Google Cloud

Partner Certification Academy

Professional Cloud Developer

The information in this presentation is classified:

Google confidential & proprietary

⚠ This presentation is shared with you under NDA.

- Do **not** record or take screenshots of this presentation.
- Do **not** share or otherwise distribute the information in this presentation with anyone **inside** or **outside** of your organization.

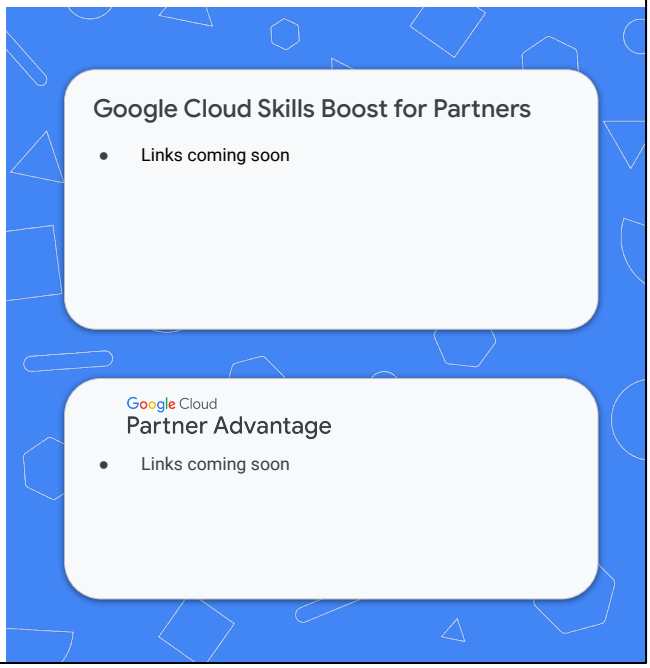
Thank you!

Google Cloud

Source Materials

Some of this program's content has been sourced from the following resources:

- Google Cloud certification site
- Google Cloud documentation
- Google Cloud console
- Google Cloud courses and workshops
- Google Cloud white papers
- Google Cloud Blog
- Google Cloud YouTube channel
- Google Cloud samples
- Google codelabs
- Google Cloud partner-exclusive resources

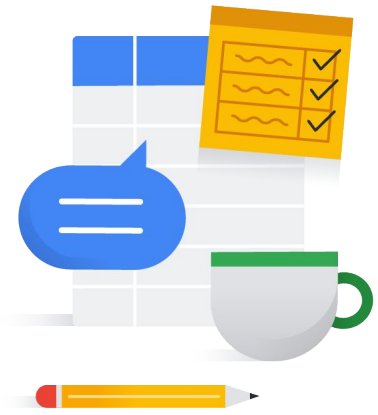


Session logistics

- When you have a question, please:
  - Click the Raise hand button in Google Meet.
  - Or add your question to the Q&A section of Google Meet.
  - Please note that answers may be deferred until the end of the session.
- These slides are available in the Student Lecture section of your Qwiklabs classroom.
- The session is **not recorded.**
- Google Meet does not have persistent chat.
  - If you get disconnected, you will lose the chat history.
  - Please copy any important URLs to a local text file as they appear in the chat.

Program issues or concerns?

- For questions regarding Cloud Skills Boost access, Qwiklabs issues, voucher queries, etc.
  - cloud-partner-training@google.com
- For questions regarding Partner Advantage access
  - https://support.google.com/googlecloud/topic/9198654



Partner Certification Academy

A differentiated learning experience for the busy professional

Our goal is to help you prepare for Google Cloud certification exams

These programs may include:

- On-demand learning
- Self-paced labs
- Mentor-led workshops
- A voucher for the exam

The workshop sessions:

- **Are NOT training sessions - that's the purpose of the on-demand content.**
- Help you review key concepts on the exam guide.
- Will NOT discuss actual exam questions.

Professional Cloud Developer (PCD)

A Professional Cloud Developer builds scalable and highly available applications using Google-recommended tools and best practices. This individual has experience with cloud-native applications, developer tools, managed services, and next-generation databases. A Professional Cloud Developer also has proficiency with at least one general-purpose programming language and instruments their code to produce metrics, logs, and traces.

