Google Cloud

Best Practices for Using Cloud Storage

If you need to store and retrieve a large number of files, or you need to store multi terabyte file exports or dumps of data for data analytics pipelines, Cloud Storage is the ideal solution for you.

In the module "Best Practices for Using Cloud Storage," you learn concepts related to creating buckets, storing and retrieving file objects, and choosing suitable storage classes to address issues such as, data privacy, compliance and latency.

You learn which operations are eventually consistent, and which ones are strongly consistent. To design for scalability and resilience, you learn how to compose large objects using smaller component objects. You can use Cloud Storage to store and serve static content such as HTML, CSS and Javascript files.

We'll explore how to set up a course configuration to allow your website to pull content from the bucket. You'll learn best practices related to naming buckets and managing traffic. We'll also discuss how to secure buckets and objects using IAM permissions, access control lists, signed URLs, and more.

Google Cloud Storage concepts

Resources are entities in Google Cloud including:

- Projects

- Buckets - the basic Cloud Storage container

- Objects - the individual pieces of data that you store in Cloud Storage

First, we'll go over performing operations on buckets and objects.

Resources are entities in Google Cloud, and they include projects, buckets (which are the basic Cloud Storage container), and objects, the individual pieces of data that you store in Google Cloud Storage.

Storage classes

Storage Class	Minimum duration	Availability SLA	Typical monthly availability	Use cases	Name for APIs and gcloud storage
Standard Storage	None	Multi-region 99.95% Dual-region 99.95% Region 99.9%	>99.99% availability in multi-regions and dual-regions; 99.99% in regions	Access data frequently ("hot" data) and/or store for brief periods ● Serve website content ● Stream videos ● Interactive workloads ● Mobile and gaming apps	STANDARD
Nearline Storage	30 days	Multi-region 99.9% Dual-region 99.9% Region 99.0%	99.95% availability in multi-regions and dual-regions; 99.9% in regions	Read/modify data ≤ once per month ● Data backup ● Serve long-tail multimedia content	NEARLINE
Coldline Storage	90 days			Read/modify data no more than once a quarter	COLDLINE
Archive Storage	365 days	None		Read/modify data < once a year ● Cold data storage ● Disaster recovery	ARCHIVE

Google Cloud Storage has four primary storage classes, with different characteristics, use cases, and prices for your needs.

Storage classes

Storage Class	Minimum duration	Availability SLA	Typical monthly availability	Use cases	Name for APIs and gcloud storage
Standard Storage	None	Multi-region 99.95% Dual-region 99.95% Region 99.9%	>99.99% availability in multi-regions and dual-regions; 99.99% in regions	Access data frequently ("hot" data) and/or store for brief periods ● Serve website content ● Stream videos ● Interactive workloads ● Mobile and gaming apps	STANDARD
Nearline Storage	30 days	Multi-region 99.9% Dual-region 99.9% Region 99.0%	99.95% availability in multi-regions and dual-regions; 99.9% in regions	Read/modify data ≤ once per month ● Data backup ● Serve long-tail multimedia content	NEARLINE
Coldline Storage	90 days			Read/modify data no more than once a quarter	COLDLINE
Archive Storage	365 days	None		Read/modify data < once a year ● Cold data storage ● Disaster recovery	ARCHIVE

Standard Storage is best for data that is frequently accessed or stored for only brief periods of time. There is no minimum duration for data stored using the Standard Storage class.

When used in a single region, co-locating your resources maximizes the performance for data-intensive computations and can reduce network charges.

When used in a dual-region, you still get optimized performance when accessing Google Cloud products that are located in one of the associated regions, but you also get the improved availability and fault tolerance that comes from storing data in geographically separate locations.

When used in a multi-region, Standard Storage is appropriate for storing data that is accessed around the world.

Standard Storage use cases include serving website content, streaming videos, executing interactive workloads, and serving data supporting mobile and gaming applications.

Storage classes

Storage Class	Minimum duration	Availability SLA	Typical monthly availability	Use cases	Name for APIs and gcloud storage
Standard Storage	None	Multi-region 99.95% Dual-region 99.95% Region 99.9%	>99.99% availability in multi-regions and dual-regions; 99.99% in regions	Access data frequently ("hot" data) and/or store for brief periods • Serve website content • Stream videos • Interactive workloads • Mobile and gaming apps	STANDARD
Nearline Storage	30 days	Multi-region 99.9% Dual-region 99.9% Region 99.0%	99.95% availability in multi-regions and dual-regions; 99.9% in regions	Read/modify data ≤ once per month • Data backup • Serve long-tail multimedia content	NEARLINE
Coldline Storage	90 days			Read/modify data no more than once a quarter	COLDLINE
Archive Storage	365 days	None		Read/modify data < once a year • Cold data storage • Disaster recovery	ARCHIVE

Nearline Storage is a low-cost, highly durable storage service for storing infrequently accessed data.

