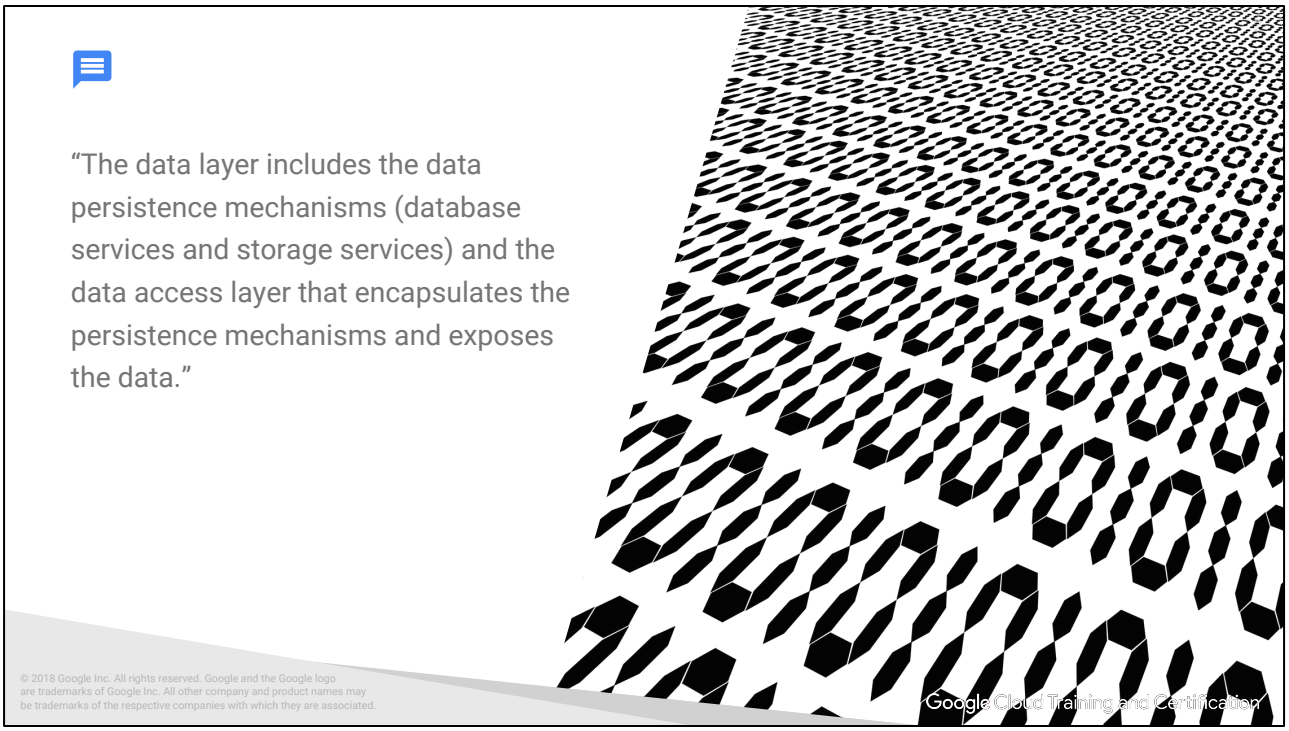
Data Layer Design

Architecting with Google Cloud Platform: Design and Process

Last modified 2018-08-08



https://pixabay.com/en/binary-digitization-null-one-pay-2007350/
Adapted from:
https://en.wikipedia.org/wiki/Multitier_architecture#Three-tier_architecture

Agenda

Classifying and characterizing data

Data ingest and data migration

Identification of storage needs and mapping to storage systems

Photo service is having Intermittent outages

Design challenge #2: Complication

No GCP lab in this module

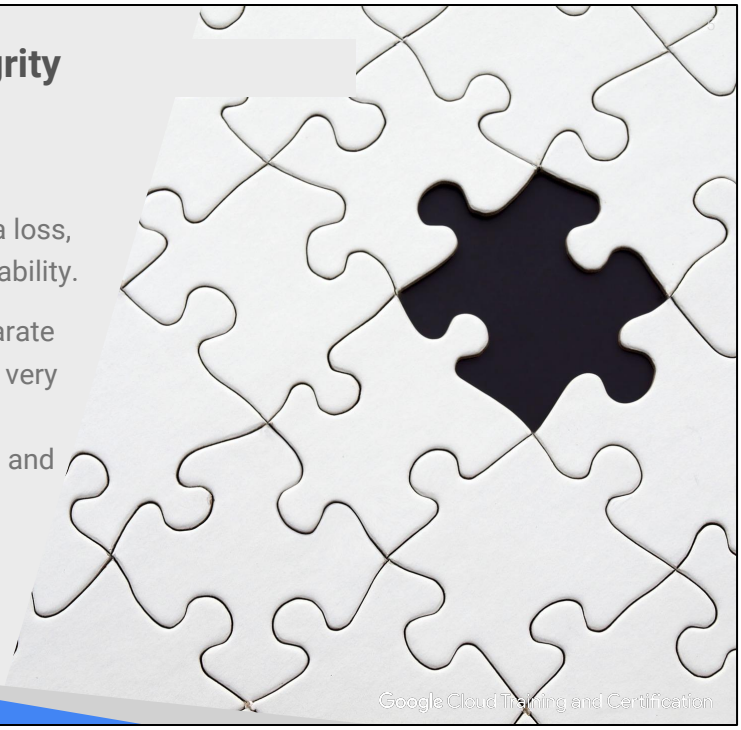
Classifying and characterizing data

Users care about **data integrity**

Every service has separate data requirements

Users do not distinguish between data loss, data corruption, and extended unavailability.

- Persistence and Access are separate
- Access - a loss of data access is very important to users
- Persistence - proactive detection and rapid recovery



In general an SLO of 99.99% uptime (access), which is only 1 hour of downtime per year, exceeds most Internet and Enterprise users expectations.
If an item of data is corrupted, and it goes out of service for 30 minutes, during which time it is fixed and returned to service, it has 99.99% availability during the year.
https://pixabay.com/en/puzzle-match-fit-missing-hole-693865/

What **data transaction properties** are required?

ACID

- Atomicity
- **Consistency**
- Isolation
- Durability

BASE

- Basically **Available**
- Soft state
- Eventual consistency

CAP Theorem

Database services provide a model of consistency. Consistency makes certain guarantees with respect to data transactions. And whatever guarantees are not made by the data service become the responsibility of the application code.

ACID - SQL databases such as MySQL and PostgreSQL.
BASE - NoSQL systems like Bigtable

https://en.wikipedia.org/wiki/Consistency_(database_systems)
https://en.wikipedia.org/wiki/ACID
Atomicity - A transaction is either "all or nothing"
Consistency - Any transaction brings the database from one valid state to another.
Isolation - Transactions executed concurrently produce the same result as if executed sequentially.
Durability - Once a transaction is committed, the results are stable. Even in the event of a power loss the result is non-volatile.