The Professional Cloud Developer exam assesses your ability to:

- ✓ Design highly scalable, available, reliable cloud-native applications
- ✓ Deploy applications
- ✓ Manage deployed applications
- ✓ Build and test applications
- ✓ Integrate Google Cloud services

https://cloud.google.com/learn/certification/cloud-developer

Module Agenda

01 Cloud Storage

02 Filestore

03 Database Overview

04 Cloud SQL

05 Cloud Spanner

06 Firebase Overview

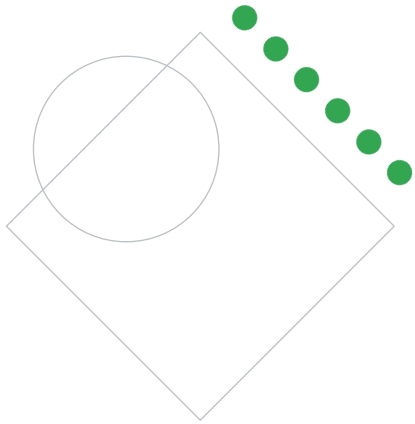
07 Firebase Database Options

08 Cloud Bigtable

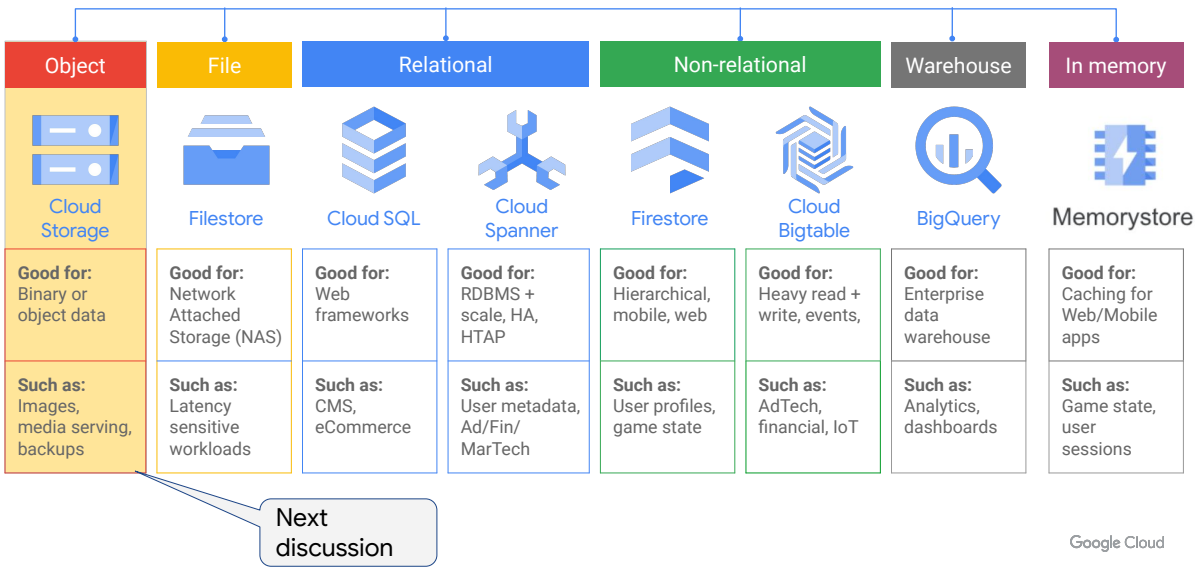
09 BigQuery

10 Cloud Memorystore

Cloud Storage

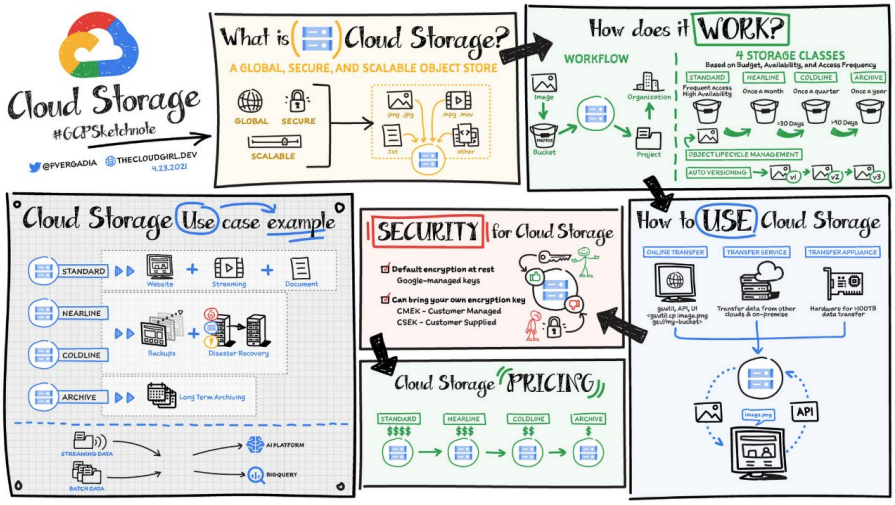


Storage and database services



This table shows the storage and database services and highlights the storage service type (object, file, relational, non-relational, and data warehouse), what each service is good for, and intended use.