Nearline Storage is a better choice than Standard Storage in scenarios where slightly lower availability, a 30-day minimum storage duration, and costs for data access are acceptable trade-offs for lower at-rest storage costs.

Nearline Storage is ideal for data you plan to read or modify on average once per month or less.

Nearline Storage is appropriate for data backup, long-tail multimedia content, and data archiving.

Storage classes

Storage Class	Minimum duration	Availability SLA	Typical monthly availability	Use cases	Name for APIs and gcloud storage
Standard Storage	None	Multi-region 99.95% Dual-region 99.95% Region 99.9%	>99.99% availability in multi-regions and dual-regions; 99.99% in regions	Access data frequently ("hot" data) and/or store for brief periods • Serve website content • Stream videos • Interactive workloads • Mobile and gaming apps	STANDARD
Nearline Storage	30 days	Multi-region 99.9% Dual-region 99.9% Region 99.0%	99.95% availability in multi-regions and dual-regions; 99.9% in regions	Read/modify data ≤ once per month • Data backup • Serve long-tail multimedia content	NEARLINE
Coldline Storage	90 days			Read/modify data no more than once a quarter	COLDLINE
Archive Storage	365 days	None		Read/modify data < once a year • Cold data storage • Disaster recovery	ARCHIVE

Coldline Storage is a very-low-cost, highly durable storage service for storing infrequently accessed data.

Coldline Storage is a better choice than Standard Storage or Nearline Storage in scenarios where slightly lower availability, a 90-day minimum storage duration, and higher costs for data access are acceptable trade-offs for lower at-rest storage costs.

Coldline Storage is ideal for data you plan to read or modify at most once a quarter.

Storage classes

Storage Class	Minimum duration	Availability SLA	Typical monthly availability	Use cases	Name for APIs and gcloud storage
Standard Storage	None	Multi-region 99.95% Dual-region 99.95% Region 99.9%	>99.99% availability in multi-regions and dual-regions; 99.99% in regions	Access data frequently ("hot" data) and/or store for brief periods • Serve website content • Stream videos • Interactive workloads • Mobile and gaming apps	STANDARD
Nearline Storage	30 days	Multi-region 99.9% Dual-region 99.9% Region 99.0%	99.95% availability in multi-regions and dual-regions; 99.9% in regions	Read/modify data ≤ once per month • Data backup • Serve long-tail multimedia content	NEARLINE
Coldline Storage	90 days			Read/modify data no more than once a quarter	COLDLINE
Archive Storage	365 days	None		Read/modify data < once a year • Cold data storage • Disaster recovery	ARCHIVE

Archive Storage is the lowest-cost, highly durable storage service for data archiving, online backup, and disaster recovery.

Archive Storage has higher costs for data access and operations, as well as a 365-day minimum storage duration.

Archive Storage is the best choice for data that you plan to access less than once a year, or never at all.

Archive Storage is excellent for cold data storage, such as data stored for legal or regulatory reasons. It is also a good choice for data that will be used only for disaster recovery.

Characteristics applicable to all storage classes

- Unlimited storage with no minimum object size.
- Worldwide accessibility and worldwide storage locations.
- Low latency (time to first byte typically tens of milliseconds).
- High durability (99.999999999% annual durability).
- Geo-redundancy if the data is stored in a multi-region or dual-region.
- A uniform experience with Cloud Storage features, security, tools, and APIs.

We've discussed the four primary storage classes, and differentiated between them in terms of minimum duration, availability and use cases.

There are a number of characteristics that apply to all Google Cloud Storage data, regardless of storage class.

These include:
- Unlimited storage with no minimum object size requirement,
- Worldwide accessibility and storage locations,
- Low latency and high durability,
- Geo-redundancy if data is stored in a multi-region or dual-region, and
- A uniform experience, which extends to security, tools, and APIs.

The following operations are strongly consistent

- Read-after-write

- Read-after-metadata-update

- Read-after-delete

- Bucket listing

- Object listing

- Granting access to resources

Bucket

Success
Response

Write   Read
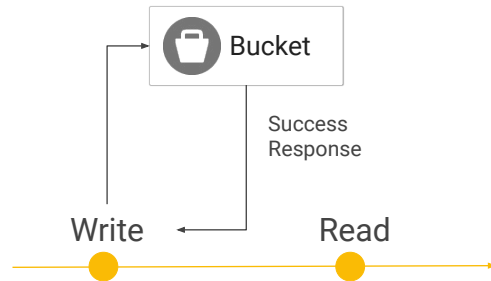
Strongly consistent: when you *perform an operation* in Cloud Storage and *receive a success response,* the object is *immediately available* for download and metadata operations.

