This page contains a summary of best practices drawn from other pages in the Cloud Storage documentation. You can use the best practices listed here as a quick reference of what to keep in mind when building an application that uses Cloud Storage. Follow these best practices when launching a commercial application.

If you are just starting out with Cloud Storage, this page may not be the best place to start, because it does not teach you the basics of how to use Cloud Storage. If you are a new user, we suggest that you start with Getting Started: Using the Cloud Console (/storage/docs/gettingstarted-console) or Getting Started: Using the gsutil Tool (/storage/docs/gettingstarted-gsutil).

- The bucket namespace is global and publicly visible. Every bucket name must be unique across the entire Cloud Storage namespace. For more information, see Bucket and Object Naming Guidelines (/storage/docs/bucket-naming).
- If you need a lot of buckets, use GUIDs or an equivalent for bucket names, put retry logic in your code to handle name collisions, and keep a list to cross-reference your buckets. Another option is to use domain-named buckets (/storage/docs/domain-name-verification) and manage the bucket names as sub-domains.
- Don't use user IDs, email addresses, project names, project numbers, or any personally identifiable information (PII) in bucket names because anyone can probe for the existence of a bucket. Similarly, be very careful with putting PII in your object names, because object names appear in URLs for the object.
- Bucket names should conform to standard DNS naming conventions, because a bucket name can appear in a DNS record as part of a `CNAME` redirect. For details on bucket name requirements, see Bucket Name Requirements (/storage/docs/bucket-naming#requirements).
- Forward slashes in objects have no special meaning to Cloud Storage, as there is no native directory support. Because of this, deeply nested directory- like structures using slash delimiters are possible, but won't have the performance of a native filesystem listing deeply nested sub-directories.

- Avoid using sequential filenames such as timestamp-based filenames if you are uploading many files in parallel. Because files with sequential names are stored consecutively, they are likely to hit the same backend server, meaning that throughput will be constrained. In order to achieve optimal throughput, you can add the hash of the sequence number as part of the filename to make it non-sequential. For more information, see Request Rate and Access Distribution Guidelines (https://cloud.google.com/storage/docs/request-rate).
- Perform a back-of-the-envelope estimation of the amount of traffic that will be sent to Cloud Storage. Specifically, think about:
  - Operations per second. How many operations per second do you expect, for both buckets and objects, and for create, update, and delete operations.
  - Bandwidth. How much data will be sent, over what time frame?
  - Cache control. Specifying the `Cache-Control` metadata (/storage/docs/metadata#cache-control) on objects will benefit read latency on hot or frequently accessed objects. See Viewing and Editing Metadata (/storage/docs/viewing-editing-metadata#edit) for instructions for setting object metadata, such as `Cache-Control`.
- Design your application to minimize spikes in traffic. If there are clients of your application doing updates, spread them out throughout the day.
- While Cloud Storage has no upper bound on the request rate, for the best performance when scaling to high request rates, follow the Request Rate and Access Distribution Guidelines (/storage/docs/request-rate).
- Be aware that there are rate limits (/storage/quotas) for certain operations and design your application accordingly.
- If you get an error:
  - Retry with exponential backoff (/storage/docs/exponential-backoff) to avoid problems due to large traffic bursts.
  - Retry using a new connection and possibly re-resolve the domain name. This is to avoid "server stickiness" i.e. you want your retry to go through a different path to avoid hitting the same unhealthy component that the initial request hit.

- If your application is latency sensitive, use hedged requests. Hedged requests allow you to retry faster and cut down on tail latency. They do this while not reducing your request deadline, which could cause requests to time out prematurely. For more information, see https://www2.cs.duke.edu/courses/cps296.4/fall13/838-CloudPapers/dean_longtail.pdf
- Understand the performance level customers will expect from your application. This information will help you choose a storage option and region when creating new buckets.

- Data that will be served at a high rate with high availability should use the Standard Storage (/storage/docs/storage-classes#standard) class. This class provides the best availability with the trade-off of a higher price.
- Data that will be infrequently accessed and can tolerate slightly lower availability can be stored using the Nearline Storage (/storage/docs/storage-classes#nearline) or Coldline Storage (/storage/docs/storage-classes#coldline) class.
- Store your data in a region closest to your application's users. For instance, for EU data you might choose an EU bucket, and for US data you might choose a US bucket. For more information, see Bucket Locations (/storage/docs/locations).
- Keep compliance requirements in mind when choosing a location for user data. Are there legal requirements around the locations that your users will be providing data?