BASE is described under eventual consistency in Wikipedia:
https://en.wikipedia.org/wiki/Eventual_consistency
An eventually consistent system does not have Atomic transactions. So once a transaction has started to be committed, there are no guarantees until all the parts of the system have converged. This means that users requesting data may get "any" result (ie there are no guarantees), but in practice it means the user gets stale data. That is a guarantee that they will get SOME data, some result, but not necessarily the most current result.

https://en.wikipedia.org/wiki/CAP_theorem
In database systems, ACID (most SQL systems) optimize for consistency and BASE (most NoSQL systems) optimize for availability.

https://pixabay.com/en/menu-yemekservisi-akula-1197654/

What are the **data consistency** requirements?

Cloud Storage global strong consistency

- Read-after-write
- Read-after-metadata-update
- Read-after-delete
- Bucket listing
- Object listing
- Granting access to resources

Cloud Storage Eventually consistent

- Revoking access (~1m)
- Enabling object versioning in a bucket wait ~30 sec before write/overwrite

A cached object may not show strong consistency

- Control the degree of consistency of objects in a cache
- by setting metadata and cache lifetime

https://cloud.google.com/storage/docs/consistency

"**Eventual consistency** is a **consistency** model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value."

https://en.wikipedia.org/wiki/Eventual_consistency

Cloud Storage provides read after write consistency, read after update consistency.

Example of Cloud Storage eventually consistent operations:
Revoking access: It typically takes about a minute to take effect. In some cases it may take longer.
If you remove a user's access to a bucket, the metadata will reflect the change immediately, but the user may still have access to the bucket for a short period of time.

https://cloud.google.com/storage/docs/consistency

What are you trying to optimize?

The data strategy you choose for a service depends on which requirements you are trying to optimize.

Uptime	The proportion of time that a service is unavailable to users. Availability.
Latency	How responsive a service appears to be to its users.
Scale	The volume of users and mix of workloads the service can tolerate before latency suffers or the service begins to fail.
Velocity	How fast a service can innovate to provide superior value at reasonable cost.
Privacy	Data must be destroyed within a reasonable time after a user deletes it.

Data ingest and data migration

Cloud Storage tools for data migration

Console (drag-and-drop) interface

- Simulated folder icons for familiarity

gsutil

- Resumable upload and download
- Streaming transfer
  - Chunked transfer encoding

Control access with Cloud IAM, Signed URLs, and ACLs

JSON API

- gzip
  - Reduce network bandwidth at the expense of CPU
  - Transcoding of gzip compressed files
- Partial resource request/reply

Cloud Storage has many features. These were covered in the Architecting GCP: Infrastructure class and are in the online documentation.

Console is easy to use for a one-time manual event. gsutil is feature-rich and can be incorporated into scripts, cron jobs, etc. JSON API enables integration with other tools such as compression and partial resource request/reply (access to specific fields in the data) so you don't have to transfer the whole object to get a tiny part of it.

https://cloud.google.com/storage/docs/quickstart-gsutil
https://cloud.google.com/storage/docs/json_api/
https://cloud.google.com/storage/docs/cloud-console#_uploadingdata

Resumable: Important for large files because the likelihood of a network drop increases with the size of the data. Resume is based on bytes already received to avoid resending.
gzip transcoding: Change the encoding of a file before serving it.
https://cloud.google.com/storage/docs/transcoding

https://cloud.google.com/storage/docs/streaming
gsutil tool or boto library provides streaming data based on HTTP chunked transfer encoding. Lets you stream data as soon as it becomes available without requiring that the data be first saved to a separate file. RFC 7230:
https://tools.ietf.org/html/rfc7230#section-4.1

Transcoding: https://cloud.google.com/storage/docs/transcoding

Cloud Storage Transfer Service

Import online data to Cloud Storage

- Amazon S3
- HTTP/HTTPS Location
- Transfer data between Cloud Storage buckets

Backup data to Cloud Storage

- Move from Multi-Regional to Nearline

Synchronize

- One time, recurring, import at time of day
- delete objects not in source
- delete source objects after transfer
- filter on file-name, creation date

When transferring data from an on-premise location, use gsutil. OR RESTful API
When transferring data from another cloud storage provider, use Storage Transfer Service.
Otherwise, evaluate both tools with respect to your specific scenario.

Multiple-content encoding: https://tools.ietf.org/html/rfc7231#section-5.3.4

https://cloud.google.com/storage/transfer/
https://cloud.google.com/data-transfer/docs/introduction

Google Transfer Appliance

Rackable device up to 1PB ship to Google.

Use Transfer Appliance if your dataset meets the following conditions:

- If it would take more than one week to upload your data.
- If you have 60 TB or more data, regardless of the connection speed.

Google Transfer Appliance (Beta)
Transfer Appliance comes in two sizes, 100 terabytes (TB) and 480 TB. The 100 TB appliance can store from 100 TB up to potentially 200 TB, depending on the deduplication and compression ratio of your data. Similarly, the 480 TB appliance can store from 480 TB up to potentially 1 petabyte (PB). Both rackable and standalone appliances are available.

Photo from Google Blog:
https://cloudplatform.googleblog.com/2017/07/introducing-Transfer-Appliance-Sneakernet-for-the-cloud-era.html

Online transfer > 7 days - use Transfer Appliance

	1 Mbps	10 Mbps	100 Mbps	1 Gbps	10 Gbps	100 Gbps
1 GB	3 hrs	18 mins	2 mins	11 secs	1 sec	0.1 sec
10 GB	30 hrs	3 hrs	18 mins	2 mins	11 secs	1 sec
100 GB	12 days	30 hrs	3 hrs	18 mins	2 mins	11 secs
1 TB	124 days	12 days	30 hrs	3 hrs	18 mins	2 mins
10 TB	3 years	124 days	12 days	30 hrs	3 hrs	18 mins
100 TB	34 years	3 years	124 days	12 days	30 hrs	3 hrs
1 PB	340 years	34 years	3 years	124 days	12 days	30 hrs
10 PB	3,404 years	340 years	34 years	3 years	124 days	12 days
100 PB	34,048 years	3,404 years	340 years	34 years	3 years	124 days

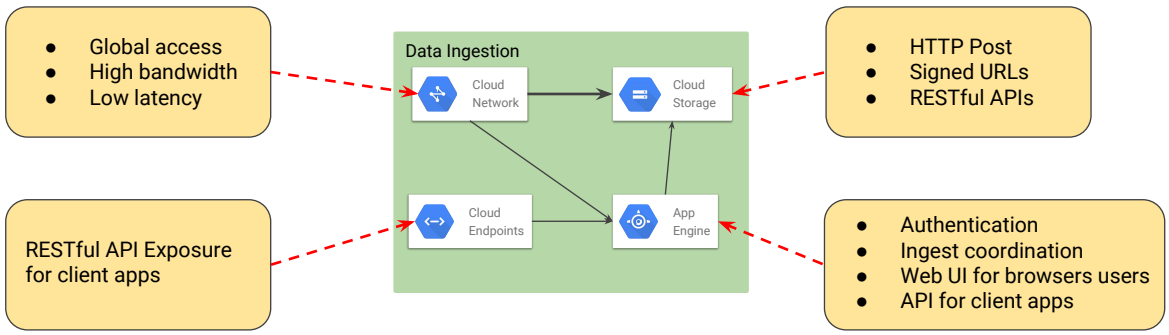
Use Google Transfer Appliance

The left side of the table are closer to physical speeds, the right side of the table is closer to online speeds.
Therefore, it is much faster to accumulate data online and work with it and transfer it online than to collect data physically and then transfer it.