All you need to know about Cloud Storage

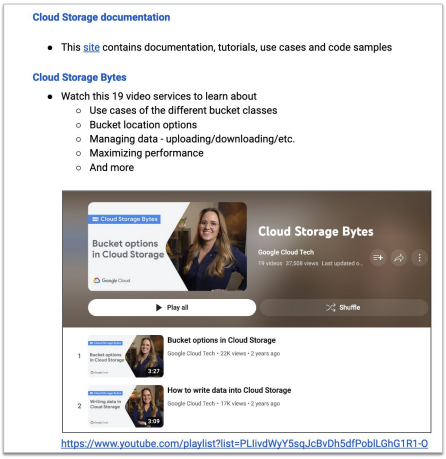


https://cloud.google.com/blog/topics/developers-practitioners/all-you-need-know-about-cloud-storage

Guide: Cloud Storage

Topics covered include

- Documentation
- Youtube videos covering Cloud Storage in depth
- Cloud Storage IAM roles
- Data encryption options
- Best practices
- Troubleshooting methods

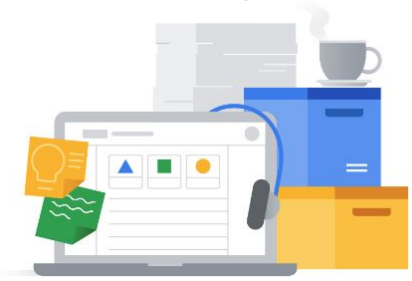


Cloud Storage

Cloud Storage is a fully managed binary large-object (BLOB) storage service

Worldwide storage and retrieval of any amount of data at any time, including:

- Online content
- Backup and archiving
- Storage of intermediate results
- And much more....



Object storage for companies of all sizes
https://cloud.google.com/storage

Cloud Storage's primary use is whenever binary large-object storage (also known as a "BLOB") is needed for online content such as videos and photos, for backup and archived data, and for storage of intermediate results in processing workflows.

Files are organized into buckets

Unique name

Geographic location

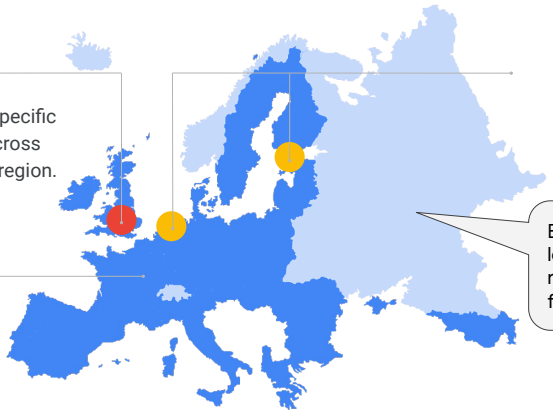
Cloud Storage files are organized into buckets. A bucket needs a globally-unique name and a specific geographic location for where it should be stored, and an ideal location for a bucket is where latency is minimized. For example, if most of your users are in Europe, you probably want to pick a European location so either a specific Google Cloud region in Europe, or else the EU multi-region.

Store data closest to where it will be used

Store data in regional buckets close to where it is used, e.g., analytic workloads

Regional

Your data is stored in a specific region with replication across availability zones in that region.



Dual-regional

Your data is replicated across a specific pair of regions.

Be careful of your location if you have residency requirements for data, e.g., GDPR

Store data in multi/dual regions for redundancy

Multi-regional

Your data is distributed redundantly across US, EU or Asia.

