

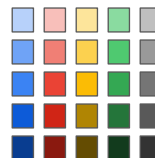
# Google Cloud Platform

## Data Storage Services

v 1.0









Data  
Services



# Data Storage Services

Data Processing  
Services

	Cloud Storage 	Cloud SQL 	Cloud Spanner  Beta	Datastore 	Bigtable 	BigQuery 
<b>Capacity</b>	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
<b>Access metaphor</b>	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
<b>Read</b>	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
<b>Write</b>	One file	INSERT row		put object	put row	Batch/stream
<b>Update granularity</b>	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
<b>Usage</b>	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

Virtually all applications need persistent and durable storage to accomplish their purpose.

Applications vary in their storage requirements. So Google Cloud Platform offers many persistent storage services.

Notice that BigQuery is grayed out. BigQuery sits on the edge between data storage and data processing. You can store data in BigQuery, but the usual reason to do this is to use BigQuery's big data analysis and interactive querying capabilities. For this reason we are going to discuss BigQuery in the Managed Services module.

All data in GCP is encrypted while at rest and encrypted in flight.

These services provide ranges of scaling, performance, and data structure characteristics. Variations within each service complicate things. For example, Cloud Storage stores large objects. However, one type of Cloud Storage is good for streaming video, while another type is intended to archive data that will be accessed no more than once a year.

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

# Scope

- **Infrastructure Track**

- Infrastructure
- Service differentiators
- When to consider using each service
- Basic knowledge for setting up and connecting to a service
- Administration tasks

- **Data Engineering Track**

- How to use a database system
- Design, organization, structure, schema, and use for an application
- Details about how a service stores and retrieves structured data

Google offers an entire learning track on Data Engineering. This module is not a duplicate of that content. The purpose here is to understand from an infrastructure perspective, what services are available and when to consider using them. It's not intended to teach you how to use database systems.

# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

# Data Storage Services

	Cloud Storage	Cloud SQL	Cloud Spanner <small>Beta</small>	Datastore	Bigtable	BigQuery
<b>Capacity</b>	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
<b>Access metaphor</b>	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
<b>Read</b>	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
<b>Write</b>	One file	INSERT row		put object	put row	Batch/stream
<b>Update granularity</b>	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
<b>Usage</b>	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

Cloud Storage stores objects in buckets. There are several differences between Cloud Storage and a file system.

1. A file system has a hierarchical structure. Cloud Storage is unstructured. It is a flat system of buckets (not directories) that cannot be nested.
2. An object name may consist of up to 222 characters. A valid character in an object name include '/' (forward slash). Using this character in object names can simulate some of the hierarchical structure of a file system, even though the slash is not a functionally significant entity.
3. Objects are replicated and distributed for availability. However, there is no distributed equivalent of a file lock. Therefore, the last entity to write to an object "wins". If you use Cloud Storage in a distributed application, the application is responsible for locking and serialization of access.
4. Cloud Storage treats objects as an unstructured series of bytes.

# Storage classes

	Regional	Multi-Regional	Nearline	Coldline
Design Patterns	<b>Data that is used in one region</b> or needs to remain in region	<b>Data that is used globally</b> and has no regional restrictions	<b>Backups</b> Data that is accessed no more than once a month	Archival or <b>Disaster Recovery</b> (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	99.999999999%	99.999999999%	99.999999999%	99.999999999%
Duration	Hot data	Hot data	30 day minimum	90 day minimum
Retrieval cost	none	none	\$	\$\$

Multi-Regional = Data is stored redundantly in multiple locations separated by at least 100 miles. Actual locations are not specified.

Multi-Regional is only available in "Multi-Regional" locations:

<https://cloud.google.com/storage/docs/bucket-locations#location-mr>

Regional is typically lower cost than Multi-Regional. When you select Regional, you must choose a location.

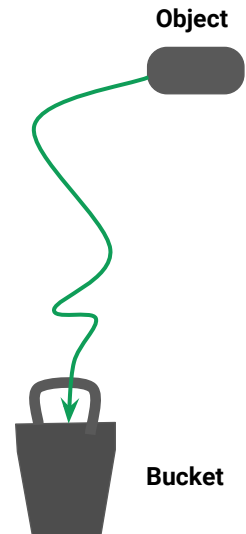
<https://cloud.google.com/storage/pricing>

Nearline and Coldline storage incur an "early deletion" charge of the minimum number of days, if an object is deleted before that time.

Regional and Multi-Regional storage classes typically return the first byte in less than a second (and often in tens of milliseconds). Nearline and Coldline storage may take seconds before the first byte is retrieved.

# Cloud Storage overview

- Buckets
  - Naming requirements
  - Can not be nested
- Objects
  - Inherit storage class of bucket when created
  - No minimum size, unlimited storage
- Access
  - `gsutil` command
  - (RESTful) JSON API or XML API



## Name requirements:

- globally unique
- lowercase, #s, -, . (3-63 chars)
- URI = DNS CNAME
- Access control
  - ACL
  - Signed access
- Encryption at rest
- No minimum size, unlimited storage
- Pay for use
- 99.999999999% durability
- Low latency (time to first byte is typically tens of milliseconds)
- API access

The dot "." is also a valid character in a bucket name. It can be used to create domain-named buckets, such as `mybucket.example.com`. However, there is a verification process required to prove that you are the owner of the domain before this kind of bucket can be created. Max URI bucket name is 222 characters with max 63 characters between dots.

<https://cloud.google.com/storage/docs/domain-name-verification>

The JSON REST API is the preferred API. The XML API is a subset, it doesn't support all features, and it is used commonly with 3rd party tools.

[https://cloud.google.com/storage/docs/json\\_api/](https://cloud.google.com/storage/docs/json_api/)

<https://cloud.google.com/storage/docs/xml-api/overview>

`gsutil` is the command line tool (a python application) used for managing Cloud Storage.

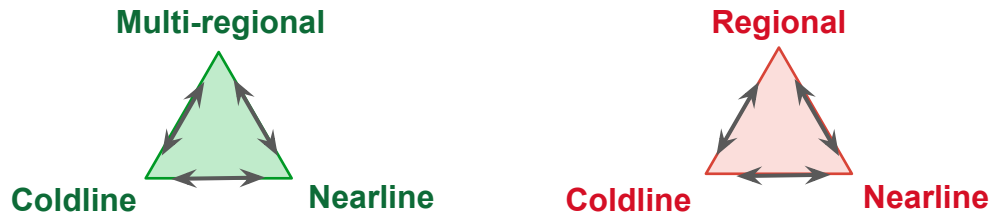
`gsutil` was developed separately from and is not assimilated into the `gcloud` tool.

<https://cloud.google.com/storage/docs/gsutil>

Note: If you have an authenticated and authorized domain, it is possible to use the Google DNS service to host a website in a bucket. The bucket does not have the ability to terminate SSL session, so it can natively serve only HTTP web traffic. However, as you will see in a later module it is possible to terminate an SSL session with a Load Balancer.