https://cloud.google.com/data-transfer/

Ingesting data into your service

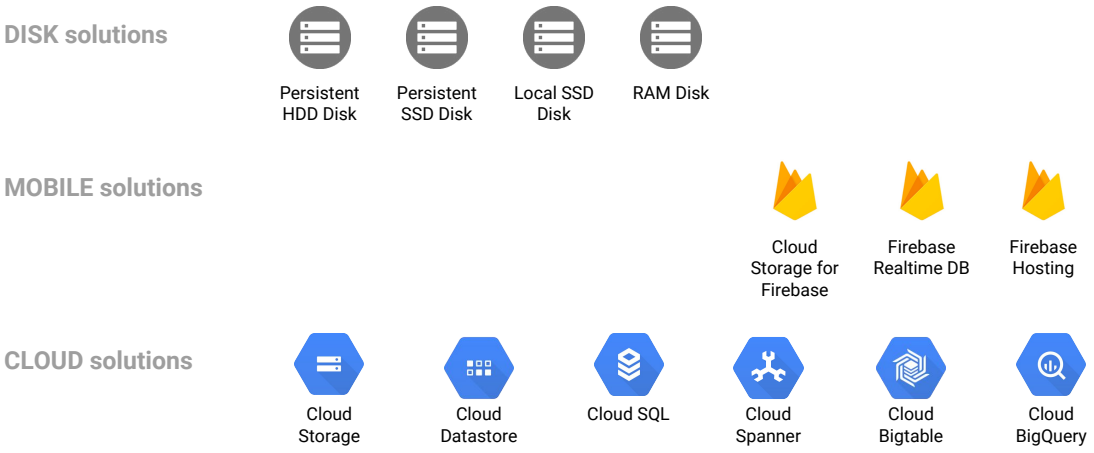
These are the key components for ingesting data into GCP and the features they contribute to data ingestion.



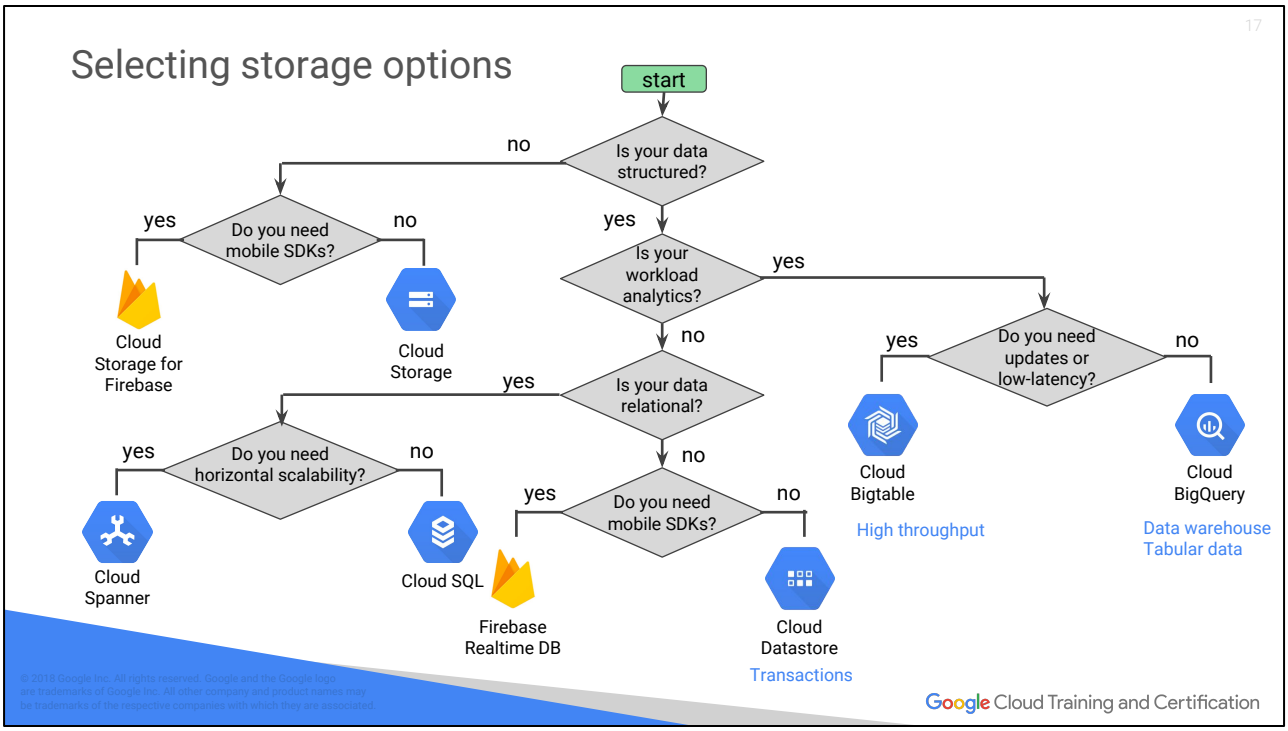
Global Google Cloud Network integration with Google Cloud Storage enables high bandwidth upload from anywhere.
App Engine provides authentication and ingest coordination and management between the network and storage.
Upload is exposed to browsers through a Web UI implemented on App Engine. And upload is provided for client applications via RESTful APIs exposed by Cloud Endpoints.

Identification of storage needs and mapping to storage systems

GCP offers many storage options.



There are storage reference slides following the last slide in this module so that you can cover in as much or as little detail as you choose.



BigQuery is recommended as a data warehouse.
BigQuery is the default storage for tabular data.
Use Datastore if you need transactions.
Use Bigtable if you want low-latency/high-throughput.

Choosing Cloud Storage

	Regional	Multi-Regional	Nearline	Coldline
Design Patterns	**Data that is used in one region** or needs to remain in region	**Data that is used globally** and has no regional restrictions	**Backups** Data that is accessed no more than once a month	Archival or **Disaster Recovery** (DR) data that is accessed once a year or less often
Feature	Regional	Geo-redundant	Backup	Archived or DR
Availability	99.9%	99.95%	99.0%	99.0%
Durability	99.9999999999%	99.9999999999%	99.9999999999%	99.9999999999%
Duration	Hot data	Hot data	30 day minimum	90 day minimum
Retrieval cost	none	none	$	$$

Cloud storage options fit different use cases and designs.

Another reason to choose Regional storage is if you want to ensure that your data stays within a specific geographic or political region, to comply with legal requirements, for example.

Cloud Storage can be a solution in itself. For example, Cloud Storage is often used for general storage or as backup with no other components needed.

