Chapter 7. Storage

This chapter is for storage solutions in GCP that are not considered as databases (databases are covered in another chapter).

7.1. Cloud Storage

Cloud Storage is GCP's object storage solution.

7.1.1. Key Concepts for Cloud Storage

Buckets

A **bucket** is the basic container that holds objects. Every object in Cloud Storage belongs to a bucket.

Bucket names must be globally-unique, because every bucket resides in a *single Cloud Storage namespace*. Even if you create buckets in separate projects, they cannot have the same name.

Objects

Objects are the individual pieces of data stored within buckets. There is no limit to the number of objects you can create in a bucket.

An object in Cloud Storage can have different versions (when versioning is enabled). By default, overwriting a versioned object will delete the old version and replace it with a new version.

Bucket locations

There are **three** different bucket locations:

- **Region**, i.e. a specific geographic place, such as London.
- **Dual-region**, i.e. a specific *pair* of regions (separated by at least 100 miles), such as Finland and the Netherlands.
- **Multi-region**, i.e. a large geographic area, such as the United States, that contains two or more geographic places (including data centers not explicitly listed as a region).

Objects stored in a **multi-region** or **dual-region** are ***geo-redundant***, which means that in the event of a large-scale disruption (that makes a region unavailable), your objects remain accessible (from other unaffected locations). However, more availability generally means a marginally higher price.

Storage classes

Each object in Cloud Storage belongs to a specific **storage class**. When you create a bucket, you specify a *default* storage class for the bucket. Objects added to the bucket will inherit this

storage class unless explicitly set otherwise. This means that a bucket can hold objects of various storage classes.

Storage Class	Minimum storage duration	Availability SLA	Use case
Standard	None	>= 99.9%	Hot, frequently accessed data
Nearline	30 days	>= 99.0%	Infrequently accessed data, lower cost
Coldline	90 days	>= 99.0%	Very infrequently accessed data, very low cost
Archive	365 days	None	Archive, backup, and disaster recovery data.

Regardless of the storage class, the following aspects apply:

- **Unlimited storage** with **no minimum object size**
- **Low latency** (time to first byte typically tens of **milliseconds**, unlike AWS S3 Glacier which takes hours)
- High durability (**99.999999999% annual durability**)

  If you store 10 million objects, then you expect to lose an object of your data every 10,000 years. However, if an asteroid hits your bucket in a single-region, this number won't save you. Geo-redundancy helps.

- Geo-redundancy if the data is stored in a multi-region or dual-region.

7.2. Key Features of Cloud Storage

7.2.1. Object Lifecycle Management

You can define a set of rules for each bucket. The rules will apply to current and future objects in the bucket. Each rule contains only **one action**, triggered by **one or more conditions**. When an object meets the criteria of one of the rules, Cloud Storage automatically performs specified action on the object.

Lifecyle actions

The following actions are supported for a lifecycle rule:

- **`Delete`**: Deletes objects.

- **`SetStorageClass`**: Changes the storage class of objects.

Action precedence

Should multiple rules have their conditions simultaneously satisfied for a single object, Cloud Storage performs the action with only *one* of the rules:

- **`Delete`** action takes precedence over any `SetStorageClass` action.
- **`SetStorageClass`** action that switches the object to the storage class with the lowest storage pricing takes precendence.

Lifecyle conditions

Among the conditions that a lifecycle rule supports are:

- **`Age`**: Object has reached the specified age (in days).
- **`CreatedBefore`**: Object is created before midnight of the specified date in UTC.
- **`IsLive`**: Used in conjunction with object versioning, set to `true` or `false` to select latest live version or noncurrent version of an object respectively. If not using object versioning, all objects are considerd live.
- **`MatchesStorageClass`**: Objects that match the specified storage class.
- **`NumberOfNewerVersions`**: Used in conjunction with object versioning, object version satisfies the condition when there are at least N versions newer than it. Not used when object versioning is disabled (the default).