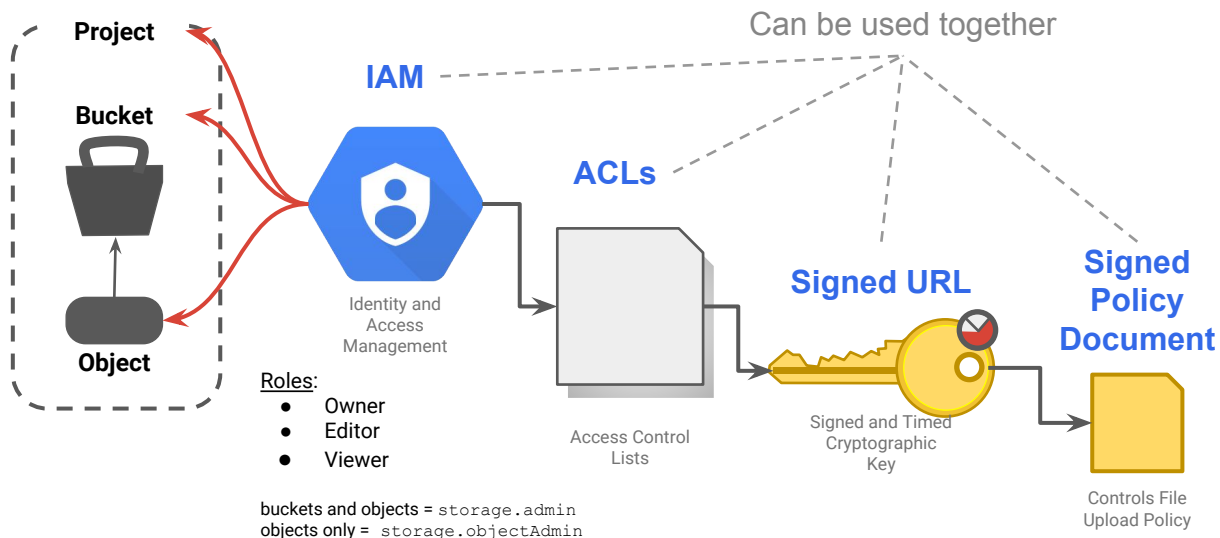


# Changing default storage classes



- You can change the default storage class of a bucket
- The default class is applied to objects as they are created in the bucket
- The change only affects new object added after the change
- New Coldline or Nearline buckets are always Multi-regional
- A Regional bucket can never be changed to Multi-regional
- A Multi-regional bucket can never be changed to Regional
- Objects can be moved from one bucket to another bucket with the same storage class from the console, however moving *objects* to buckets of different storage classes requires using the `gsutil` command from CloudShell

# Access Control



Cloud storage offers layers of increasingly granular access control. For most purposes, IAM is sufficient, and Roles are inherited from project to bucket to object. Access Control Lists (ACL) offer finer control. And for detailed control, Signed URLs provide a cryptographic key that gives time-limited access to a bucket or object. A signed policy document further refines the control by determining what kind of file can be uploaded by someone with a Signed URL.

<https://cloud.google.com/storage/docs/access-control/>

## IAM

<https://cloud.google.com/storage/docs/access-control/iam>

Works as just as with using IAM with any other resource. Project Owners are automatically granted Bucket Owner role for all buckets in the project.

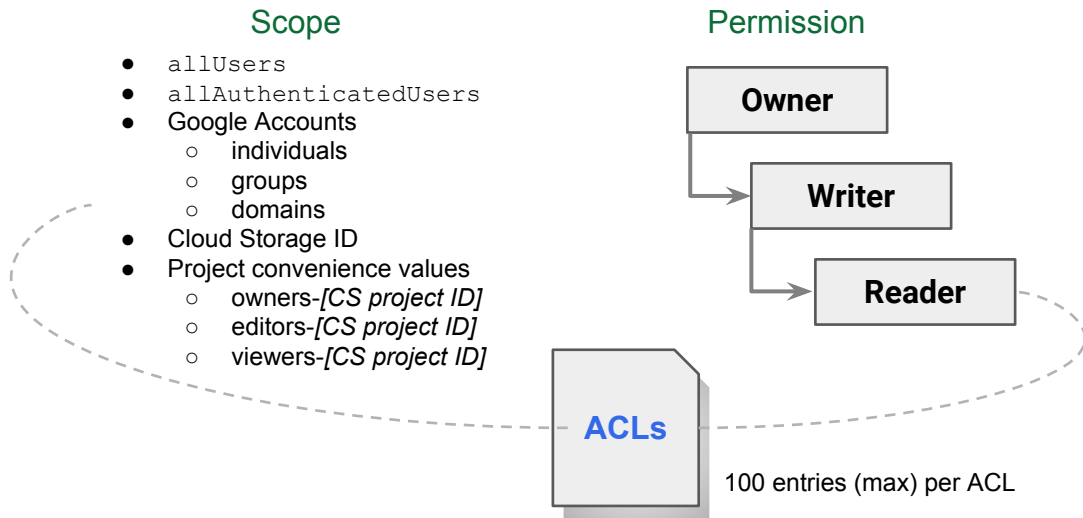
Note that ACLs and IAM are independent. So Project-level IAM permissions will not appear in bucket or object ACLs.

## Signed URLs

A Signed URL gives you the ability to grant access to a bucket without IAM user authentication for a limited period of time.

<https://cloud.google.com/storage/docs/access-control/signed-urls>

# Access Control Lists (ACLs)



Predefined ACL *project-private* is applied by default to all new buckets and objects

ACL permissions are concentric, meaning that the greater access level includes the lesser.

Permission is the action that can be performed.

Scope (sometimes called "grantee") is the identity that can perform the action.

Applied to bucket or to object.

"Owner" is called "FULL\_CONTROL" in the API.

Predefined ACLs provide convenient way to change permissions for common scenarios, for example, revoking access from everyone.

<https://cloud.google.com/storage/docs/access-control/lists>

# Signed URLs

- “Valet key” access to buckets and objects via ticket
  - Ticket is a cryptographically signed URL
  - Time-limited
  - Operations specified in ticket - HTTP GET, PUT, DELETE (not POST)
  - Program can decide when or if to give out signed URL
  - Any user with URL can invoke permitted operations
- Example using private account key and gsutil:  
`gsutil signurl -d 10m path/to/privatekey.p12  
gs://bucket/object`

For some applications, it's easier and more efficient to grant limited-time access tokens that can be used by any user rather than use account-based authentication for controlling resource access (e.g. when you don't want to require users have Google accounts).

Signed URLs allow you to do this for Google Cloud Storage. You create a URL that grants read or write access to a specific GCS resource and specifies when the access expires. That URL is signed using a private key associated with a service account. When the request is received, Cloud Storage can verify that the access-granting URL was issued on behalf of a trusted security principal (the service account), and delegates its trust of that account to the holder of the URL.

After you give out the signed URL, it is out of your control. People copy it or steal it. So you want the signed URL to expire after some reasonable amount of time.

The types of operations that could be performed by using signed URL, kind of align with the XML APIs.

# Example: Signed URL

```
http://google-testbucket.storage.googleapis.com/testdata.txt?  
GoogleAccessId=1234567890123@developer.gserviceaccount.com&  
Expires=1331155464&  
Signature=BClz9e...LoDeAGnfzCd4fTswcLbal9sFpqXsQI8IQi1493mw%3D
```

- Query parameters
  - GoogleAccessId: Email address of the service account
  - Expires: When the signature expires in UNIX epoch
  - Signature: Signature is cryptographic hash of composite URL string
- The string that is digitally signed must contain:
  - HTTP Verb (GET)
  - Expiration
  - Canonical resource (/bucket/object)

This is an example signed URL that authorizes reading an object.