- Virtually unlimited capacity
- Durable - redundant storage and transparent recovery
- Globally accessible
- App Engine, Compute Engine, Kubernetes Engine
- Externally accessible

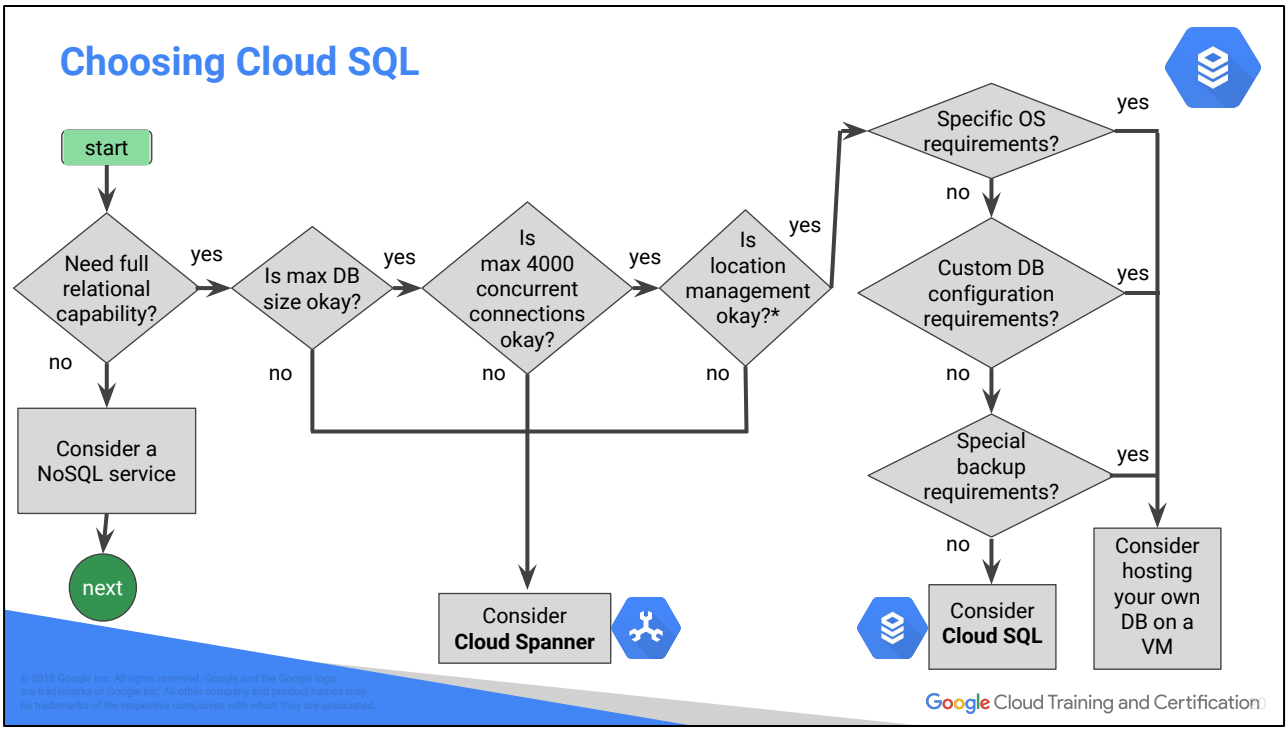
Cloud Storage pricing: https://cloud.google.com/storage/#pricing

Choosing BigQuery

Enterprise Data Warehouse

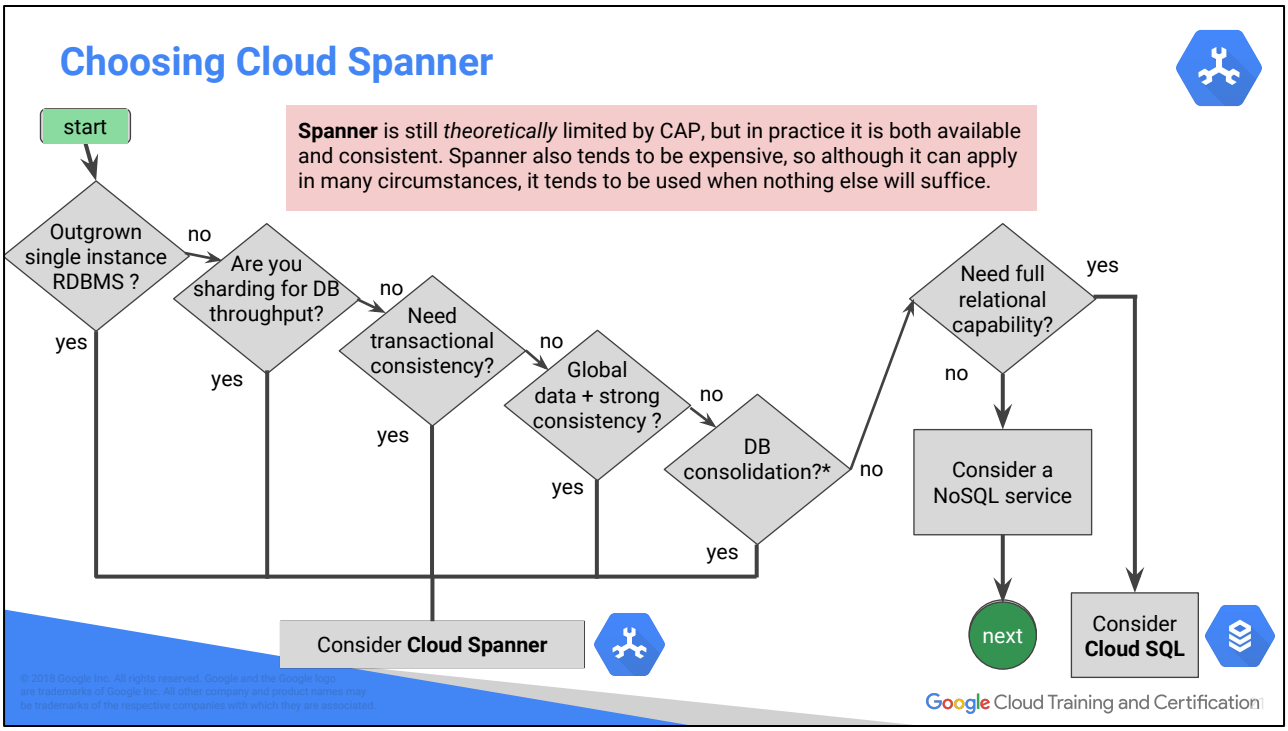
- OLAP workloads up to petabyte-scale
- Big Data exploration and processing
- Reporting via Business Intelligence (BI) tools
- Analytical reporting on large data
- Data Science and advanced analyses
- Big Data processing using SQL with fast response times

In many cases, BigQuery is the storage solution of choice due to analytics requirements. BigQuery provides a general data warehouse solution. This might serve as the core storage service and then specific applications may use other storage services for additional capabilities or qualities. For example, transactional requirements might be solved with Datastore, and high-throughput and low latency requirements might be handled with Bigtable, with the data originating from or destined for BigQuery.