Strongly consistent operations mean that when you perform an operation in Cloud Storage and you receive a success response, the object is immediately available for download and metadata operations from any location where Google offers service. The following operations in Google Cloud Storage are strongly consistent: read-after-write, read-after-metadata-update, read-after-delete, bucket listing, object listing, and granting access to resources.

The following operations are eventually consistent

- Revoking access from objects

- Accessing publicly readable cached objects

Bucket

Success Response

Revoke access

Access revoked

Revoking access can take up to a minute to take effect.
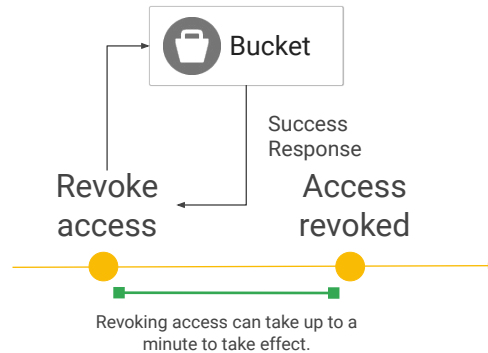
Eventually consistent: when you perform an operation, it may take some time for the operations to take effect.

The following operations are eventually consistent: revoking access from objects and accessing publicly readable cached objects. Revoking access can take up to a minute for changes to take effect, so operations won't be consistent until the revocation has taken place. Publicly readable cached objects are eventually consistent if the object is in the cache when it is updated or deleted. When the object in the cache is updated or deleted, the operation doesn't take effect until its cache lifetime expires.

Use the following request endpoints

	URIs	HTTP	HTTPS
Typical API requests	XML: storage.googleapis.com/<bucket>/<object> <bucket>.storage.googleapis.com/<object> JSON: www.googleapis.com/download/storage/v1/b/<bucket>/o/<object-encoded-as -URL-path-segment>?alt=media	✅	✅
CNAME redirects	Use the following URI in the host name portion of your CNAME record: c.storage.googleapis.com. Example: if you publish travel-maps.example.com CNAME c.storage.googleapis.com., you could access the object at: http://travel-maps.example.com/paris.jpg	✅	
Authenticated browser downloads	To download an object using cookie-based authentications: https://storage.cloud.google.com/<bucket>/<object>		✅
Content-based load balancing	Create backend Cloud Storage buckets and serve content based on the URL sent to an external HTTPS load balancer. Example: a file in an EU region bucket could be accessed using https://example.com/static/eu/content.html, and a file in a US bucket could be accessed using https://example.com/static/us/content.html.	✅	✅

Depending on the operation you are performing and your application requirements, you can access Cloud Storage through three request endpoints (URIs).

Using the XML API, you can use the two listed URIs, and using the JSON API, you can use the listed URI. These URIs support secure sockets layer (SSL) encryption, so you can use either HTTP or HTTPS. If you authenticate to the Cloud Storage API using OAuth 2.0, you should use HTTPS.

A CNAME redirect is a special DNS record that lets you use a URI from your own domain to access a resource in Cloud Storage without revealing the Cloud Storage URI. For example, if you published the following, *travel-maps.example.com CNAME c.storage.googleapis.com*, you could access an object with the following URI: http://travel-maps.example.com/paris.jpg. You can only use CNAME redirects with HTTP.

You can also access resources using authenticated browser downloads, which let you download data through your browser if you are signed in to your Google account and have been granted permission to read the data. Authenticated browser downloads use cookie-based Google account authentication in conjunction with Google account-based ACLs. To download an object using cookie-based authentication, you must use the following URI: https://storage.cloud.google.com/<bucket>/<object>. Only HTTPS is supported.

You can modify the content-based load balancing configuration to route requests for

static content to a Cloud Storage bucket. Requests to URI paths that begin with */static* can be sent to your storage bucket, and all other requests can be sent to your virtual machine instances. You can also use content-based load balancing to route to buckets in a specific region. For example, requests to a path starting with */static/eu* could route to a bucket in an EU region.

Composite objects and parallel uploads

Combine up to 32 objects into a single new object.

Use cases include:

- Dividing your data and uploading each chunk to a distinct object, composing your final object, and deleting any temporary objects.
- Uploading data to a temporary new object, composing it with the object you want to append it to, and deleting the temporary object.

Compose an object of smaller chunks using `gcloud storage`:

```
gcloud storage objects compose gs://example-bucket/component-obj-1

gs://example-bucket/component-obj-2

gs://example-bucket/composite-object
```

You can compose up to 32 existing objects into a new object without transferring additional object data.