The URL contains:

- GCS path to the object
- GoogleAccessID - used by Cloud Storage to get the corresponding public key to decrypt the signature
- Expiration time (Unix) epoch
- Cryptographic signature

The string you sign is based on the following:

- HTTP Verb + '\n'\*
- Content\_MD5 + '\n'
- Content\_Type + '\n'
- Expiration + '\n'\*
- Canonicalized\_Extension\_Headers + '\n'
- Canonicalized\_Resource + '\n'\*

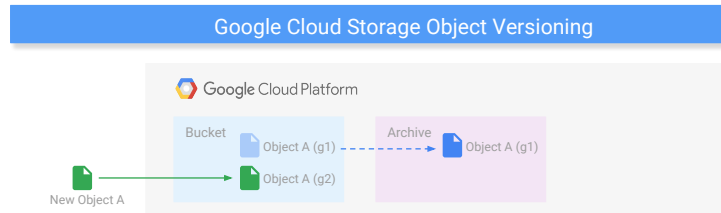
\* = required

# Cloud Storage Features

- Customer Supplied Encryption Key (CSEK)
  - Use your own key instead of Google-managed keys
- Lifecycle Management
  - Automatically delete or archive objects
- Object Versioning
  - Maintain multiple versions of objects\*
- Directory Synchronization
  - Synchronizes a VM directory with a bucket
- Object Change Notification
- Data Import

\*Note: You are charged for the versions as if they were multiple files.

# Object Versioning



- Buckets with Object Versioning enabled maintain a history of modifications (overwrite/delete) of objects
- Bucket users can list archived versions of an object, restore an object to an older state, or delete a version
- Supports conditional updates

For more information, see:

<https://cloud.google.com/storage/docs/object-versioning>

# Object Lifecycle Management

- Lifecycle management policies specify actions to be performed on objects that meet certain rules
- Examples include:
  - Downgrade storage class on objects older than a year
  - Delete objects created before a specific date
  - Keep only the 3 most recent versions of an object
- Object inspection occurs asynchronous batches, so rules may not be applied right away
- Changes to lifecycle configurations can take 24 hours to apply

Lifecycle actions include:

- Delete
- SetStorageClass

Lifecycle conditions include:

- Age
- CreatedBefore
- IsLive
- MatchesStorageClass
- NumberOfNewerVersions

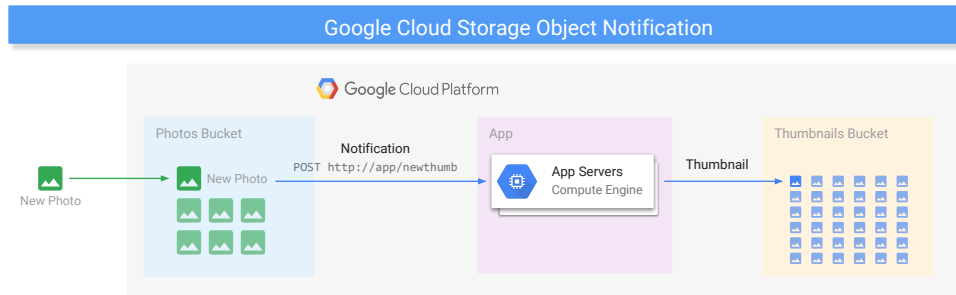
Lifecycle configurations can be updated using the XML API, the JSON API, or the `gsutil` utility.

For more information see: <https://cloud.google.com/storage/docs/lifecycle>



# Object Change Notification

- Cloud Storage can watch a bucket and send notifications to external applications when objects change
- Notification channels are essentially webhooks
  - POST requests are made to a specified URL



For more information see:

<https://cloud.google.com/storage/docs/object-change-notification>

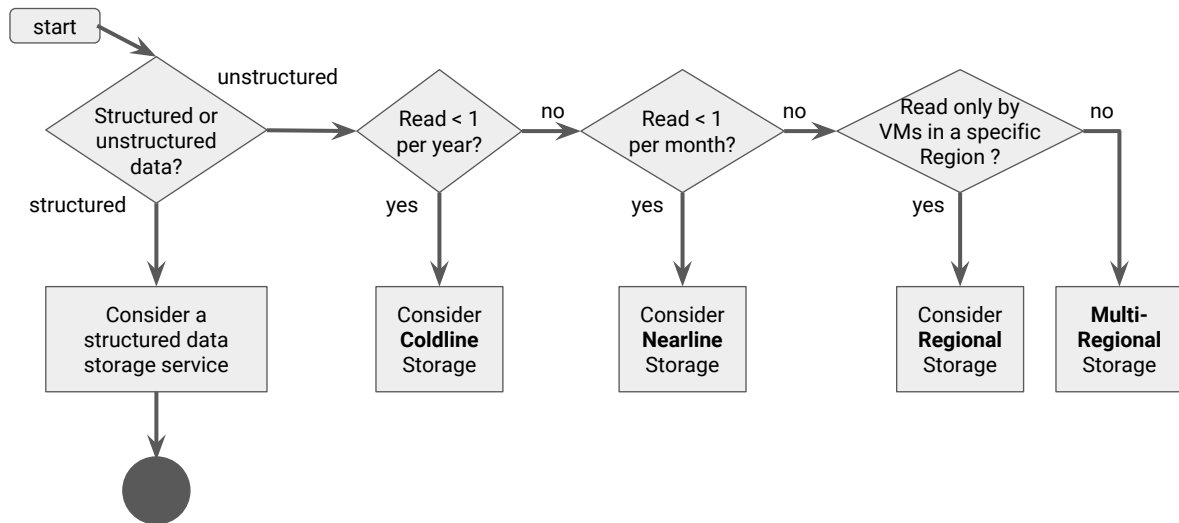
# Data Import

- The **Cloud Storage Transfer service** enables high-performance imports of online data into Cloud Storage buckets
  - Imports from another bucket, an S3 bucket, or Web source
  - Can schedule transfers, create filters, etc.
  - Accessible via console or APIs
- **Offline Media Import** is a service where physical media is sent to a 3rd-party provider who uploads the data
  - Useful if you have expensive or unreliable Internet connections
  - Providers generally offer export services as well
  - Providers include: Iron Mountain, Prime Focus Technologies

For more information see:

<https://cloud.google.com/storage/docs/offline-media-import-export>

# Choosing Cloud Storage



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.

# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

# Lab #1 Cloud Storage Lab

In this lab you will create a Cloud Storage bucket and upload sample files to it. To exercise the advanced features of Cloud Storage you will authorize a VM with a service account and use the `gsutil` command. You will also modify the configuration for the `gsutil` command.

Features you will explore:

ACL, CSEK, Rotate CSEK Keys, Lifecycle Management, Versioning, Directory Synchronization

# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

# Data Storage Services

	Cloud Storage	Cloud SQL	Cloud Spanner <small>Beta</small>	Datastore	Bigtable	BigQuery
<b>Capacity</b>	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
<b>Access metaphor</b>	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
<b>Read</b>	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
<b>Write</b>	One file	INSERT row		put object	put row	Batch/stream
<b>Update granularity</b>	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
<b>Usage</b>	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

Cloud SQL is a hosted MySQL service.  
It provides:

- Rich query language
- Primary and secondary indexes
- ACID transactions
- Relational integrity
- Stored procedures

# Cloud SQL

- A fully managed MySQL database service
  - Fully managed instances
  - Patches and updates automatically applied
  - You still have to administer MySQL users
- Cloud SQL supports many clients
  - `gcloud beta sql`
  - Google App Engine (GAE), G Suite scripts
  - Applications and tools
    - SQL Workbench, Toad
    - External applications using standard MySQL drivers



Cloud SQL  
Service

You can't reuse an instance name for 7 days after it is deleted.