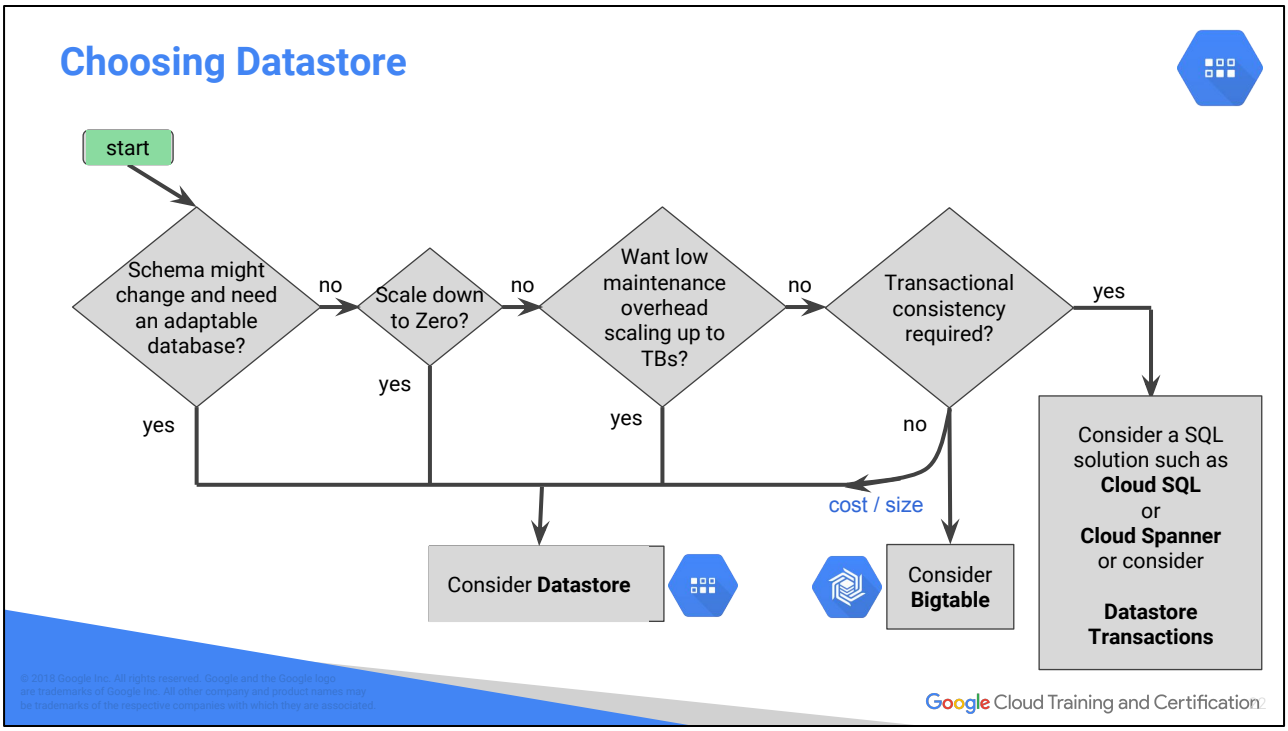
Cloud SQL First Generation has a maximum database size of 250 GB and supports MySQL 5.5 and 5.6.
If you need version compatibility and your application won't surpass the 250 GB size, you can consider Cloud SQL First Generation.

*Note that you can get multi-regional capability from MySQL by creating MySQL replicas. Cloud Spanner was built with global scale in mind. The real question here is, if you are scaling up globally, do you want your application design to be responsible for scaling, availability and location management? Or would you like to delegate those complicated problems to a service? Cloud Spanner allows you to expand horizontally globally, and the service handles the many of the details.



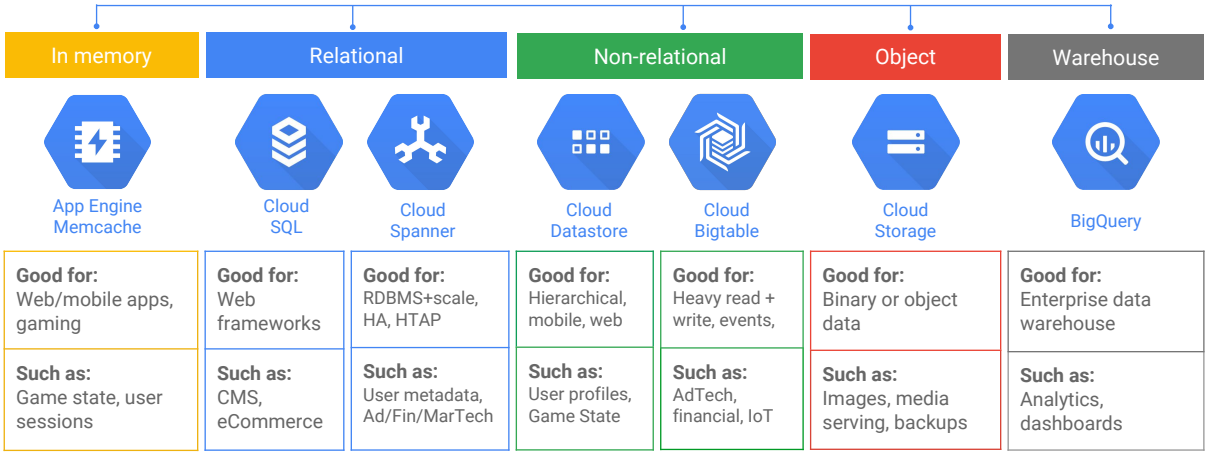
Some companies evolve complex database applications targeted to specific customer needs. The maintenance overhead for managing the multiple systems may be significant and variable. Cloud Spanner may be a candidate to consolidate the systems into a single managed service.

http://www.infoworld.com/article/3209719/cloud-computing/review-google-cloud-spanner-takes-sql-to-nosql-scale.html

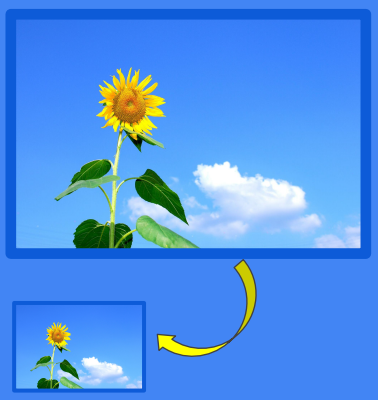


Datastore transactions. Is an optional way to use datastore. You group datastore operations into a transaction, and either all of them are applied or none of them are applied, which means that the entire transaction is atomic. If any datastore statement in the group raises and exception, the entire transaction is scrubbed and an error is returned. There are timing considerations. Transactions have a maximum duration of 60 seconds with a 10 second idle expiration time after 30 seconds. You can read more about it in the online documentation: https://cloud.google.com/datastore/docs/concepts/transactions

Storage & Database Portfolio



Intermittent outages



Occasionally, the service fails to produce the required results. It appears to be random.

The system is failing to generate some thumbnail images

The thumbnail service is growing in popularity and number of users.

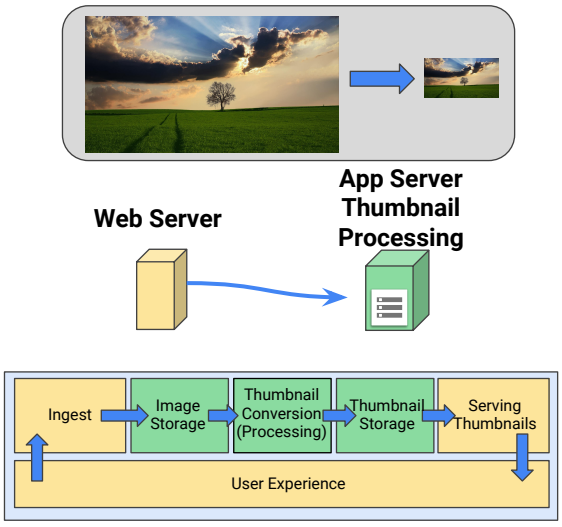
However, there are increasing reports that sometimes the service fails to produce the thumbnail image.