- The first and foremost precaution is: Never share your credentials. Each user should have distinct credentials.
- When you print out HTTP protocol details, your authentication credentials, such as OAuth 2.0 tokens, are visible in the headers. If you need to post protocol details to a message board or need to supply HTTP protocol details for troubleshooting, make sure that you sanitize or revoke any credentials that appear as part of the output.
- Always use TLS (HTTPS) to transport your data when you can. This ensures that your credentials as well as your data are protected as you transport data over the network. For example, to access the Cloud Storage API, you should use https://storage.googleapis.com.
- Make sure that you use an HTTPS library that validates server certificates. A lack of server certificate validation makes your application vulnerable to man-in-the-middle attacks or other

attacks. Be aware that HTTPS libraries shipped with certain commonly used implementation languages do not, by default, verify server certificates. For example, Python before version 3.2 has no built-in or complete support for server certificate validation, and you need to use third-party wrapper libraries to ensure your application validates server certificates.

- When applications no longer need access to your data, you should revoke their authentication credentials. For Google services and APIs, you can do this by logging into your Google Account Permissions (https://myaccount.google.com/permissions) and clicking on the unneeded applications, then clicking **Remove Access**.
- Make sure that you securely store your credentials. This can be done differently depending on your environment and where you store your credentials. For example, if you store your credentials in a configuration file, make sure that you set appropriate permissions on that file to prevent unwanted access. If you are using Google App Engine, consider using `StorageByKeyName` to store your credentials.
- Cloud Storage requests refer to buckets and objects by their names. As a result, even though ACLs will prevent unauthorized third parties from operating on buckets or objects, a third party can attempt requests with bucket or object names and determine their existence by observing the error responses. It can then be possible for information in bucket or object names to be leaked. If you are concerned about the privacy of your bucket or object names, you should take appropriate precautions, such as:
  - **Choosing bucket and object names that are difficult to guess.** For example, a bucket named `mybucket-gtbytul3` is random enough that unauthorized third parties cannot feasibly guess it or enumerate other bucket names from it.
  - **Avoiding use of sensitive information as part of bucket or object names.** For example, instead of naming your bucket `mysecretproject-prodbucket`, name it `somemeaninglesscodename-prod`. In some applications, you may want to keep sensitive metadata in custom Cloud Storage headers (/storage/docs/metadata#custom-metadata) such as `x-goog-meta`, rather than encoding the metadata in object names.
- Use groups in preference to explicitly listing large numbers of users. Not only does it scale better, it also provides a very efficient way to update the access control for a large number of objects all at once. Lastly, it's cheaper as you don't need to make a request per-object to change the ACLs.
- Before adding objects to a bucket, check that the default object ACLs (/storage/docs/access-control/lists#default) are set to your requirements first. This could save you a lot of time updating ACLs for individual objects.
- Bucket and object ACLs are independent of each other, which means that the ACLs on a bucket do not affect the ACLs on objects inside that bucket. It is possible for a user without

permissions for a bucket to have permissions for an object inside the bucket. For example, you can create a bucket such that only GroupA is granted permission to list the objects in the bucket, but then upload an object into that bucket that allows GroupB `READ` access to the object. GroupB will be able to read the object, but will not be able to view the contents of the bucket or perform bucket-related tasks.

- The Cloud Storage access control system includes the ability to specify that objects are publicly readable. Make sure you intend for any objects you write with this permission to be public. Once "published", data on the Internet can be copied to many places, so it's effectively impossible to regain read control over an object written with this permission.
- The Cloud Storage access control system includes the ability to specify that buckets are publicly writable. While configuring a bucket this way can be convenient for various purposes, we recommend against using this permission - it can be abused for distributing illegal content, viruses, and other malware, and the bucket owner is legally and financially responsible for the content stored in their buckets.

  If you need to make content available securely to users who don't have Google accounts we recommend you use signed URLs (/storage/docs/access-control/signed-urls). For example, with signed URLs you can provide a link to an object and your application's customers do not need to authenticate with Cloud Storage to access the object. When you create a signed URL you control the type (read, write, delete) and duration of access.
- If you use gsutil (https://cloud.google.com/storage/docs/gsutil), see these additional recommendations (/storage/docs/gsutil/addlhelp/SecurityandPrivacyConsiderations#recommended-user-precautions).