There are several unsupported features and statements:

<https://cloud.google.com/sql/docs/features>

Cloud SQL supports: Stored procedures, Triggers, and Views

Cloud SQL does not support: User-defined functions, Internal MySQL replication, statements and functions related to files and plugins



# Cloud SQL Generations

- **Second Generation**

- Up to 104GB of RAM and 10TB data storage
- MySQL 5.6 or 5.7
- InnoDB only

- **First Generation**

- Up to 16GB of RAM and 500GB data storage
- MySQL 5.5

*Some MySQL features not supported; UDFs*

2nd Gen compares with 1st Gen:

- Up to 7x greater throughput and 20x storage capacity
- Less expensive for most use cases
- Failover and replica options

# Cloud SQL services

- Replica services
  - Read replicas
  - Failover replicas
  - External replicas
  - External master to Cloud SQL 1st Gen replica
- Backup service
  - On demand
  - Scheduled/automated
- Auto-quiesce
- Import / Export
  - Import data to Cloud SQL
  - Export data from Cloud SQL
- Scaling
  - Scale-up (change machine capacity)
    - Requires restart
  - Scale-out
    - Via read replicas

## Replication

Instances are replicated across zones in a region. Automatically fails over to another zone in event of an outage.

Options: 1) synchronous replication for slower, more reliable writes, 2) asynchronous replication for faster writes that could result in loss of latest updates in unlikely event of data center failure

<https://cloud.google.com/sql/docs/mysql/replication/tips>

Backup schedule: Daily backups over a 7 day period, the time slot can be configured

Auto-quiesce can be disabled, on by default. Quiesces MySQL after a period with no traffic. Restarts when traffic arrives.

Feature to reduce cost when traffic is bursty or infrequent. Or disable it to run "hot" for faster first responses.

# Connecting to Cloud SQL

- Basic connection
  - IP address
  - Add authorized network
- Secure access
  - Whitelist IP addresses
  - Add to authorized networks list
  - Configure SSL
  - Static IP
- Instance level access
- MySQL database level access
  - MySQL users
  - Database access

<https://cloud.google.com/sql/docs/mysql/configure-ssl-instance>  
<https://cloud.google.com/sql/docs/mysql/connect-admin-ip>

## Connecting from outside Google Cloud Platform

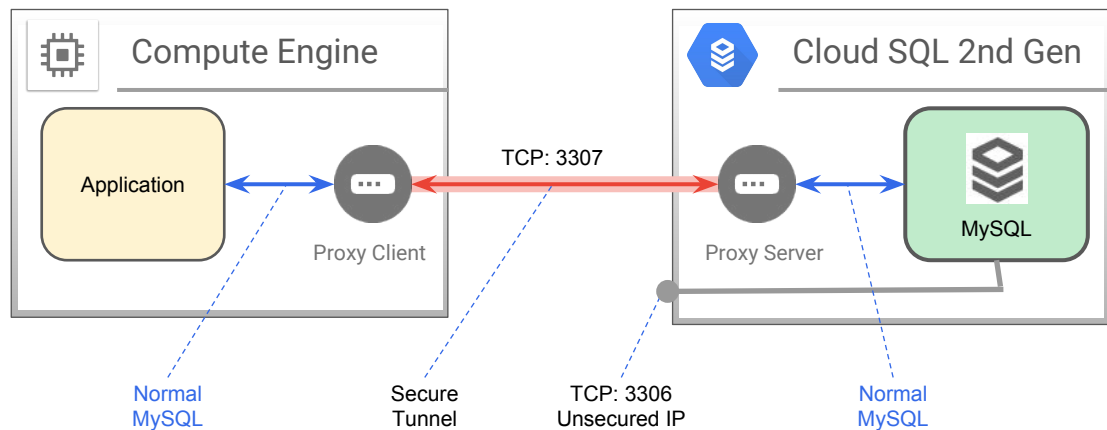
1. Create Cloud SQL instance in desired region
2. Determine the IP address of your machine
3. Add external IP address of machine in Cloud SQL instance authorized networks list
4. Assign Cloud SQL an IPv4 address
5. Connect with username and password to Cloud SQL address

## Connecting from Compute Engine

1. Optionally create Cloud SQL instance in same zone as Compute Engine instance
2. Assign Compute Engine external IP address
3. Add external IP address of Compute Engine instance in Cloud SQL instance authorized networks list
4. Assign Cloud SQL an IPv4 address
5. Connect with username and password to Cloud SQL address



# Cloud SQL Proxy and Secure SSL



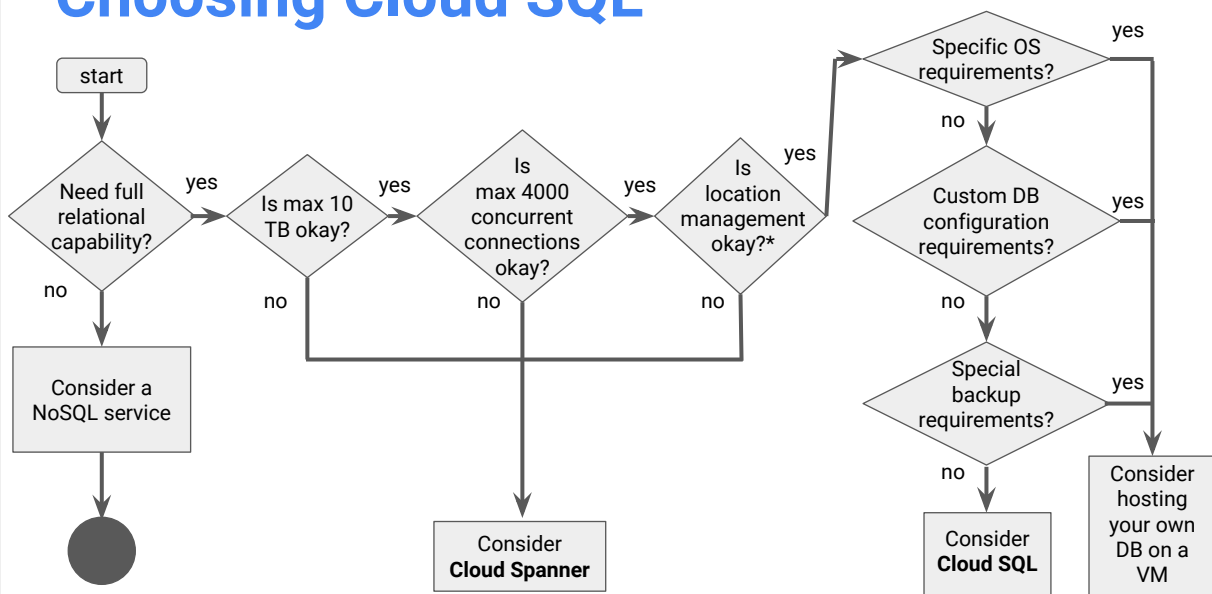
<https://cloud.google.com/sql/docs/mysql-connect-proxy>

<https://cloud.google.com/sql/docs/sql-proxy>

If you are not using Cloud SQL Proxy, you are strongly encouraged to use Cloud SSL:

<https://cloud.google.com/sql/docs/mysql/configure-ssl-instance>

# Choosing Cloud SQL



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.

# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

## Lab #2 Cloud SQL Lab

In this lab you will create a Cloud SQL instance.

You'll configure a failover replica.

You'll connect to it using the `gcloud beta sql` command from cloudshell

You will authorize a VM with a service account install and configure the Cloud SQL Proxy for secure access to the Master instance

`gcloud beta sql`, Cloud SQL Proxy



# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → **Cloud Spanner**
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

# Data Storage Services

	Cloud Storage	Cloud SQL	Cloud Spanner <small>Beta</small>	Datastore	Bigtable	BigQuery
<b>Capacity</b>	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
<b>Access metaphor</b>	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
<b>Read</b>	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
<b>Write</b>	One file	INSERT row		put object	put row	Batch/stream
<b>Update granularity</b>	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
<b>Usage</b>	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

Before Cloud Spanner, architects were forced to choose between scalability using a NoSQL solution or transactional integrity using a Relational solution.

# Cloud Spanner <sup>Beta</sup>

- Strong consistency
  - Strongly consistent secondary indexes
  - Globally consistent
- SQL support
  - SQL (ANSI 2011 with extensions)
  - ALTER statements for schema changes
- Managed instances
  - High availability through data replication



Cloud  
Spanner

Cloud Spanner is suited for applications that require relational database support, strong consistency, transactions, and horizontal scalability. Natural use cases would include financial applications and inventory applications traditionally served by relational database technology.

# Characteristics

	Cloud Spanner	Relational DB	Non-Relational DB
Schema	Yes	Yes	No
SQL	Yes	Yes	No
Consistency	Strong	Strong	Eventual
Availability	High	Failover	High
Scalability	Horizontal	Vertical	Horizontal
Replication	Automatic	Configurable	Configurable

Spanner is the first horizontally scalable globally consistent database. It is proprietary, not open source.

Cloud Spanner is used for applications that use relational, structured, and semi-structured data, and require high availability, strong consistency, and transactional reads and writes.

# Spanner

- Open Standards
  - Standard SQL (ANSI 2011)
  - Encryption, Audit logging, Identity and Access Management
  - Client libraries in popular languages
    - Java, Python, Go, Node.js
    - JDBC driver
- Workloads
  - Transactional
  - Scale-out
  - Global data plane
  - Database consolidation

## Transactional

Companies that have outgrown their single-instance RDBMS and have already moved to NoSQL solution, but need transactional consistency, or they are looking to move to a scalable solution

## Scale-out

Companies currently sharding databases because they need more read or write throughput than can be placed on a single node

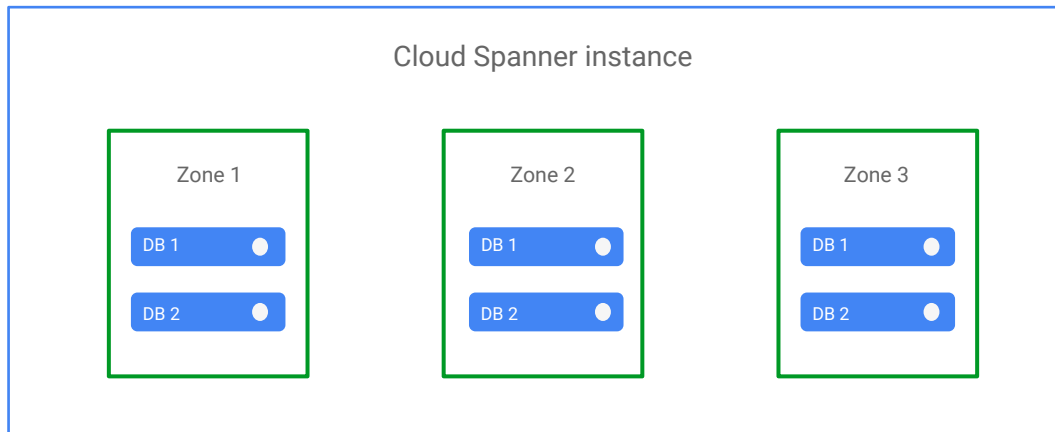
## Global data plane

Companies and/or developers building applications that have global data and need strong consistency

## Database consolidation

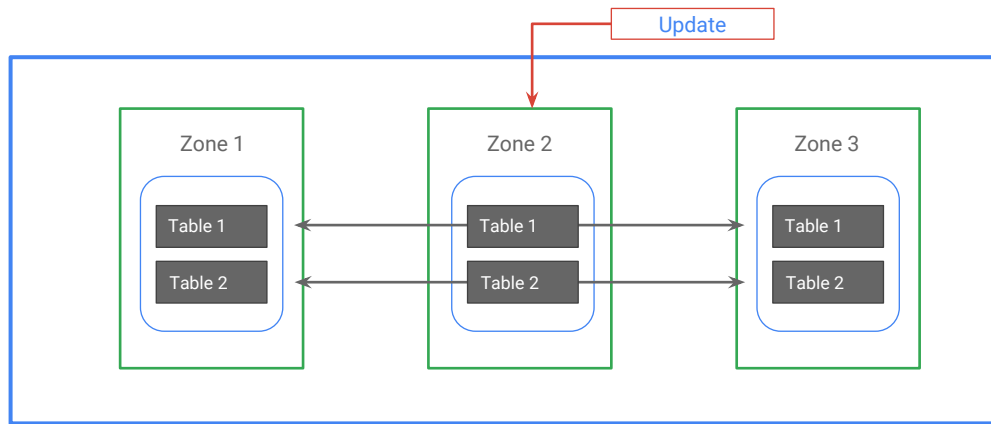
Companies that store their business data in multiple database products with variable maintenance overheads and capabilities and need consolidation of their data

# Architecture



This architecture allows HA and global placement  
Data is replicated in N cloud zones which can be within one region or across several regions.  
DB placement is configurable. You can choose which region to put your DB in.

# Data replication



Writes are synchronous. Data is always consistent and has ACID properties like any other relational DB

Relational semantics

High throughput → data is sharded within the zone

HA built-in → No manual intervention needed on zone failure.

Paxos replication (leslie lamport paper) - only require majority (not all zones) for commits

How Spanner works: <https://research.google.com/pubs/pub45855.html>

Cloud Spanner best practices: <https://cloud.google.com/spanner/docs/best-practices>

# IAM Roles

- You can apply IAM roles to individual databases
  - `spanner.admin`
  - `spanner.databaseAdmin`
  - `spanner.databaseReader`
  - `spanner.databaseUser`
  - `spanner.viewer`

We can't cover everything about Cloud Spanner here. We could spend 3 to 4 days on Cloud Spanner alone.

Please reference the Cloud Spanner documentation.

<https://cloud.google.com/spanner/docs/>



# Feature details

- Tables
  - Optional parent-child relationships between tables
  - Interleaving of child rows within parent rows on primary key
- Primary keys and secondary keys
- Database splits
- Transactions
  - Locking read-write, and read-only
- Timestamp bounds
  - Strong, Bounded staleness, Exact staleness, Maximum staleness

<https://cloud.google.com/spanner/>

<https://cloud.google.com/spanner/docs/schema-and-data-model>

Cloud Spanner divides your data into chunks called "splits", where each split can move independently from each other and get assigned to different servers, which could be in different physical locations.