How to choose between regional, dual-region and multi-region Cloud Storage

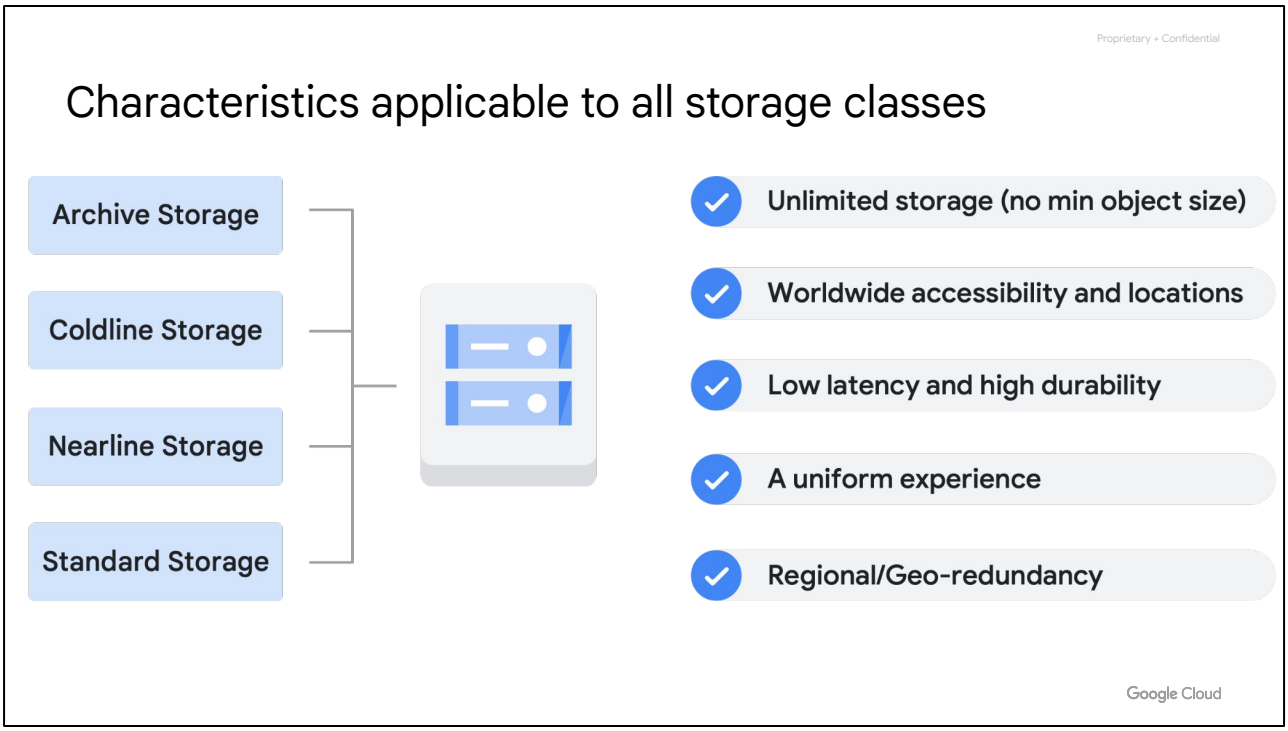
Important Note: Compliance requirements apply to ALL storage services (e.g., databases) not just Cloud Storage.

Google Cloud

Bucket locations
https://cloud.google.com/storage/docs/locations

How to migrate Cloud Storage data from multi-region to regional
https://cloud.google.com/blog/products/storage-data-transfer/multi-region-google-cloud-storage-to-regional-data-migration

How to choose between regional, dual-region and multi-region Cloud Storage
https://cloud.google.com/blog/products/storage-data-transfer/choose-between-regional-dual-region-and-multi-region-cloud-storage/



Although each of these four classes has differences, it's worth noting that several characteristics apply across all these storage classes.
These include:

- Unlimited storage with no minimum object size requirement,
- Worldwide accessibility and locations,
- Low latency and high durability,
- A uniform experience, which extends to security, tools, and APIs, and
- Geo-redundancy if data is stored in a multi-region or dual-region. So this means placing physical servers in geographically diverse data centers to protect against catastrophic events and natural disasters, and load-balancing traffic for optimal performance.

Storage classes

Storage Class	Minimum duration*	Availability SLA	Typical monthly availability	Use cases	Name for APIs and gsutil
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

*Minimum duration = if delete file before x days, still pay for x days

Google Cloud

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

Standard Storage is best for data that is frequently accessed ("hot" data) and/or stored for only brief periods of time. When used in a 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 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, such as serving website content, streaming videos, executing interactive workloads, or serving data supporting mobile and gaming applications.

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 lowered 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.

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 lowered at-rest storage costs. Coldline Storage is ideal for data you plan to read or modify at most once a quarter.

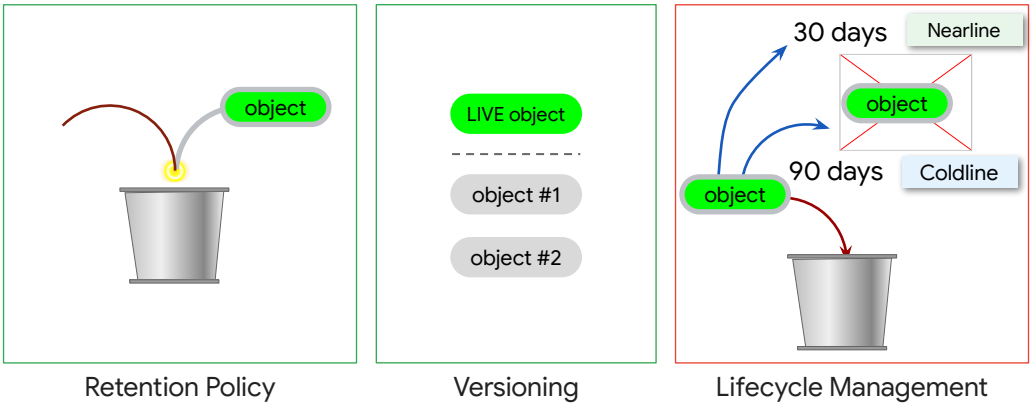
Archive Storage is the lowest-cost, highly durable storage service for data archiving, online backup, and disaster recovery. Archive Storage has higher costs for data access and operations, as well as a 365-day minimum storage duration. Archive Storage is the best choice for data that you plan to access less than once a year. For example, cold data storage, such as data stored for legal or regulatory reasons, and disaster recovery.