- If you use XMLHttpRequest (XHR) callbacks to get progress updates, do not close and re-open the connection if you detect that progress has stalled. Doing so creates a bad positive feedback loop during times of network congestion. When the network is congested, XHR callbacks can get backlogged behind the acknowledgement (ACK/NACK) activity from the upload stream, and closing and reopening the connection when this happens uses more network capacity at exactly the time when you can least afford it.
- For upload traffic, we recommend setting reasonably long timeouts. For a good end-user experience, you can set a client-side timer that updates the client status window with a message (e.g., "network congestion") when your application hasn't received an XHR callback for a long time. Don't just close the connection and try again when this happens.

- If you use Compute Engine instances with processes that `POST` to Cloud Storage to initiate a resumable upload (/storage/docs/resumable-uploads), then you should use Compute Engine instances in the same locations as your Cloud Storage buckets. You can then use a geo IP service to pick the Compute Engine region to which you route customer requests, which helps keep traffic localized to a geo-region.
- For resumable uploads, the resumable session should stay in the region in which it was created. Doing so reduces cross-region traffic that arises when reading and writing the session state, improving resumable upload performance.
- Avoid breaking a transfer into smaller chunks if possible and instead upload the entire content in a single chunk. Avoiding chunking removes fixed latency costs and improves throughput, as well as reducing QPS against Cloud Storage.

  Situations where you should consider uploading in chunks include when your source data is being generated dynamically, your clients have request size limitations (which is true for many browsers), or your clients are unable to stream bytes in a single request without first loading the full request into memory. If your clients receive an error, they can query the server for the commit offset and resume uploading (/storage/docs/performing-resumable-uploads#resume-upload) remaining bytes from that offset.
- If possible, avoid uploading content that has both `content-encoding: gzip` and a `content-type` that is compressed, as this may lead to unexpected behavior (/storage/docs/transcoding#gzip-gzip).

If you are concerned that your application software or users might erroneously delete or overwrite objects at some point, Cloud Storage has features that help you protect your data:

- A retention policy (/storage/docs/bucket-lock) that specifies a retention period can be placed on a bucket (/storage/docs/using-bucket-lock#set-policy). An object in the bucket cannot be deleted or overwritten until it reaches the specified age.
- An object hold (/storage/docs/bucket-lock#object-holds) can be placed on individual objects (/storage/docs/holding-objects#place-object-hold) to prevent anyone from deleting or overwriting the object until the hold is removed.
- Object Versioning (/storage/docs/object-versioning) can be enabled on a bucket in order to retain older versions of objects when they are deleted or overwritten. Object Versioning increases storage costs, but this can be partially mitigated by configuring Object Lifecycle Management (/storage/docs/lifecycle) to delete older object versions.

n: If Object Lifecycle Management (/storage/docs/lifecycle) causes millions of objects in your bucket to be changed
d, object listing performance is severely degraded while the lifecycle actions are occurring. Reach out to Google Clouc
rt (/support/) or your Account Manager before setting up such a policy.

If you just deleted a lot of objects from your bucket, object listing could temporarily become very slow. This is because the deleted records are not purged from the underlying storage system immediately, thus object listing needs to skip over the deleted records when finding the objects to return.

Eventually the deleted records are removed from the underlying storage system, and object listing performance becomes normal again. This typically takes a few hours, but in some cases may take a few days.

You should design your workload to avoid listing an object range with a lot of recent deletions. For example, if you are trying to delete objects from a bucket by repeatedly listing objects then deleting them, you should use the page token returned by the object listing response to issue the next listing request, instead of restarting the listing from the beginning for each request. When you restart your listing from the beginning, each request needs to skip over all of the objects that were just deleted, causing the object listing to become slower. If you have deleted a lot of objects under a certain prefix, then try to avoid listing objects under that prefix right after the deletions.

The Cross-Origin Resource Sharing (CORS) (/storage/docs/cross-origin) topic describes how to allow scripts hosted on other websites to access static resources stored in a Cloud Storage bucket. The converse scenario is when you allow scripts hosted in Cloud Storage to access static resources hosted on a website external to Cloud Storage. In the latter scenario, the website is serving CORS headers so that content on `storage.googleapis.com` is allowed access. It is recommended that you dedicate a specific bucket for this data access. For example, it is better to have the website serve the CORS header `Access-Control-Allow-Origin: https://mybucket.storage.googleapis.com` instead of `Access-Control-Allow-Origin: https://storage.googleapis.com`. This approach prevents your site from inadvertently over-exposing static resources to all of `storage.googleapis.com`.