The issue appears to be random.

After systematic and logical troubleshooting, and answering the "five why's", the team determines that the root problem is that the persistent disk on the application server cannot keep up with the scale of demands being placed on it.

When it can't keep up, it is sometimes randomly dropping transactions.

Refresher



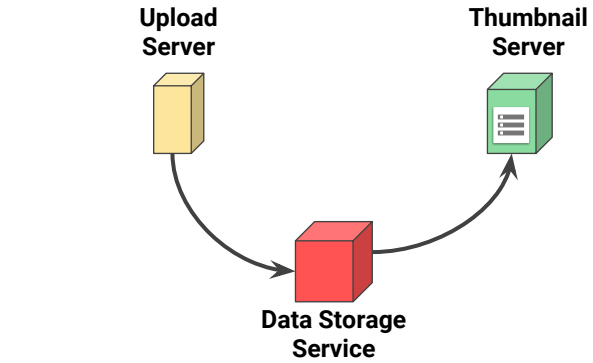
https://pixabay.com/en/countryside-tree-landscape-sunlight-2175353/

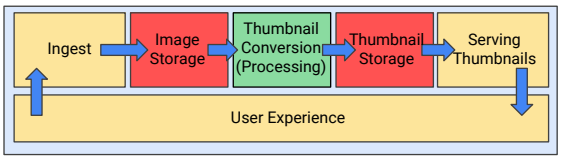
Migrate to an object store for a more scalable service.

Business Issue: Persistent disk is causing scaling issues. File system cannot handle millions* of images.

- Decouple storage for unlimited scale
- Rewrite code to use API calls vs file system lookups
- Have to compromise on read/write latency due to externalization of files

**1 million images per day*.





Objectives and Indicators

Objectives	Indicators
Availability, 23/24 hours/day = 95.83% availability	Aggregate server up/down time
99% of user operations completed in < 1 minute	End to end latency
Failure to produce a thumbnail < 0.01% (100 errors per million)	Completion errors (log entry) @ 1m images/day **Error budget** = 3,000 errors per month

User pain is experienced when the service fails to produce a thumbnail. This can be measured because the application generates a completion log entry after the thumbnail is generated and stored to Data Storage. If, for any reason, the thumbnail is either not generated or not stored an error message is output to the log.
.01% of 1,000,000 = 10,000 = 100 errors per million.