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

In parallel uploads, you divide an object into multiple pieces, upload the pieces to a temporary location simultaneously, and compose the original object from these temporary pieces.

One use case is to fully use your available bandwidth by dividing your data into chunks and uploading them separately. Another is to copy many files in parallel with a single command in big data scenarios.

Design your application to handle network failures with truncated exponential backoff

Truncated exponential backoff:

- Is a standard error-handling strategy for network applications.
- Periodically retries failed requests with increasing delays between requests.
- Should be used for all requests to Cloud Storage that return HTTP 5xx and 429 response codes.

```
@retry(wait_exponential_multiplier=1000, wait_exponential_max=10000)

def wait_exponential_1000():

  print "Wait 2^x * 1000 milliseconds between each retry, up to 10
  seconds, then 10 seconds afterwards"
```

Truncated exponential backoff is a standard error-handling strategy for network applications in which a client periodically retries a failed request with increasing delays between requests.

Clients should use truncated exponential backoff for all requests to Cloud Storage that return HTTP 5xx and 429 response codes, including uploads and downloads of data or metadata.

Understanding how truncated exponential backoff works is important if you are building client applications that use the Cloud Storage XML API or JSON API directly, accessing Cloud Storage through a client library, or using the gcloud storage command line tool.

The Google Cloud console sends requests to Cloud Storage on your behalf and will handle any necessary backoff.

The example shows a backoff implementation for the Python retry library when you retry distributed services and other remote endpoints.

☁ Demo

Enable CORS Configuration in Cloud Storage

1. Create a Compute Engine instance
2. Install software prerequisites
3. Download and configure the demo application
4. Demo the working application
5. Relocate the data.json file and update the demo application
6. Apply CORS configuration

Demo: Enabling CORS with Cloud Storage

You have a script on a page hosted from Google App Engine at example.appspot.com and want to use static resources stored in a Cloud Storage bucket at example.storage.googleapis.com. The browser won't allow a script from example.appspot.com to fetch resources from example.storage.googleapis.com using XMLHttpRequest because the resource being fetched is from a different origin.

The Cross Origin Resource Sharing (CORS) spec was developed by the World Wide Web Consortium (W3C) to get around this limitation. Cloud Storage supports this specification by allowing you to configure your buckets to return CORS-compliant responses. Because Cloud Storage supports CORS, a browser can ask example.storage.googleapis.com for permission to share its resources with scripts from example.appspot.com.

For more information, see: https://cloud.google.com/storage/docs/cross-origin

Best Practices for Using
Cloud Storage

Let's discuss some best practices for using Cloud Storage.

Follow these best practices for naming

Do	Don't
Use globally unique bucket names: `us-east1-appdev-bucket01` ✓	Use personally identifiable information (PII): `johndoebucket` ✗

Use GUIDs or the equivalent if your application needs a lot of buckets:
`037763b8-2b55-4887-bbb9-600e2c6c6014`
`f41b0c0c-9cfc-405d-970e-d4ee46e41279`
`7f62c421-ce22-4c3f-ad9c-17f342500f62` ✓
	Use user IDs, emails, project names/IDs, etc:
	`johndoe`
	`project-gcp-sales` ✗

Conform to standard DNS naming conventions: `mycompany-unique-bucket01` `yourllc-011-eu-central1` ✓	Use IP address notation: `192.168.5.4` ✗
	Use the goog prefix or include any spelling or close misspelling of google: `goog_bucket01284` `bucket-for-google-course01` ✗

Google Cloud Storage bucket names are global and publicly visible, and must be unique across the entire Google Cloud Storage service. If your applications require many buckets, use GUIDs or the equivalent for bucket names. When creating a bucket name, your application should have retry logic in place to handle name collisions. Consider keeping a list to cross-reference your buckets.

Avoid using any information in bucket names that can be used to probe for the existence of other resources. Use caution if putting personally identifiable information in object or bucket names, because they do appear in URLs. Bucket names should conform to standard DNS naming conventions, because the bucket name can appear in a DNS record as part of a CNAME redirect.

If you use part of your company's domain name in a bucket name, forward or reverse, Google will try to verify that you own the domain. Avoid using your company domain name in a bucket unless your DNS admin can verify ownership.

This slide also shows examples of bucket names that do not follow best practices.

# Follow these best practices for Cloud Storage traffic

- Consider:
  - Operations per second
  - Bandwidth
  - Cache control

- Design your application to minimize spikes in traffic.

- Use exponential backoff if you get an error.

- For request rates > **1000 write requests/second** or **5000 read requests/second**:

  - Start with a request rate below or near the threshold.

  - Double the request rate no faster than every 20 minutes.