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

Python code to upload a file to a storage bucket

```python
from google.cloud import storage


def upload_blob(bucket_name, source_file_name, destination_blob_name):
    """Uploads a file to the bucket."""
    # bucket_name = "your-bucket-name"
    # The path to your file to upload
    # source_file_name = "local/path/to/file"
    # destination_blob_name = "storage-object-name"

    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(destination_blob_name)

    blob.upload_from_filename(source_file_name)

    print(
        f"File {source_file_name} uploaded to {destination_blob_name}."
    )
```

Cloud Storage has many object management features



Best practices for Cloud Storage cost optimization
https://cloud.google.com/blog/products/storage-data-transfer/best-practices-for-cloud-storage-cost-optimization

Cloud Storage has many object management features. For example, you can set a retention policy on all objects in the bucket. For example, the objects should be expired after 30 days.

You can also use versioning, so that multiple versions of an object are tracked and available if necessary. You might even set up lifecycle management, to automatically move objects that haven't been accessed in 30 days to Nearline and after 90 days to Coldline.

Enable retention policies/locks to help maintain data compliance

- 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, 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.

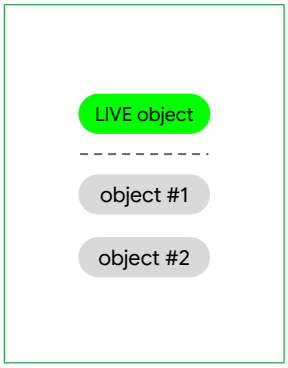
You can add a retention policy to a bucket to specify a retention period. If a bucket does not have a retention policy, you can delete or replace objects in the bucket at any time. If a bucket has a retention policy, objects in the bucket can only be deleted or replaced once their age is greater than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

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. For more information see:
https://cloud.google.com/security/compliance/sec

Enable object versioning to retain older versions of objects

- When the live version of an object is deleted or replaced, it becomes noncurrent
- If a live object version is accidentally deleted, can restore the noncurrent version back to the live version.
- Object Versioning increases storage costs
  - Can be mitigated by Lifecycle Management to delete older objects

```
gcloud storage buckets update
gs://my_test_bucket --versioning

gcloud storage buckets update
gs://my_test_bucket --no-versioning
```

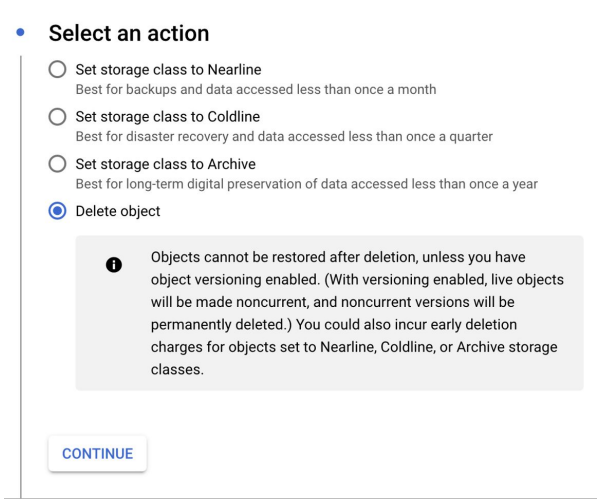


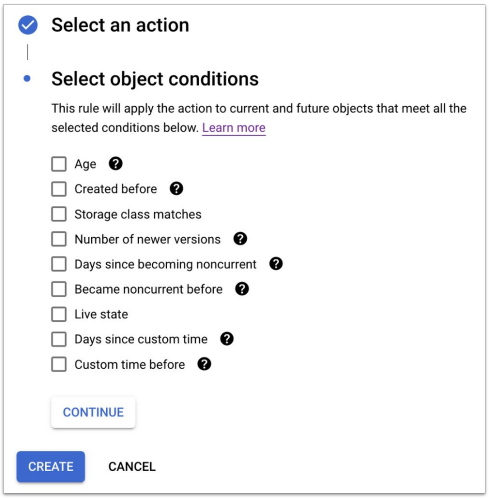
Versioning

Use Object Versioning
https://cloud.google.com/storage/docs/using-object-versioning#set-cli

Youtube video: https://www.youtube.com/watch?v=-HSnyu0_nBg

Set a lifecycle policy to delete older versions or move objects from one storage class to another





Google Cloud

Object Lifecycle Management
https://cloud.google.com/storage/docs/lifecycle

Use Signed URLs to allow access to to Cloud Storage resources for a limited time

- Signed URLs allows users without credentials to have read/write access to a file in Cloud Storage for a limited time
  - Can also enforce a file size limit on uploads
- To implement
  - Create a service account with the appropriate Cloud Storage IAM
  - Create a service account key
  - Use signurl command to create a URL that allows access to the resource