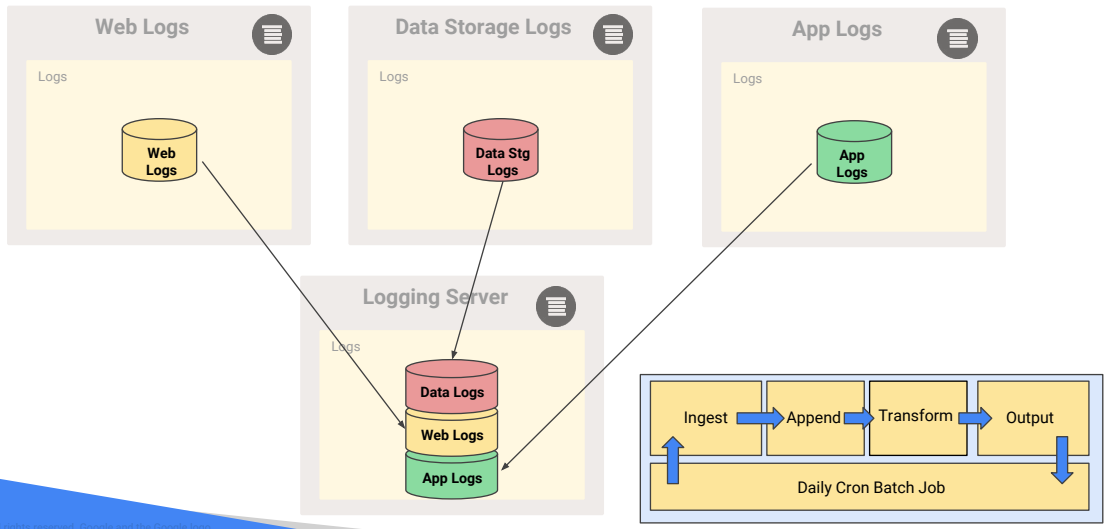
YOUR TURN

Design challenge #2

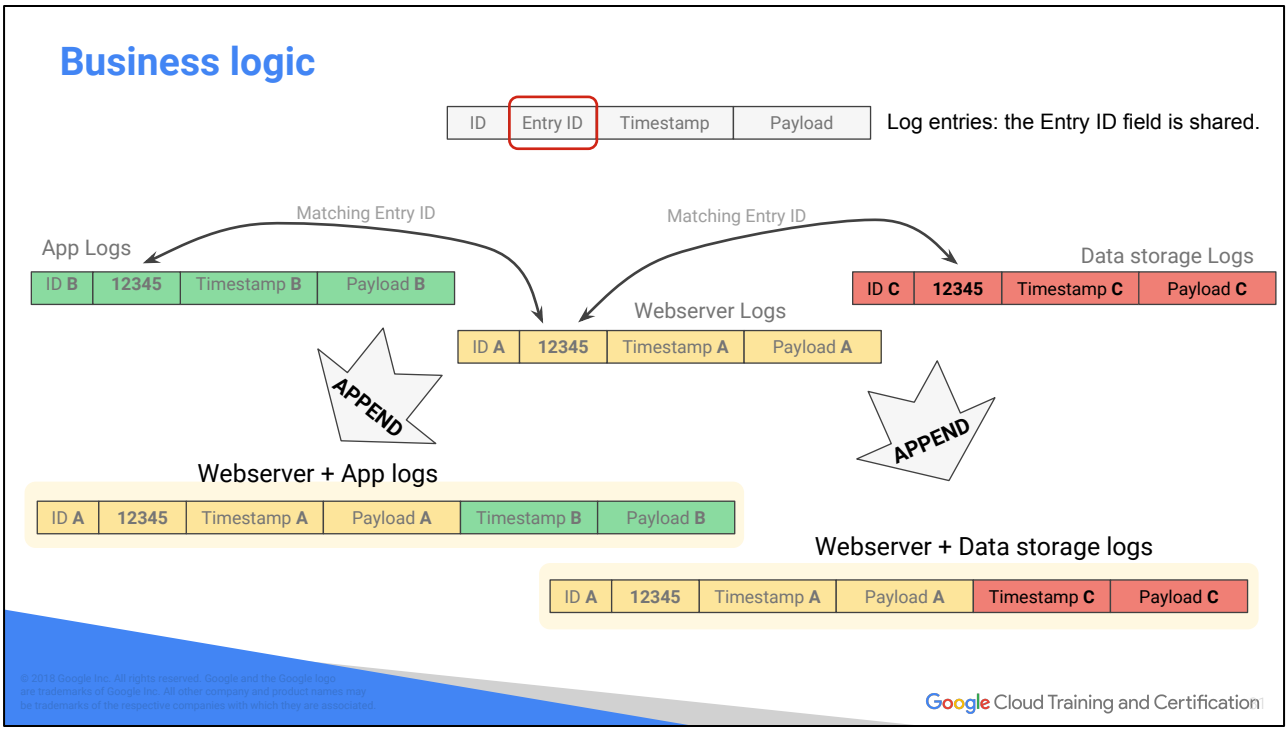
Complication

https://pixabay.com/en/the-strategy-win-champion-1080527/

Another log source; the data storage system



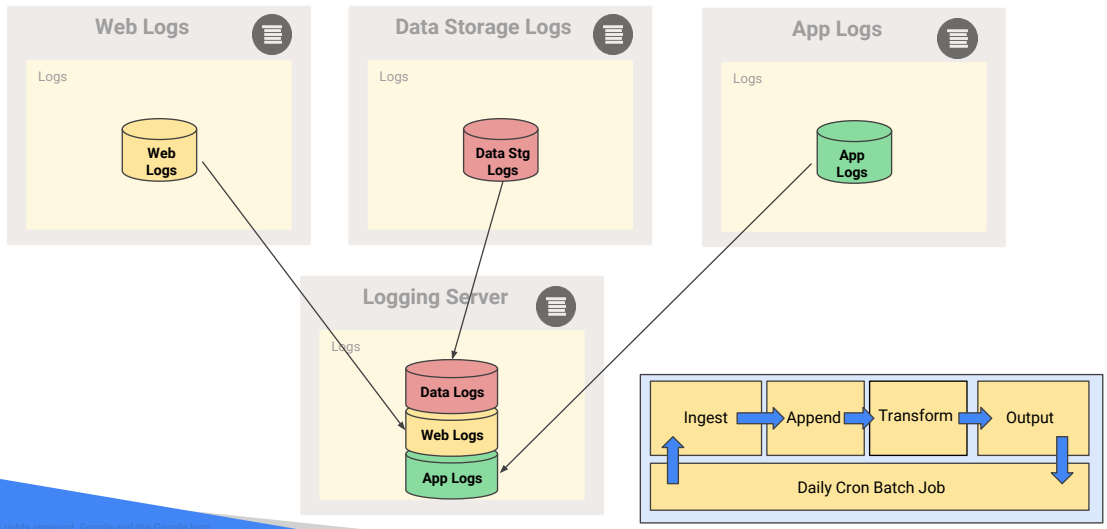
If the application might quickly outgrow local disk, what is a better way to store the log data that will scale to additional log ingest streams?



3 types of log entries: A, B, and the new Data storage logs, C; each log entry contains Entry ID, timestamp and payload.
Design a system that appends log entries of type A+B and A+C, based on a shared Entry ID.

Each entry has EntryID, timestamp, and some entry-specific payload.
Your mission is to append logs of Logtype A and B, and Logtype B and C.

Logging server outgrows disk. What storage service?



If you expect to quickly outgrow local disk, what is a better way to store the log data that will scale to additional log ingest streams?

Take a few minutes to design your solution.

Problem: The logs that are inputs into the aggregation logging server have outgrown the capacity of the server disk. Design a solution.

There are multiple designs possible depending on your assumptions. Your solution may be better than the one shown. The point of this exercise is to "think about the design" to develop your architecting skills.

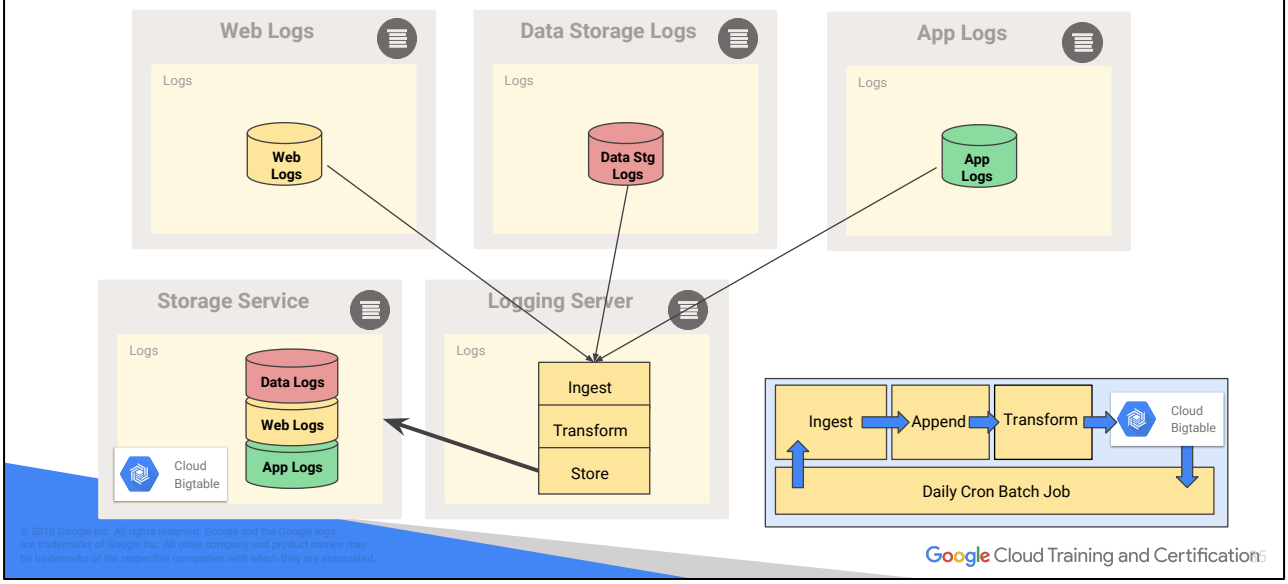
You can sketch your design in a tool like http://docs.google.com/drawings

One solution

Remember, there are multiple valid solutions to this challenge.

Compare your design with the example solution.
Did your design account for all the elements addressed in the example solution?

Logging server outgrows disk. What storage service?



In this proposed design, the logging server stores transformed log records in Cloud Bigtable.