Consider operations per second, bandwidth (how much data will be sent over what time frame), and cache control when designing your application with Cloud Storage. If you specify the Cache-Control metadata on objects, read latency will be lowered on hot or frequently accessed objects.

Design your application so that it minimizes spikes in traffic, spreading updates throughout the day. Use exponential backoff if you get an error.

Google Storage has no upper bound on request rate, but for best performance when scaling to high request rates, follow the request rate and access distribution guidelines. If your request rate is less than 1000 write requests per second or 5000 read requests per second, then no ramp-up is needed. If your request rate is expected to go over these thresholds, you should start with a request rate below or near the thresholds, and then double the request rate no faster than every 20 minutes.

Consider the location and availability of your data
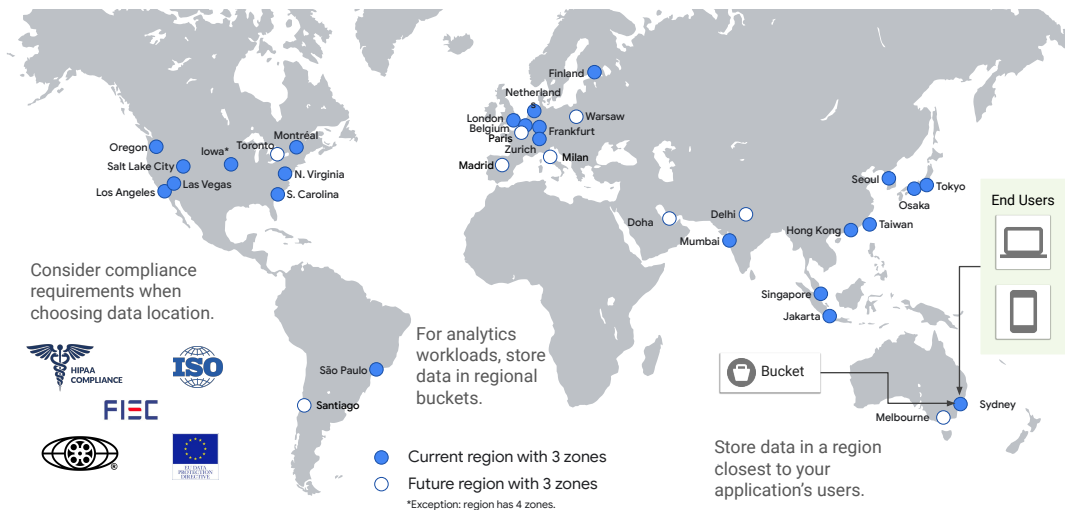
Consider the location and required availability of your data when choosing your storage options.

Store data in a region closest to your application's users, and consider region-specific compliance requirements when choosing data location. For analytics workloads, store your data in regional buckets to reduce network charges and for better performance, as compared to multi-region or dual-region.

Consider the characteristics of your data

- Standard Storage:
  - Use for data served at a high rate with high availability
  - No minimum storage duration and lowest operation charges

- Nearline Storage and Coldline Storage:
  - Use for infrequently accessed data that tolerates slightly lower availability
  - 30 and 90 day minimum storage durations respectively with proportionate operation and storage charges

- Archive Storage:
  - Use for data accessed less than once a year with no availability SLA
  - 365 days minimum storage duration and lowest per-month storage charges

Standard Storage provides the best availability of the storage classes, with the lowest operation charges and no minimum duration. It does, however, have a higher per-month storage price. Standard Storage is a good option for data that is served at a high rate with high availability.

Nearline Storage and Coldline Storage are good options for infrequently accessed data and for data that tolerates slightly lower availability. With minimum storage durations of 30 days and 90 days respectively, operation charges are less than Archive Storage, and storage charges are less than Standard Storage.

Archive Storage is suited for data that is accessed less than one a year, such as data stored for legal or regulatory reasons, or data required for data recovery. Archive Storage has no availability SLA, though the typical availability is comparable to Nearline Storage and Coldline Storage. Archive Storage has the lowest per-month storage charges with a minimum storage duration of 365 days.

Secure your buckets using the following options

Use Identity and Access Management (IAM) permissions to grant:
- Access to buckets
- Bulk access to a bucket's objects

Use Signed Policy Documents to:
- Specify what can be uploaded to a bucket.
- Control size, content type, and other upload characteristics.

Use Access Control Lists (ACLs) to grant:
- Read or write access to users for individual buckets or objects.
- Access when fine-grained control over individual objects is required.

Firebase Security Rules provide:
Granular, attribute-based access control to mobile and web apps using the Firebase SDKs for Cloud Storage.

Signed URLs (query string authentication):
- Provide time-limited read or write access to an object through a generated URL.
- Can be created using gcloud storage or programmatically.