Remember to secure the key

```
#creating the key
gcloud iam service-accounts keys create ~/key.json --iam-account
example@example-project.iam.gserviceaccount.com

#allow read access to a file for 10 minutes
gsutil signurl -d 10m ~/key.json gs://super-secure-bucket/noir.jpg

#allow write access to a file for 1 hour
gsutil signurl -m PUT -d 1h -c image/jpg ~/key.json gs://super-secure-bucket/noir.jpg
```


Signed URLs
https://cloud.google.com/storage/docs/access-control/signed-urls
https://cloud.google.com/storage/docs/access-control/signing-urls-with-helpers

Sometimes, you want to programatically grant temporary access to an object in a bucket. You can do this with a signed URL. Use the gsutil signurl command as shown on the screen. The -d parameter determines how long the signed URL works.

Signed URL example output for read access

https://storage.googleapis.com/example-bucket/cat.jpeg?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=example%40example-project.iam.gserviceaccount.com%2F20181026%2Fus-central1%2Fstorage%2Fgoog4_request&X-Goog-Date=20181026T181309Z&X-Goog-Expires=900&X-Goog-SignedHeaders=host&X-Goog-Signature=247a2aa45f169edf4d187d54e7cc46e4731b1e6273242c4f4c39a1d2507a0e58706e25e3a85a7dbb891d62afa849 (more here)

- X-Goog-Algorithm: The algorithm used to sign the URL.
- X-Goog-Credential: Information about the credentials used to create the signed URL.
- X-Goog-Date: The date and time the signed URL became usable, in the ISO 8601 basic format YYYYMMDD'T'HHMMSS'Z'.
- X-Goog-Expires: The length of time the signed URL remained valid, measured in seconds from the value in X-Goog-Date. In this example the Signed URL expires in 15 minutes. The longest expiration value is 604800 seconds (7 days).

A long URL will be generated. This URL provides access to the file in storage for the duration you specified in the command.

Creating a Signed URL with Python example


```python
import datetime

from google.cloud import storage


def generate_download_signed_url_v4(bucket_name, blob_name):
    """Generates a v4 signed URL for downloading a blob.
    Note that this method requires a service account key file.
    """
    # bucket_name = 'your-bucket-name'
    # blob_name = 'your-object-name'

    storage_client = storage.Client()
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(blob_name)

    url = blob.generate_signed_url(
        version="v4",
        # This URL is valid for 15 minutes
        expiration=datetime.timedelta(minutes=15),
        # Allow GET requests using this URL.
        method="GET",
    )

    print("Generated GET signed URL:")
    print(url)
    print("You can use this URL with any user agent, for example:")
    print(f"curl '{url}'")
    return url

generate_download_signed_url_v4("bt-acme-test-data", "fish2.png")
```

Must generate a service account key and populate the GOOGLE_APPLICATION_CREDENTIALS environment variable with key path

Call the method passing in a bucket name and file name

https://storage.googleapis.com/bt-acme-test-data/fish2.png?X-Goog-Algorithm=GOOG4-RSA-SHA256&X-Goog-Credential=bucket-admin%40bt-dev-general.iam.gserviceaccount.com%2F20230906%2Fauto%2Fstorage%2Fgoog4_request&X-Goog-Date=20230906T154204Z&X-Goog-Expires=900&X-Goog-SignedHeaders=host&X-Goog-Signature=5aca5d9163c0c2d9152d524c8ae048549cdd3a6ca5a3464d883b6b5b1c71ac86fdd17d628ce1bb5dc90dd96f823ec4672c47c97a1f954f49ff7c2059731217538e44c48f87bafbe1e48b49cbdcf195ce5bde09d8e906387dabc6fafc12b973d993fbbad2601febd45ee309946c69f8b129db8eebdde9d84b6ffc84c3be9438f1c683add09161528bcc38119d7b74e4bf0f68c9fe7ed8d1884b77a1d8e4ef141743f7c794be0ddf6053232633984a49290b6a080d974f9078fddf7db03ae76e3e1fe1f41d8361bda7e734665a86b4d35d4ee9114ad11b20cc65ba8a6f68f4cc974bd715a329b7016ed5662f564942c7d540b149fdb4c6c7a667d874a3f0a1e367

Example output

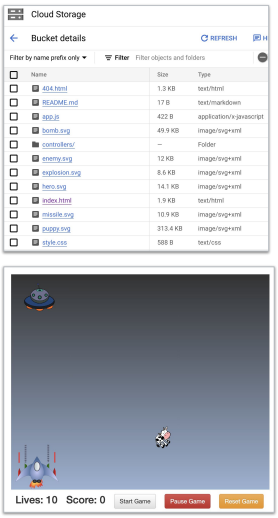
https://cloud.google.com/storage/docs/samples/storage-generate-signed-url-v4#storage_generate_signed_url_v4-python

