`.
- You should avoid using a generation-match precondition at the same time as a generation number in the object name. If you use both and the numbers match, the use of the precondition is redundant. If the numbers do not match, the request always fails.
- If you make several concurrent mutation requests with a generation-match precondition, Cloud Storage's strong consistency allows only one of those requests to succeed. This feature is useful if your objects are updated from several sources and you need to ensure that users don't accidentally overwrite them.
- If you set a generation-match precondition to `0` when uploading an object, Cloud Storage performs the specified request only if there is no live version of the object. For example, if you perform a `PUT` request with the XML API to create a new object with the header `x-goog-if-generation-match:0`, the request succeeds if the object does not exist, or if there are only noncurrent versions of the object. If there is a live version of the object, Cloud Storage aborts the update with a status code of `412 Precondition Failed`.