This page describes composite objects, which you create from existing objects without transferring additional object data. Composite objects are useful for making appends to an object, as well as for building an object using components that you've uploaded in parallel. For a step-by-step guide to performing object composition, see Composing objects (/storage/docs/composing-objects).

The compose operation creates a new *composite object* whose contents are the concatenation of a given sequence of source objects. The source objects all must:

- Have the same storage class (/storage/docs/storage-classes).
- Be stored in the same Cloud Storage bucket.
- **NOT** use customer-managed encryption keys (/storage/docs/encryption/customer-managed-keys).

When you perform a composition:

- The source objects are unaffected.
- You can use up to 32 source objects.
- Source objects can themselves be composite objects.
- The resulting composite object does not change if the source objects are subsequently replaced or deleted.
- When using gsutil (/storage/docs/gsutil) to perform object composition, the `Content-Type` of the resulting composite object is set to match the `Content-Type` of the first source object.

There are several differences between the metadata of a composite object and the metadata of other objects:

- Each composite object has a *component count* metadata field. The component count indicates the number of components in the object, where a single component is a non-composite object.

For example, say you have 3 source objects: 2 non-composite objects and 1 composite object which has a component count of 12. If you compose these 3 source objects, the resulting composite object would have a component count of 14.

  - While there is no limit to the number of components that a composite object can contain, the component count metadata field for an object saturates at 2,147,483,647.

    For example, say you have an object that contains 3,000,000,000 components. In this case, the component count for the object has a value of 2,147,483,647.

  - If you rewrite a composite object to a different location and/or storage class, the result is a composite object with a component count of 1.

- Composite objects *do not* have an MD5 hash metadata field.
- The ETag value of a composite object is not based on an MD5 hash, and client code should make no assumptions about composite object ETags except that they change whenever the underlying object changes per the IETF specification for HTTP/1.1 (https://tools.ietf.org/html/rfc7232#section-2.3).

Caution: Exercise caution when first using composite objects, since any clients expecting to find an MD5 digest within the ETag header may conclude that object data has been corrupted, which could trigger endless data retransmission attempts.

Cloud Storage uses CRC32C (http://tools.ietf.org/html/rfc4960#appendix-B) for integrity checking each source object at upload time and for allowing the caller to perform an integrity check of the resulting composite object when it is downloaded. CRC32C is an error detecting code that can be efficiently calculated from the CRC32C values of its components. Your application should use CRC32C as follows:

- When uploading source objects, you should calculate the CRC32C for each object using a CRC32C library such as one of those listed below, and include that value in your request.
- For the compose operation, you should include a CRC32C in the request. Cloud Storage responds with the CRC32C of the composite object, which can be validated by building a CRC32C value from the CRC32C values of the source objects.
- At download time, you should calculate the CRC32C of the downloaded object, and compare that with the value included in the response.

- If your application could change source objects between the time of uploading and composing those objects, you should specify generation-specific names for the source objects to avoid race conditions.

Libraries for computing CRC32C values include Boost (http://www.boost.org/doc/libs/1_42_0/libs/crc/) for C++, GoogleCloudPlatform crc32c (https://github.com/GoogleCloudPlatform/crc32c-java) for Java, crcmod (http://crcmod.sourceforge.net/) for Python, and digest-crc (https://github.com/postmodern/digest-crc) for Ruby. Note also that CRC32C is supported in hardware in current Intel CPUs.

CRC32C is not intended to protect against "man-in-the-middle" (https://en.wikipedia.org/wiki/Man-in-the-middle_atta s, where someone modifies the content in a way that still matches the provided checksum. Protection against such s is provided by using SSL connections when uploading and downloading objects (which is the default for many tool es, including gsutil and boto).

Object composition is a useful tool when performing parallel uploads or as a means of appending data to an object.

Object composition can be used for uploading an object in parallel: divide your data into multiple chunks, upload each chunk to a distinct object in parallel, compose your final object, and delete any temporary source objects.

n: Source objects and the resulting composite object are stored and billed as distinct objects. This means that:

Neglecting to delete temporary source objects incurs extra costs.

Storing temporary source objects as Nearline Storage, Coldline Storage, or Archive Storage incurs costs for early del

If you upload to a bucket where you've set a retention policy (/storage/docs/bucket-lock), you won't be able to delete temporary source objects until they meet the retention period.

In order to protect against changes to source objects between the upload and compose requests, you should provide an expected generation number for each source. For more information about object generations, see Generations and Preconditions (/storage/docs/generations-preconditions).

You can use the compose operation to perform limited object appends and edits.

You accomplish appending by uploading data to a temporary new object, composing the object you wish to append along with this new data, optionally naming the output of the compose operation the same as the original object, and deleting the temporary object.

You can also use composition to support a basic flavor of object editing. For example, you could compose an object *X* from the sequence *{Y1, Y2, Y3}*, replace the contents of *Y2*, and recompose *X* from those same components. Note that this requires that *Y1*, *Y2*, and *Y3* be left undeleted, so you will be billed for those components as well as for the composite.

ın: Compose operations create a new version of an object. When performing appends in a bucket with Object Versior age/docs/object-versioning) enabled, be sure that you properly manage the noncurrent version of the object that ea d generates.

- Compose an object (/storage/docs/composing-objects).
- Learn more about CRC32C and integrity checking in Cloud Storage (/storage/docs/hashes-etags).