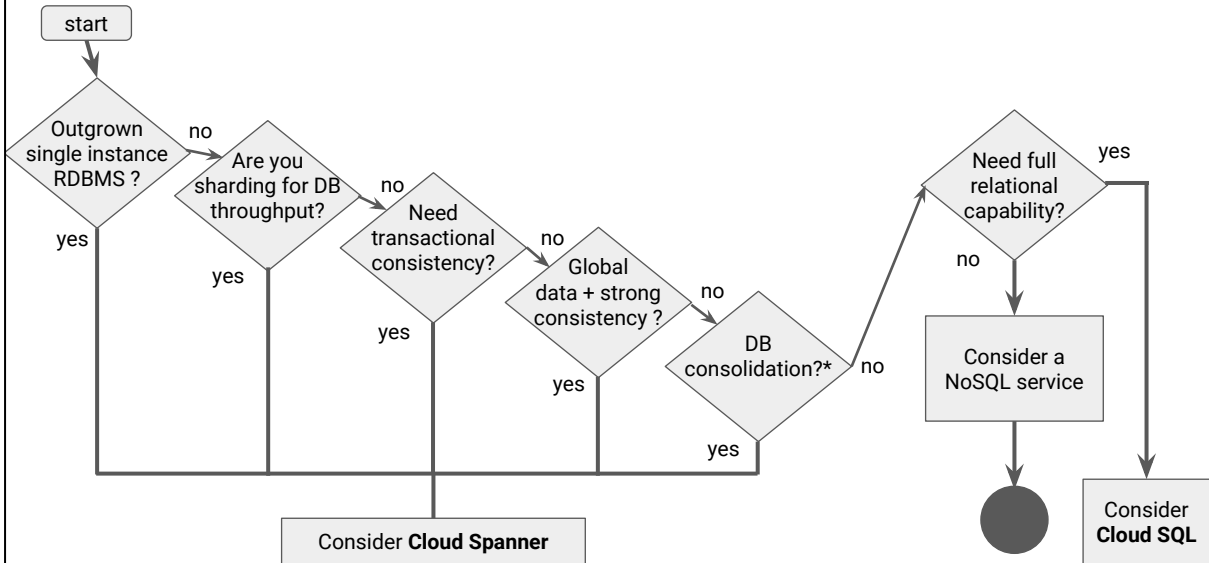
## Timestamp bounds

**Strong:** Cloud Spanner provides a bound type for strong reads. Strong reads are guaranteed to see the effects of all transactions that have committed before the start of the read.

**Bounded staleness:** Bounded staleness modes allow Cloud Spanner to pick the read timestamp, subject to a user- provided staleness bound.

**Exact staleness:** The timestamp can either be expressed as an absolute Cloud Spanner commit timestamp or a staleness relative to the current time.

# Choosing Cloud Spanner



## \*DB Consolidation

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.

# Agenda

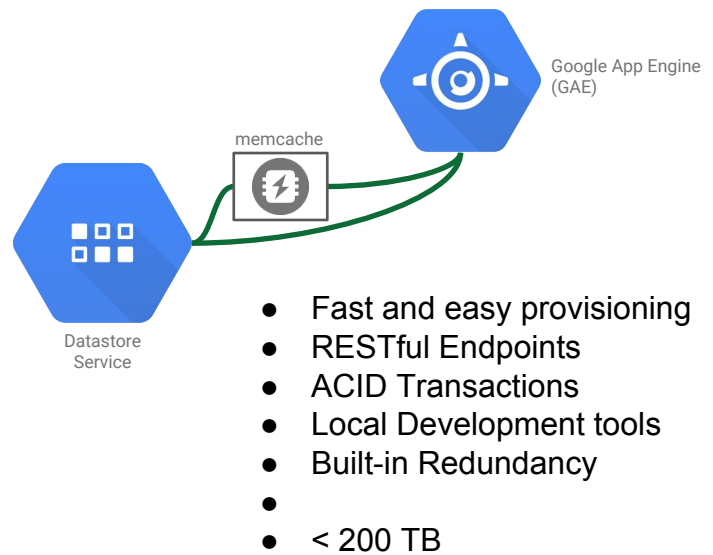
- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

# Data Storage Services

	Cloud Storage	Cloud SQL	Cloud Spanner <small>Beta</small>	Datastore	Bigtable	BigQuery
<b>Capacity</b>	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
<b>Access metaphor</b>	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
<b>Read</b>	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
<b>Write</b>	One file	INSERT row		put object	put row	Batch/stream
<b>Update granularity</b>	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
<b>Usage</b>	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

# Datastore

- API support for several languages
- Global access
  - Data replicated across several data centers
- Equally fast queries for any sized dataset
- Autoscale
- Schemaless access
- SQL-like capabilities
- Authentication, secure sharing



Datastore is used with GAE applications. It is good for structured pure-serve use cases. It's not a good choice for relational database needs or for analytical data processing. Consider Cloud SQL and Bigtable, respectively.

Datastore is paired with a memcache service to increase performance for repeatedly read data. Typically in development, the GAE application will try memcache first and then on a cache-miss, access Datastore. This strategy radically improves performance and reduces costs.

JSON API, Java (JPA, JPO, Objectify), Python (NDB), Ruby, Node.js

Charges for storage and for read/write operations.

Information on consistency models in Datastore:

<https://cloud.google.com/datastore/docs/articles/balancing-strong-and-eventual-consistency-with-google-cloud-datastore/>

# Datastore

- **Entity** - a single object
- **Kind** - category of an object
- **Property** - individual data for an object
- **Key** - unique ID
  
- Entity groups and ancestor queries
  - An "ancestor query" enables a strong consistency result versus a non-ancestor query which yields an eventually consistent result
  - Enables application developers to balance requests

Compared to a relational database, here are the rough equivalents: Kind = Table, Entity = Row, Property = Field, Key = Primary Key.

<https://cloud.google.com/datastore/docs/datastore-api-tutorial>

<https://cloud.google.com/datastore/docs/concepts/overview>

Fully managed service, no provisioning required

Regional (US or Europe)

Transactions supported with some restrictions

Secondary indexes

Strong or Eventual consistency depending on design decisions

JSON API, Java (JPA, JPO, Objectify), Python (NDB), Ruby, Node.js

10s ms R/W latency. QPS and throughput are data model-dependent

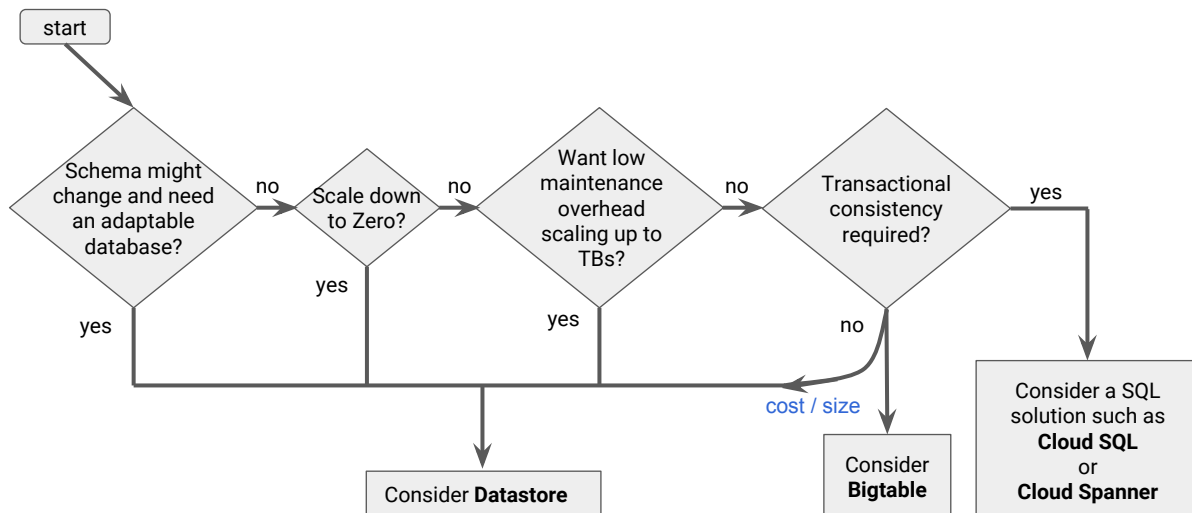
< 200TB

Charged for storage and read/write operations.