You can control access to your Cloud Storage buckets and objects using these options.

You can use Identity and Access Management permissions to grant access to buckets and to provide bulk access to a bucket's objects. IAM permissions do not give you fine-grained control over individual objects.

You can use Access Control Lists, or ACLs, to grant read or write access to users for individual buckets or objects. It is recommended that you only use ACLs when you need fine-grained control over individual objects.

Use signed URL's query string authentication to provide time-limited read or write access to an object through a URL you generate. The shared URL provides access to anyone it's shared with for the duration specified. You can create signed URLs using gcloud storage, or programmatically in your application.

Signed policy documents allow you to specify what can be uploaded to a bucket. They allow greater control over size, content type, and other upload characteristics than using signed URLs. They are used by website owners to allow visitors to upload files to Cloud Storage. Signed policy documents only work with form posts.

Firebase security rules provide granular, attribute-based access control to mobile and web applications using the Firebase SDKs for Cloud Storage.

Consider these additional security best practices

- Use TLS (HTTPS) to transport data.
- Use an HTTPS library that validates server certificates.
- Revoke authentication credentials for applications that no longer need access to data.
- Securely store credentials.
- Use groups instead of large numbers of users.
- Bucket and object ACLs are independent of each other.
- Avoid making buckets publicly readable or publicly writable.

Always use TLS to transport your data when you can. This ensures that both your credentials and your data are protected as you transport them over the network.

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.

When applications no longer need access to your data, you should remove permissions or revoke their authentication credentials.

Make sure that you securely store your credentials.

Prefer the use of groups, instead of explicitly listing large numbers of users. Not only do groups scale better, but they also provide a very efficient way to update the access control for a large number of objects at once.

Before adding objects to a bucket, check that the default object ACLs are set to your requirements. This may save you lots 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.

Cloud Storage provides the ability to specify that objects in the bucket are publicly readable. After a bucket has been made publicly readable, data can be copied out of the bucket by anyone. It's effectively impossible to regain read control over an object written with this permission, so provide this access only when necessary. Cloud Storage buckets can also be made publicly writable. Although 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 may be legally and financially responsible for the content stored in their buckets.

Consider retention policies and retention policy locks

- Add a retention policy to a bucket to specify a retention period.
    - If no policy exists, you can delete or replace objects.
    - If a policy exists, objects can only be deleted or replaced once their age is greater than the policy.
    - Applies retroactively to existing and new objects added to the bucket.
- Lock a retention policy to permanently set it on the bucket.
    - Once set, you cannot remove or reduce the retention period.
    - A bucket cannot be deleted unless every object in the bucket has met the retention period.
    - The retention period of a locked object can be increased.
    - Locking a retention policy can help with data compliance regulations.

The Bucket Lock feature allows you to configure a data retention policy for a Cloud Storage bucket that governs how long objects in the bucket must be retained. The feature also allows you to lock the data retention policy, permanently preventing the policy from being reduced or removed. There are some things you should consider when using retention policies and retention policy locks.

Consider retention policies and retention policy locks

- **Add a retention policy to a bucket to specify a retention period.**
  - If no policy exists, you can delete or replace objects.
  - If a policy exists, objects can only be deleted or replaced once their age is greater than the policy.
  - Applies retroactively to existing and new objects added to the bucket.
- Lock a retention policy to permanently set it on the bucket.
  - Once set, you cannot remove or reduce the retention period.
  - A bucket cannot be deleted unless every object in the bucket has met the retention period.
  - The retention period of a locked object can be increased.
  - Locking a retention policy can help with data compliance regulations.

You can add a retention policy to a bucket to specify a retention period. If a bucket does not have a retention policy, you can delete or replace objects in the bucket at any time. If a bucket has a retention policy, objects in the bucket can only be deleted or replaced once their age is greater than the retention period. A retention policy retroactively applies to existing objects in the bucket as well as new objects added to the bucket.

Consider retention policies and retention policy locks

- Add a retention policy to a bucket to specify a retention period.
  - If no policy exists, you can delete or replace objects.
  - If a policy exists, objects can only be deleted or replaced once their age is greater than the policy.
  - Applies retroactively to existing and new objects added to the bucket.
- Lock a retention policy to permanently set it on the bucket.
  - Once set, you cannot remove or reduce the retention period.
  - A bucket cannot be deleted unless every object in the bucket has met the retention period.
  - The retention period of a locked object can be increased.
  - Locking a retention policy can help with data compliance regulations.

You can lock a retention policy to permanently set it on the bucket. Once you lock a retention policy, you cannot remove it or reduce the retention period it has. You cannot delete a bucket with a locked retention policy unless every object in the bucket has met the retention period. You can increase the retention period of a locked retention policy. Locking a retention policy can help your data comply with record retention regulations.