Benefits of Signed URLs

- Provide a way to give temporary scoped access to an object in cloud storage
- Relatively easy to consume
  - They contain all the authentication information in their query string
    - No OAuth handshake or auth token retrieval is needed
- Users interact with directly with Cloud Storage
  - Frees up bandwidth on web servers as they no longer need to handle this task

Using Cloud Storage to host static websites

- Storage buckets can host static web pages containing client-side technologies such as HTML, CSS, and JavaScript
  - Cannot serve dynamic content such as server-side scripts like PHP
- Create an external HTTPS load balancer with a static IP and a backend pointing to the bucket
- Enable Cloud CDN for the best performance for your end users



Google Cloud

The Cross-Origin Resource Sharing (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 is serving CORS headers so that content on storage.googleapis.com is allowed access. It is recommended that you dedicate a specific bucket for this data access.

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

Security review: Secure your buckets using the following options

- Use Identity and Access Management (IAM) roles to grant:
  - Permissions at the bucket level (best practice)
- Use Access Control Lists (ACLs) to grant:
  - Read or owner access to individual objects
    - Use when fine-grained access is required
- Use Signed URLs to grant:
  - Time-limited read or write access to an object through a generated URL

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

You can use Cloud Identity and access Management (Cloud IAM) permissions to grant access to buckets and to provide bulk access to a bucket's objects. Cloud IAM permissions do not give you fine-grained control over individual objects. For more information, see:
https://cloud.google.com/storage/docs/access-control/using-iam-permissions

You can use Access Control Lists (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. For more information, see:
https://cloud.google.com/storage/docs/access-control/create-manage-lists

Use signed URLs (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 gsutil or programmatically with your application. For more information, see:
Create a signed URL using gsutil:
https://cloud.google.com/storage/docs/access-control/create-signed-urls-gsutil
Create a signed URL programmatically:
https://cloud.google.com/storage/docs/access-control/create-signed-urls-program

Cloud Storage error handling

- Design applications to handle network failures with truncated exponential backoff
  - 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.

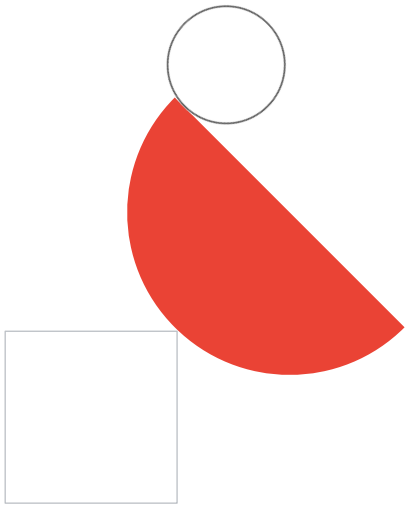
Understanding how truncated exponential backoff works is important if you are building client applications that use the Cloud Storage XML API or JSON API directly, accessing Cloud Storage through a client library (some client libraries, such as the Cloud Storage Client Library for Node.js, have built-in exponential backoff), or using the gsutil command line tool (has configurable retry handling).

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

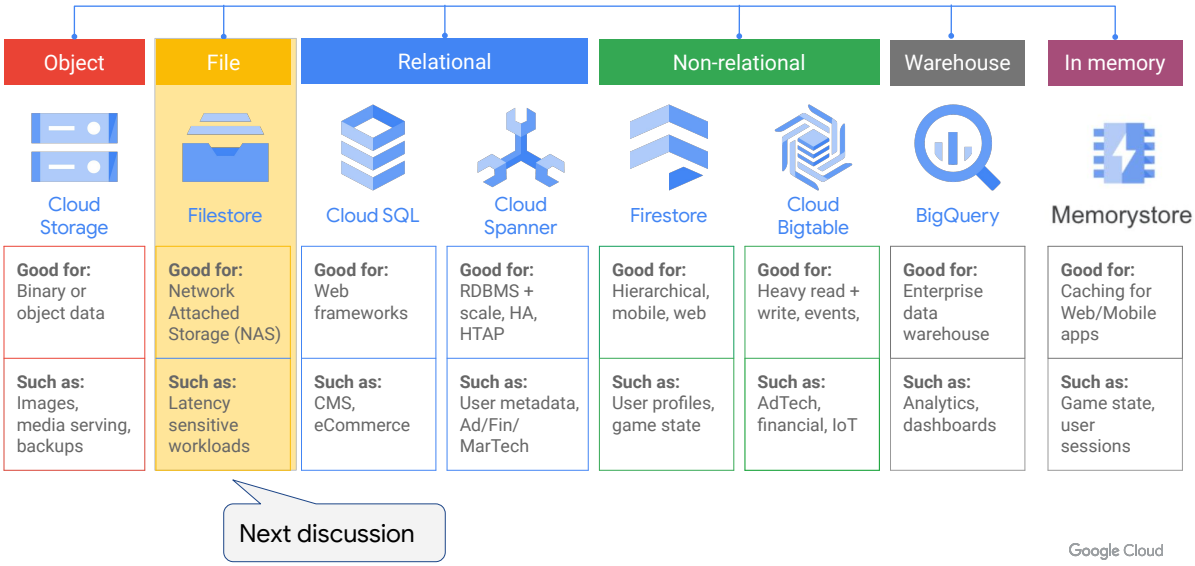
The example shows a backoff implementation for the Python retry library when you retry distributed services and other remote endpoints.