# Datastore replication

- Multiple Locations
  - Multi-Regional
    - Multi-Region redundancy, Higher Availability
  - Regional locations
    - Lower write latency, Co-location with other resources
  - Global Points of Presence - lower latency for the end user
- Service Level Agreement
  - Regional (Multi-zone) & Multi-Region replication enable a high availability with an SLA covering:
  - 99.95% for Multi-Region locations, and
  - 99.9% for Regional locations

# Choosing Datastore





# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

## Lab #3 Cloud Datastore

In this lab you create a Datastore NoSQL database, learn to add and query data, and explore the Datastore Admin features.

# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

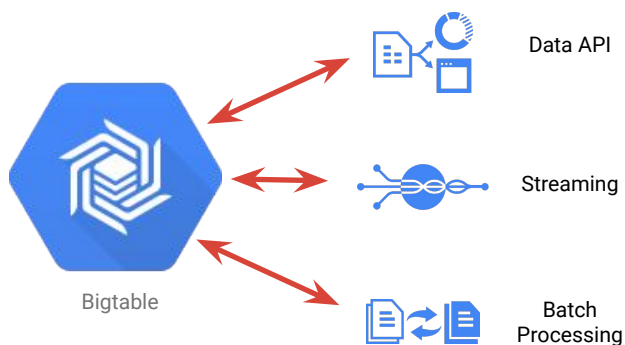
# Data Storage Services

	Cloud Storage	Cloud SQL	Cloud Spanner <small>Beta</small>	Datastore	Bigtable	BigQuery
<b>Capacity</b>	Petabytes +	Gigabytes	1000s+ nodes	Terabytes	Petabytes	Petabytes
<b>Access metaphor</b>	Like files in a file system	Relational database	Globally scalable RDBMS	Persistent Hashmap	Key-value(s), HBase API	Relational
<b>Read</b>	Have to copy to local disk	SELECT rows	transactional reads and writes	filter objects on property	scan rows	SELECT rows
<b>Write</b>	One file	INSERT row		put object	put row	Batch/stream
<b>Update granularity</b>	An object (a "file")	Field	SQL, Schemas ACID transactions Strong consistency High Availability	Attribute	Row	Field
<b>Usage</b>	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

# Cloud Bigtable

- Fully-managed NoSQL database
- Petabyte-scale with very low latency
- Seamless scalability for throughput
- Learns and adjusts to access patterns

# Cloud Bigtable



- Flexible scalable data service
- High throughput, low latency
- High capacity (Petabytes)
- Ideal for storing single-keyed data with very low latency
- Excellent for Big Data applications
- Can be access via HBase extension
- Benefits over HBase
  - Scales by machine count, not limited by QPS
  - Low Ops - admin tasks are automatic
  - Resize in seconds

NoSQL database, originally written to support search, now supports dozens of products.

Cloud Bigtable is a sparsely populated table that can scale to billions of rows and thousands of columns, allowing you to store terabytes or even petabytes of data. A single value in each row is indexed; this value is known as the row key. Cloud Bigtable is ideal for storing very large amounts of single-keyed data with very low latency. It supports high read and write throughput at low latency, and it is an ideal data source for MapReduce operations.

<https://cloud.google.com/bigtable/docs/overview>

- Bigtable is a NoSQL database system
- Petabytes in size
- A sparse, distributed, persistent multidimensional sorted map, indexed by row key, column key, and timestamp
- each value in the map is an array of bytes
- Eventually consistent
- Cross-zone replication

## Data API

Data can be read from and written to Cloud Bigtable through a data service layer like: Managed VMs, the HBase REST Server, a Java Server using the HBase client. Typically this will be to serve data to applications, dashboards and data services.

**Streaming**

Data can be streamed in (written event by event) through a variety of popular stream processing frameworks like:

Dataflow Streaming, Spark Streaming, Storm.

**Batch Processing**

Data can be read from and written to Cloud Bigtable through batch processes like:

Hadoop MapReduce, Dataflow, Spark. Often, summarized or newly calculated data is written back to Cloud Bigtable or to a downstream database

**Bigtable**

Need to create clusters and add nodes if more QPS is needed

Zonal (zones in us-central1, eu-west1 or asia-east1 regions).

Strongly consistent in one zone at a row level

HBase API (with some differences), Go Client library (gRPC)

Highly predictable. Up to 10,000 queries per second (QPS) for each node, throughput of up to 10 MB/s per node. Single ms R/W latency


2TB - 10PB

Charged for storage and provisioned nodes

# Structure and schema

"follows" column family

	Follows			
Row Key	gwashtington	jadams	tjefferson	wmckinley
gwashtington		1		
jadams	1		1	
tjefferson	1	1		1
wmckinley			1	

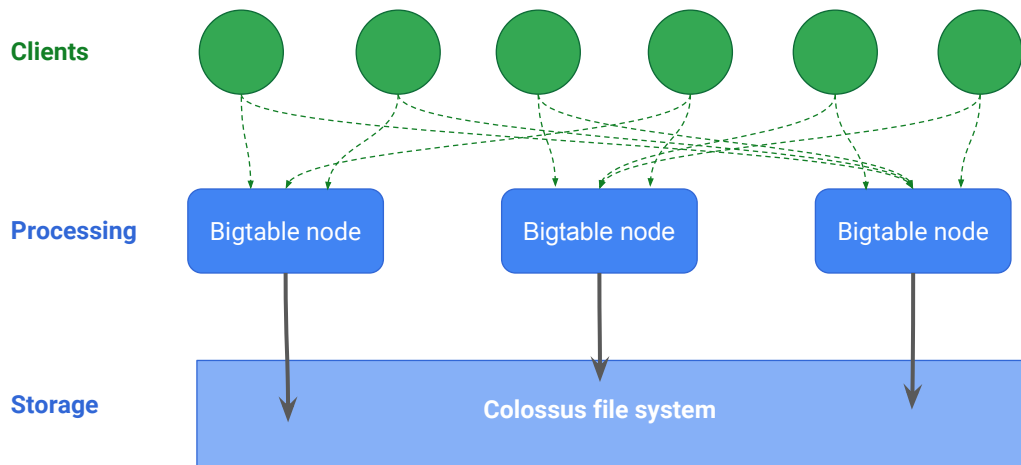


- NoSQL, sparse wide-column database
- Single index: the row key
- Atomic single-row transactions