[https://cloud.google.com/security/compliance/sec]

Uniformly control access to Cloud Storage resources

- Uniform bucket-level access allows you to uniformly control access to Cloud Storage resources.

- The feature disables ACLs. Only IAM permissions grant access to the bucket and it's objects.

- Uniform bucket-level access is recommended, because it unifies and simplifies how you grant access to your Cloud Storage resources.

Cloud Storage offers two systems for granting you permission to access your buckets and objects, Identity and Access Management (IAM) and Access Control Lists (ACLs). These systems act in parallel in order for you to access a Cloud Storage resource, and only one of the systems needs to grant you permission. IAM is used throughout Google Cloud and allows you to grant a variety of permissions at the bucket and project levels. ACLs are used only by Cloud Storage and have limited permission options, but they allow you to grant permissions on a per-object basis.

Uniformly control access to Cloud Storage resources

- Uniform bucket-level access allows you to uniformly control access to Cloud Storage resources.

- The feature disables ACLs. Only IAM permissions grant access to the bucket and it's objects.

- Uniform bucket-level access is recommended, because it unifies and simplifies how you grant access to your Cloud Storage resources.

In order to support a uniform permissioning system, Cloud Storage has uniform bucket-level access.

Uniformly control access to Cloud Storage resources

- Uniform bucket-level access allows you to uniformly control access to Cloud Storage resources.

- The feature disables ACLs. Only IAM permissions grant access to the bucket and it's objects.

- Uniform bucket-level access is recommended, because it unifies and simplifies how you grant access to your Cloud Storage resources.

Access to Cloud Storage resources then is granted exclusively through IAM. After you enable uniform bucket-level access, you have 90 days to reverse your decision.

Uniformly control access to Cloud Storage resources

- Uniform bucket-level access allows you to uniformly control access to Cloud Storage resources.

- The feature disables ACLs. Only IAM permissions grant access to the bucket and it's objects.

- Uniform bucket-level access is recommended, because it unifies and simplifies how you grant access to your Cloud Storage resources.

Generally, using uniform bucket-level access is recommended, because it unifies and simplifies how you grant access to your Cloud Storage resources. Using uniform bucket-level access also enables you to use other Google Cloud security features such as Domain Restricted Sharing and IAM Conditions. However, you might not want to use uniform bucket-level access and instead retain fine-grained ACLs. For example, if you want to control access to specific objects in a bucket via legacy ACLs or you want the uploader of an object to have full control over that object, but less access to other objects in your bucket.

Consider these best practices for uploading data

- If using XMLHttpRequests:
  - Don't close and re-open the connection.
  - Set reasonably long timeouts for upload traffic.

- Make the request to create the resumable upload URL from the same region as the bucket and upload location.

- Avoid breaking transfers into smaller chunks.

- Avoid uploading content that has both:
  - `content-encoding gzip`
  - `content-type` that is `compressed`

If you use XMLHttpRequest, or XHR callbacks to get progress updates and detect when progress has stalled, do not close and re-open the connection. 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 activity from the upload stream, and closing and reopening the connection uses more network capacity at exactly the time when you can least afford it.

For upload traffic, set 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 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 Google Compute Engine instances with processes that POST to Cloud Storage to initiate a resumable upload, 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 will help 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, which improves 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,

improves throughput, and reduces QPS against Cloud Storage.

You can use decompressive transcoding, automatically decompressing a file for the requestor, by storing the file in gzip format and setting the associated metadata to Content-Encoding gzip. However, you should avoid uploading content that has both content-encoding of gzip and a content-type that is compressed. Using content encoding gzip on compressed data offers virtually no benefit.

Consider the following if using gsutil for Cloud Storage

- `gsutil -D` will include OAuth2 refresh and access tokens in the output.

- `gsutil --trace-token` will include OAuth2 tokens and the contents of any files accessed during the trace.

- Customer-supplied encryption key information in .boto config is security-sensitive.

- In a production environment, use a service account for gsutil.

gsutil is a Python application that lets you access Cloud Storage from the command line. You can use gsutil to do a wide range of bucket and object management tasks. Where applicable, you should use gcloud storage commands instead of gsutil commands.

If you run gsutil -D (to generate debugging output) it will include OAuth2 refresh and access tokens in the output. Make sure to redact this information before sending this debug output to anyone during troubleshooting/tech support interactions.

If you run gsutil --trace-token (to send a trace directly to Google), sensitive information like OAuth2 tokens and the contents of any files accessed during the trace may be included in the content of the trace.

