This page discusses the conversion of files to and from a *gzip*-compressed state. The page includes an overview of transcoding, best practices for working with associated metadata, and compressed file behavior in Cloud Storage.

gzip is a form of data compression: it typically reduces the size of a file. This allows the file to be transferred faster and stored using less space than if it were not compressed. Compressing a file can reduce both cost and transfer time. *Transcoding*, in Cloud Storage, is the automatic changing of a file's compression before it's served to a requester. When transcoding results in a file becoming gzip-compressed, it can be considered *compressive*, whereas when the result is a file that is no longer gzip-compressed, it can be considered *decompressive*. Cloud Storage supports the decompressive form of transcoding.

Decompressive transcoding invalidates integrity checking on affected objects. This is because the stored hash repres
mpressed data, while the served data has compression removed. If requesters of your data rely on integrity checking
l not use decompressive transcoding.

If files are stored as gzip-compressed objects on Cloud Storage, they can be automatically decompressed before being sent to a requester, resulting in a file that is *identity encoded* (that is, not compressed). This allows for reduced storage costs for the object within Cloud Storage, but gives the requester the file itself, without any compression. This is useful, for example, when serving files to customers.

In order to be eligible for decompressive transcoding, an object must meet two criteria:

1. The file is gzip-compressed when stored in Cloud Storage.
2. The associated metadata (/storage/docs/metadata) includes `Content-Encoding: gzip`.

When an object meets these two criteria, it undergoes decompressive transcoding when requested, in which case it is also served without a `Content-Encoding` header. If you *want* an object that meets both criteria to be served in its compressed state (for example, to reduce egress cost or time), there are two ways to prevent decompressive transcoding from occurring:

- If the request for the object includes an `Accept-Encoding: gzip` header, the object is served as-is in that specific request, along with a `Content-Encoding: gzip` response header.
- If the `Cache-Control` (https://tools.ietf.org/html/rfc7234#section-5.2) metadata field for the object is set to `no-transform`, the object is served as a compressed object in all subsequent requests, regardless of any `Accept-Encoding` request headers.

While decompressive transcoding allows objects to be stored in Cloud Storage in a compressed state, saving space a
charges for downloading the object are based on its *decompressed* size, because that is the size of the served object
nformation, see the pricing guide (/storage/pricing).

There are several behaviors that you should be aware of concerning how `Content-Type` and `Content-Encoding` relate to transcoding. Both are metadata stored along with an object. See Viewing and Editing Object Metadata (/storage/docs/viewing-editing-metadata) for step-by-step instructions on how to add metadata to objects.

`Content-Type` (https://tools.ietf.org/html/rfc7231#section-3.1.1.5) should be included in all uploads and indicates the type of object being uploaded. For example:

indicates that the uploaded object is a plain-text file. While there is no check to guarantee the specified `Content-Type` matches the true nature of an uploaded object, incorrectly specifying its type will at best cause requesters to receive something other than what they were expecting and could lead to unintended behaviors.

`Content-Encoding` (https://tools.ietf.org/html/rfc7231#section-3.1.2.2) is optional and can, if desired, be included in the upload of files that are compressed. For example:

indicates that the uploaded object is gzip-compressed. As with `Content-Type`, there is no check to guarantee the specified `Content-Encoding` is actually applied to the uploaded object, and incorrectly specifying an object's encoding could lead to unintended behavior on subsequent download requests.

- When uploading a gzip-compressed object, the recommended way to set your metadata is to specify both the `Content-Type` and `Content-Encoding`. For example, for a compressed, plain-text file:

  This gives the most information about the state of the object to anyone accessing it. Doing so also makes the object eligible for decompressive transcoding when it is later downloaded, allowing client applications to handle the semantics of the `Content-Type` correctly.

  ★ **Note:** To automatically gzip and set the **`Content-Encoding`** header of files you upload, you can include the **`-z`** or **`-Z`** flag when using **`gsutil cp`** (/storage/docs/gsutil/commands/cp#options).

- Alternatively, you can upload the object with the `Content-Type` set to indicate compression and NO `Content-Encoding` at all. For example:

  However, in this case the only thing immediately known about the object is that it is gzip-compressed, with no information regarding the underlying object type. Moreover, the object is not eligible for decompressive transcoding.

- While it is possible to do so, a file that is gzip-compressed should not be uploaded with the compressed nature of the file omitted. For example, for a gzip-compressed plain-text file, you should avoid only setting `Content-Type: text/plain`. Doing so misrepresents the state of the object as it will be delivered to a requester.

- Similarly, objects should not be uploaded with an omitted `Content-Type`, even if a `Content-Encoding` is included. Doing so may result in `Content-Type` being set to a default value, but may result in the request being rejected, depending on how the upload is made.

- You **should not** set your metadata to redundantly report the compression of the object:

  This implies you are uploading a gzip-compressed object that has been gzip-compressed a second time, when that is not usually the case (if you *actually* plan to doubly compress a file, please see the using gzip on compressed objects (#gzip-gzip) section below). When decompressive transcoding occurs on such an incorrectly reported object, the object is served identity encoded, but requesters *think* that they have received an object which still has a layer of compression associated with it. Attempts to decompress the object will fail.

- Similarly, a file that is not gzip-compressed **should not** be uploaded with the `Content-Encoding: gzip`. Doing so makes the object *appear* to be eligible for transcoding, but when requests for the object are made, attempts at transcoding fail.

Some objects, such as many video, audio, and image files, not to mention gzip files themselves, are already compressed. Using gzip on such objects offers virtually no benefit: in almost all cases, doing so makes the object larger due to gzip overhead. For this reason, using gzip on compressed content is generally discouraged and may cause undesired behaviors.

For example, while Cloud Storage allows "doubly compressed" objects (that is, objects that are gzip-compressed but also have an underlying Content-Type that is itself compressed) to be uploaded and stored, it does not allow objects to be served in a doubly compressed state unless their `Cache-Control` metadata includes `no-transform`. Instead, it removes the outer, gzip, level of compression, drops the `Content-Encoding` response header, and serves the resulting object. This occurs even for requests with `Accept-Encoding: gzip`. The file that is received by the client thus does not have the same checksum as what was uploaded and stored in Cloud Storage, so any integrity checks fail.

When transcoding occurs, if the request for the object includes a `Range` header, that header is silently ignored. This means that requests for partial content are not fulfilled, and the response instead serves the entire requested object. For example, if you have a 10 GB object that is eligible for transcoding, but include the header `Range: bytes=0-10000` in the request, you still receive the entire 10 GB object.

This behavior arises because it is not possible to select a range from a compressed file without first decompressing the file in its entirety: each request for part of a file would be accompanied by the decompression of the entire, potentially large, file, which would poorly utilize resources. You should be aware of this behavior and avoid using the `Range` header when using transcoding, as charges are incurred for the transmission of the entire object and not just the range requested. For more information on allowed response behavior to requests with `Range` headers, see the specification (https://tools.ietf.org/html/rfc7233#section-3.1).

If requests with `Range` headers are needed, you should ensure that transcoding does not occur for the requested object. You can achieve this by choosing the appropriate properties when uploading objects to begin with. For example, range requests for objects with `Content-Type: application/gzip` and no `Content-Encoding` are performed as requested.

- Learn how to use the `-z`/`-Z` flag when using `gsutil cp` (/storage/docs/gsutil/commands/cp#options) to apply gzip content-encoding to file uploads.
- Learn how to view and edit object metadata (/storage/docs/viewing-editing-metadata).