Google Cloud Training and Certification

Appendix:
Storage reference details

Storage systems

Persistent Disk	Managed block storage for Compute Engine virtual machines and Kubernetes Engine containers, and Snapshots for data backup
Cloud Storage	Object / blob store suitable for unstructured data such as Images, pictures, videos and objects
Bigtable	NoSQL wide-column database for real-time access and analytics workloads. Low-latency read/write access, High-throughput analytics, Native time series support
Datastore	NoSQL document database for web and mobile applications. Used for semi-structured application data, hierarchical data, and durable key-value data
Cloud SQL	MySQL and PostgreSQL database service for web frameworks, structured data and OLTP workloads.
Cloud Spanner	Relational database service with transactional consistency, global scale and high availability for mission-critical applications, and high transaction applications
BigQuery	Enterprise Data Warehouse (EDW) with SQL and fast response times for OLAP workloads up to petabyte-scale, Big Data exploration and processing, and reporting via Business Intelligence (BI) tools
Google Drive	A collaborative space for storing, sharing, and editing documents and files, and syncing between cloud and local devices

https://cloud.google.com/storage-options/

Mobile storage systems (Firebase)

Cloud Storage for Firebase	Mobile and web access to Google Cloud Storage with serverless third party authentication and authorization. Images, pictures, and videos, Objects and blobs, Unstructured data
Firebase Realtime Database	A realtime, NoSQL JSON database for your web and mobile applications. Mobile and web applications, Realtime
Firebase Hosting	Production-grade web and mobile content hosting for developers. Atomic release management, JS app support (for example, URL rewriting), Firebase integration

https://cloud.google.com/storage-options/

Disk options

	Persistent disk HDD	Persistent disk SSD	Local SSD disk	RAM disk
Data redundancy	Yes	Yes	No	No
Encryption at rest	Yes	Yes	Yes	N/A
Snapshotting	Yes	Yes	No	No
Bootable	Yes	Yes	No	Not
Use case	General, bulk file storage	Very random IOPS	High IOPS + low latency	low latency + risk of data loss

STORAGE REFERENCE INFO

Detailed differentiation

	Cloud Storage	Cloud SQL	Cloud Spanner	Datastore	Bigtable	BigQuery
Capacity	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
Access metaphor	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
Read	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
Write	One file	INSERT row		put object	put row	Batch/stream
Update granularity	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
Usage	Store blobs	No-ops SQL database on the cloud		Structured data from AppEngine apps	No-ops, high throughput, scalable, flattened data	Interactive SQL* querying fully managed warehouse

https://cloud.google.com/storage-options/

Database sharding

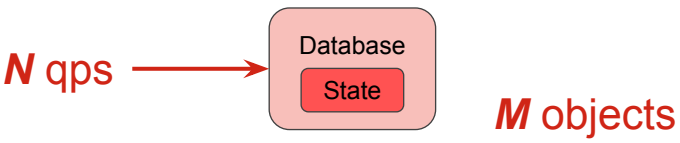
Database horizontal scaling and redundancy

It is built-in to several database services including Cloud Spanner

Database: Single state

Single server worst case

Single machine failure = 100% unavailability



Horizontally scale stateful servers?

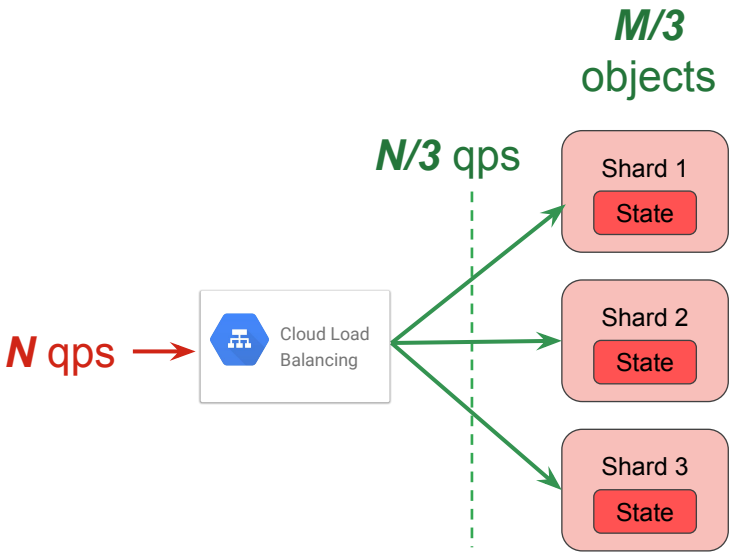
- Capacity
- Failure domains
- How to distribute

The idea is that by dividing state, it might be possible to reduce the issues associated with keeping state.
So instead of scaling vertically (bigger machine, more reliable machine) try to scale horizontally.
That brings up some considerations, like how much capacity is needed, how to establish failure domains, and how to distribute the stateful work

Horizontal scaling with shards

Dividing state by sharding

- Each piece of data gets unique ID
- Ranges of unique IDs grouped into shards
- Every shard hosted by single single server
- Balancer knows a mapping from shard range to server, forwards request

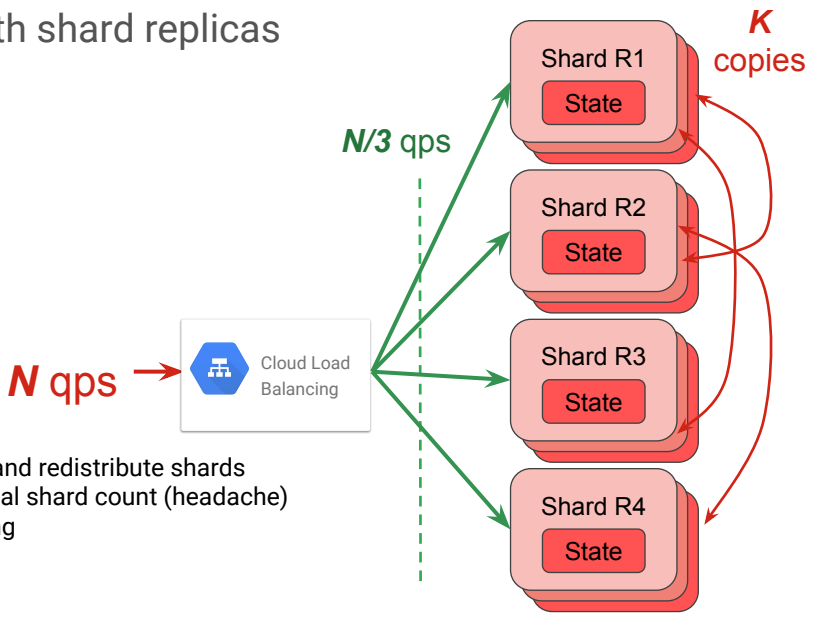


Here is an example of dividing state using horizontal scaling. In this case the state is distributed through sharding of the database.
Each piece of data gets a unique ID. The IDs are grouped into shards. The balancer is aware of the range of data in a shard and is able to distribute the load to the appropriate shards
In this example, every shard is hosted by a single server. Note that this gives as a way to divide the work to scale the state, so that N queries per second arriving at the balancer yields N/# shards queries per second at the server. Setting the right number of shards to reduce the demand on any single server becomes a variable for consideration.

Horizontal scaling with shard replicas

Make **K** copies of every shard

For consistency, you can use **consensus protocols**



Scaling

- Increase number of servers and redistribute shards
- Reshard to scale beyond initial shard count (headache)
- Consider splitting / combining
- Linear probing

Using a single server to serve up a single shard gives both a bottleneck in performance and a single point of failure. Before sharding, the database server was a single point of failure. After sharding, a data range could go out of service if the server that supports that shard is lost.

To deal with this and get a more robust and scalable solution, you can make multiple copies of the shards and host them on multiple servers. Increasing the number of servers allows the system to redistribute the shards. A distributed consensus protocol is required to ensure that the copies stay in synch across the cluster of servers. Then you could play games with performance relative to the servers by re-sharding and by splitting larger shards or combining (coalescing) smaller shards to get a more even performance, and running tests to determine how to adjust sharding for optimal performance.