Customer-supplied encryption key information in the .boto configuration is security sensitive. The proxy configuration information in the .boto configuration is security-sensitive, especially if your proxy setup requires user and password information. Even if your proxy setup doesn't require user and password information, the host and port number for your proxy is often considered security-sensitive. Protect access to your .boto configuration file.

If you are using gsutil from a production environment, use service account credentials instead of individual user account credentials. These credentials were designed for such use and protect you from losing access when an employee leaves your company.

For more information, see:
https://cloud.google.com/storage/docs/gsutil/addlhelp/SecurityandPrivacyConsiderations#recommended-user-precautions

Validate your data

Validate data transferred to/from bucket using:

Data can be corrupted during upload or download by:

- Noisy network links

- Memory errors on:
  - Client computer
  - Server computer
  - Routers along the path

- Software bugs

- CRC32c Hash
  - Is available for all cloud storage objects
  - Can be computed using these libraries:
    - Boost for C++
    - crcmod for Python
    - digest-crc for Ruby
  - `gcloud storage` automatically performs integrity checks on all uploads and downloads

- MD5 Hash
  - Is supported for non-composite objects
  - Cannot be used for partial downloads

Data can be corrupted while it is uploaded to or downloaded from the cloud by noisy network links, memory errors on client or server computers or routers along the path, and software bugs. Validate the data you transfer to/from buckets using either CRC32c or MD5 checksums.

Cloud Storage supports server-side validation for uploads, but client-side validation is also a common approach. If your application has already computed the object's MD5 or CRC32c before starting the upload, you can supply it with the upload request, and Cloud Storage will create the object only if the hash you provided matches the value Google calculated. Alternatively, users can perform client-side validation by issuing a request for the new object's metadata, comparing the reported hash value, and deleting the object in case of a mismatch.

All Cloud Storage objects have a CRC32c hash. Libraries for computing CRC32c include Boost for C++, crcmod for Python, and digest-crc for Ruby. Java users can find an implementation of the algorithm in the [GoogleCloudPlatform crc32 Java project](). `gcloud storage` automatically performs integrity checks on all uploads and downloads. Additionally, you can use the "`gcloud storage hash`" command to calculate a CRC for any local file.

MD5 hashes are supported for non-composite objects. The MD5 hash only applies to a complete object so it cannot be used to integrity check partial downloads cause by performing a range GET.

For more information about hashes and e-tags:
https://cloud.google.com/storage/docs/hashes-etags
https://cloud.google.com/sdk/gcloud/reference/storage/hash

You can host static websites

You can allow scripts hosted on other websites to access static resources stored in a Cloud Storage bucket.

You can also allow scripts hosted in Cloud Storage to access static resources hosted on a website external to Cloud Storage.

The Cross-Origin Resource Sharing, or CORS topic describes how to allow scripts hosted on other websites to access static resources stored in a Cloud Storage bucket.

You can also allow scripts hosted in Cloud Storage to access static resources hosted on a website external to Cloud Storage. The website must serve CORS headers so that content on storage.googleapis.com is allowed access. It is recommended that you dedicate a specific bucket for this type of data access.

Lab

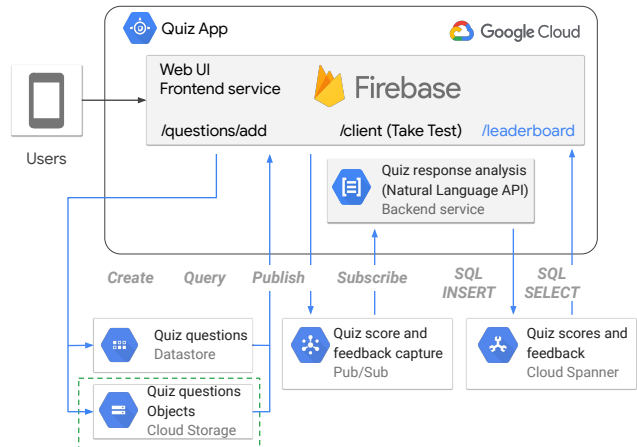## Storing Image and Video Files in Cloud Storage

Duration: 45 minutes

Lab objectives

Create a Cloud Storage bucket

Review file upload UI and code changes

Write code to store upload file data in Cloud Storage

Google Cloud

Cloud Storage is the ideal solution for file storage in the Cloud. Storage classes and object lifecycle management enable you to optimally store and archive data depending on frequency of access. You can compose large objects using smaller fragments that are uploaded separately. With this approach, you don't have to worry about intermittent network failures during uploads. You can also leverage parallel uploads to speed up the upload process. Cloud Storage enables you to secure and so rich content such as audio and video with high availability and throughput. You can use Cloud Storage as a key component of your data analytics pipelines and machine learning applications.