There are a few other omitted conditions surrounding **`Time`** (less important). The names of the conditions are not important for the exam, but you need to know what types of conditions are supported (the ones listed above).

7.2.2. Requester Pays

If the object requester provides a *billing project* with their request, the requester's project is billed instead. With Requester Pays enabled on your bucket, you can *require* requesters to include a billing project in their requests.

7.2.3. Retention policies

- 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 overwrite objects in the bucket at any time.
  - If a bucket has a retention policy, objects in the bucket can only be deleted or overwritten 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.
- 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.
- You can also place the following **holds** on individual objects which prevent them from being deleted or overwritten:
    - **Event-based holds** are used to control retention based on the occurence of some event. Once released, it resets the object's time in the bucket for the purposes of the retention period.
    - **Temporary holds** are used for regulatory or legal investigation purposes. Once released, it does not affect the object's time in the bucket with regard to retention period.

  As long as there is a **hold** on the object, that object will not be deleted until the hold is released, even if the retention period has passed. After a hold is released, the time the object has spent in the bucket (with regard to retention period) is either reset or unchanged. See example.

While the exam may not dig into retention policies much, I think as an (enterprise) Cloud Architect this would be useful to be aware of.

7.2.4. Data Encryption in Cloud Storage

Cloud Storage always encrypts your data on the server side, before it is written to disk, at no additional charge. You cannot disable this behavior, i.e. you cannot store an object in Cloud Storage unencrypted.

The following data encryption options are available in Cloud Storage (and Google Cloud in general):

- **Server-side encryption**, encryption that occurs after Cloud Storage receives your data, but before the data is written to disk and stored:
    - **Google-managed encryption keys**. Described above, this is the default behavior.
    - **Customer-supplied encryption keys** (**CSEK**). Create and manage your own encryption keys for server-side encryption.

      This acts as an additional encryption later on top of the standard encryption behavior. You

will provide a key with which Cloud Storage will use for each Cloud Storage operation. This key is purged from Google's servers after the operation is complete.

When using `gsutil`, the encryption and decryption key is provided in `~/.boto`, which is the default configuration file for `gsutil`.

~/.boto

```
encryption_key = [YOUR_ENCRYPTION_KEY]

# yes, there is a 1 after _key
decryption_key1 = [YOUR_ENCRYPTION_KEY]
```

- **Customer-managed encryption keys** (**CMEK**). Generate and manage your encryption keys using Cloud Key Management Service (KMS). Unlike CSEK where the customer provides (and stores) the encryption key, Cloud KMS will generate and store the encryption keys instead.

- **Client-side encryption**, where encryption is done before data is sent to Cloud Storage, where it will undergo another round of standard encryption.

7.2.5. Cloud Storage Quotas and Limits

Key quotas and limits you should be aware of:

- Maximum **5 TB** for invididual objects stored in Cloud Storage.
- Per-project rate limit to bucket creation and deletion of ~ 1 operation every 2 seconds; plan on *fewer buckets* with *more objects*. Plan for multiple users to share a bucket rather than one bucket for one each user.

7.2.6. Cloud Storage Best Practices

Some best practices when using Cloud Storage are the following:

- If get error codes when making requests to Cloud Storage, retry with **exponential backoff**.
- Design application to minimize spikes in traffic. Spread traffic throughout the day.
- Store data in a region closest to your application's users.
- Choose bucket names that are difficult to guess.
- Use object hold on individual objects to prevent deleting or overwriting until the hold is removed.
- Use object versioning to retain older version of objects when they are deleted or overwritten. Use Object Lifecycle Management to delete older versions.

7.3. Cloud Filestore

Cloud Filestore (not to be confused with Cloud Fi**r**estore) is a managed file storage service for applications that require a filesystem interface and a shared filesystem for data. Filestore gives users a simple and managed **Network Attached Storage** (**NAS**) with their Compute Engine and Kubernetes Engine instances.