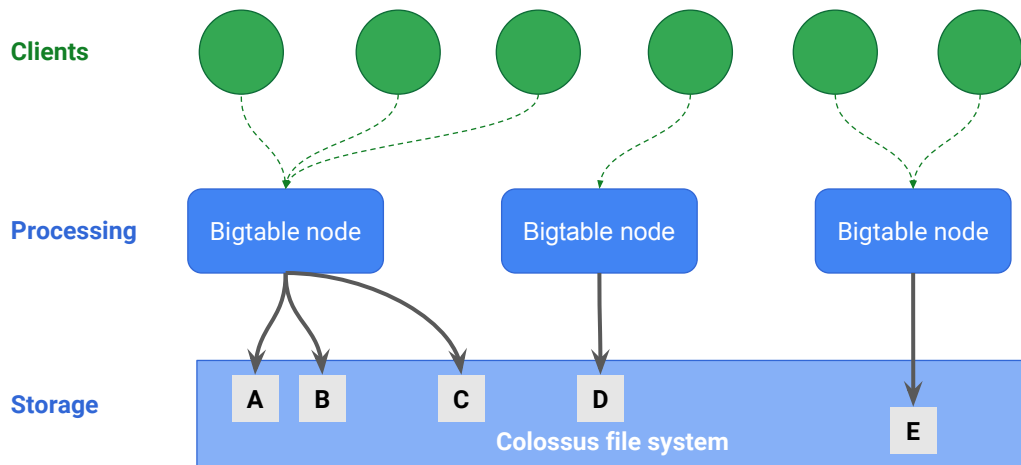
multiple versions



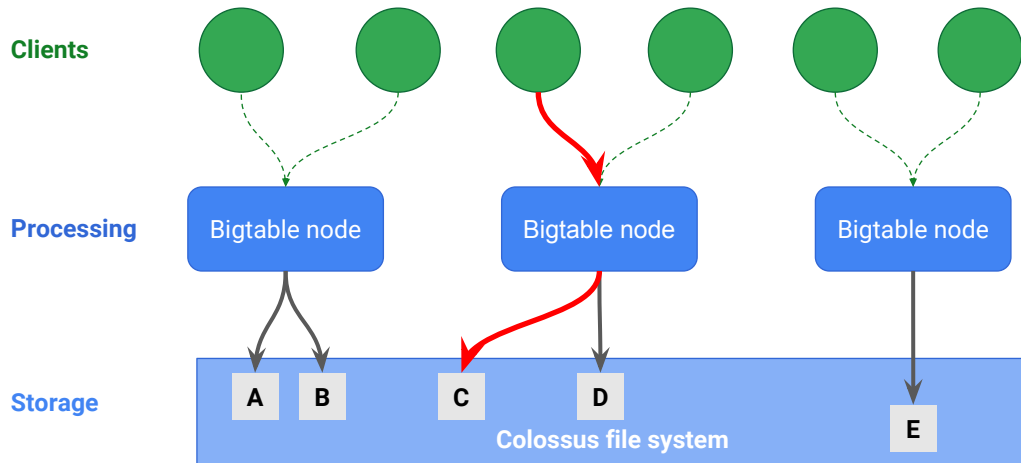
# Processing is separated from storage



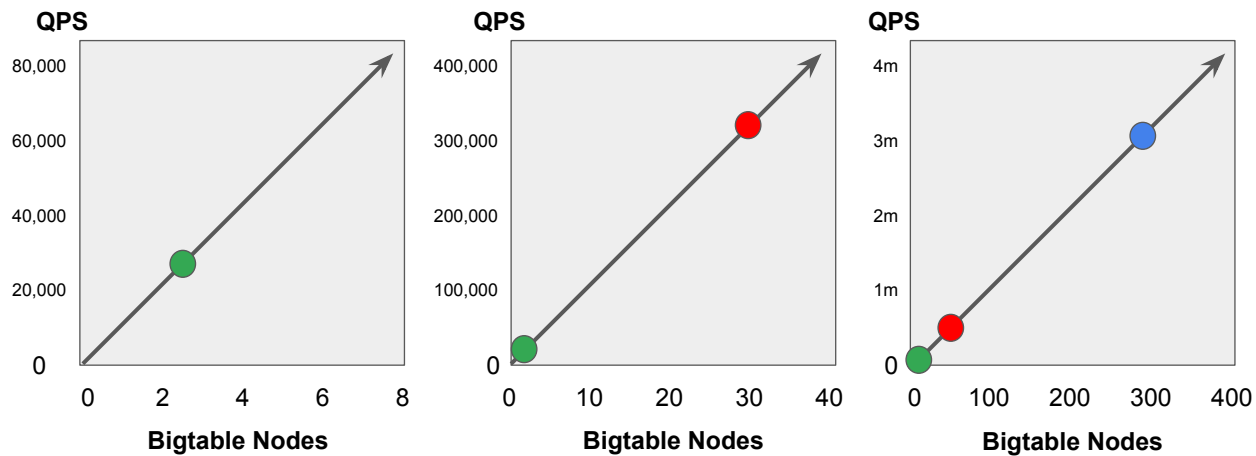
# Learns access patterns



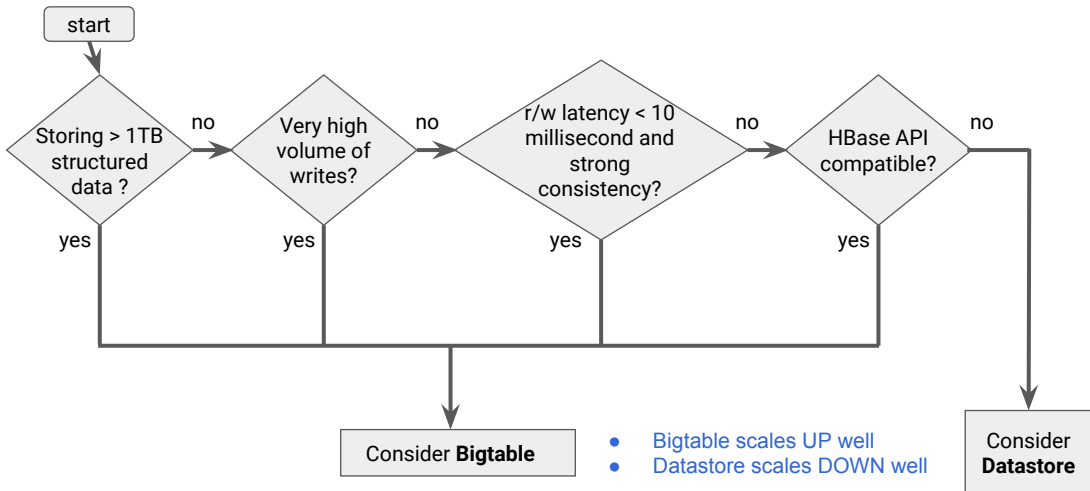
# Rebalances without moving data



# Throughput scales linearly



# Choosing Bigtable



The smallest Bigtable cluster you can create has three nodes and can handle 30,000 operations per second. You pay for those nodes while they are operational, whether your application is using them or not. Compare this with Datastore, where you only pay for the resources that your application is using.

# Agenda

- 1 → Google Cloud Storage (GCS)
- 2 → Lab #1
- 3 → Cloud SQL
- 4 → Lab #2
- 5 → Cloud Spanner
- 6 → Datastore
- 7 → Lab #3
- 8 → Bigtable
- 9 → Review

## More...

- Cloud Storage
  - <https://cloud.google.com/storage/docs/>
- Cloud SQL
  - <https://cloud.google.com/sql/docs/>
- Spanner
  - <https://cloud.google.com/spanner/docs/>
- Datastore
  - <https://cloud.google.com/datastore/docs/>
- Bigtable
  - <https://cloud.google.com/bigtable/docs/>