For more information, see: https://cloud.google.com/storage/docs/exponential-backoff

Filestore



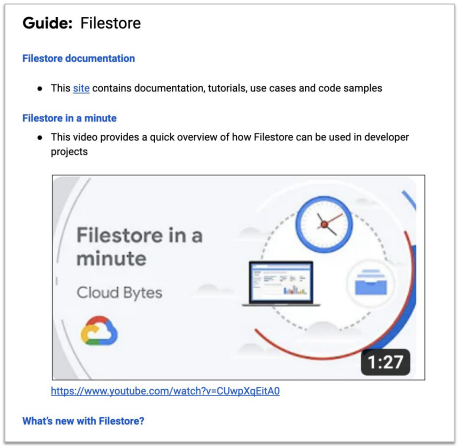
Storage and database services



Guide: Filestore

Topics covered include

- Documentation with links to use cases, tutorials and code samples
- Youtube videos discussing features and how to create an instance and mount it to Compute Engine
- Recommended hands-on lab



Cloud Filestore

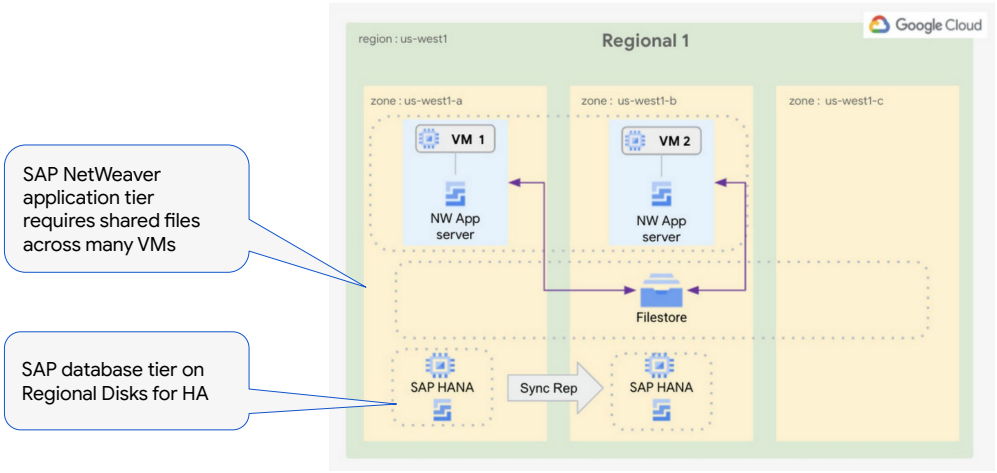
What is Filestore?

- Fully managed file system for Compute Engine and Kubernetes Engine
  - Network attached storage (NAS)
  - Mounted as file shares on Compute Engine instances
  - Used to store and serve files such as documents, images, videos, audio files, and other data
  - Pay for what you use
  - Capacity scales automatically scale based on demand

YouTube video: https://www.youtube.com/watch?v=CUwpXqEitA0

Filestore
https://cloud.google.com/filestore

Filestore example - SAP NetWeaver

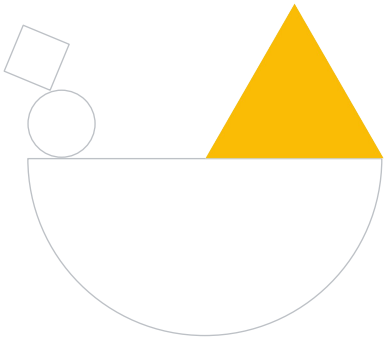


Announcing Filestore Enterprise, for your most demanding apps (2021)

Announcing Filestore Enterprise, for your most demanding apps(2021)
https://cloud.google.com/blog/products/storage-data-transfer/google-cloud-announces-filestore-enterprise-for-business-critical-apps

Note that this blog is from 2021, and the “public preview” items mentioned in the blog are now generally available (GA)

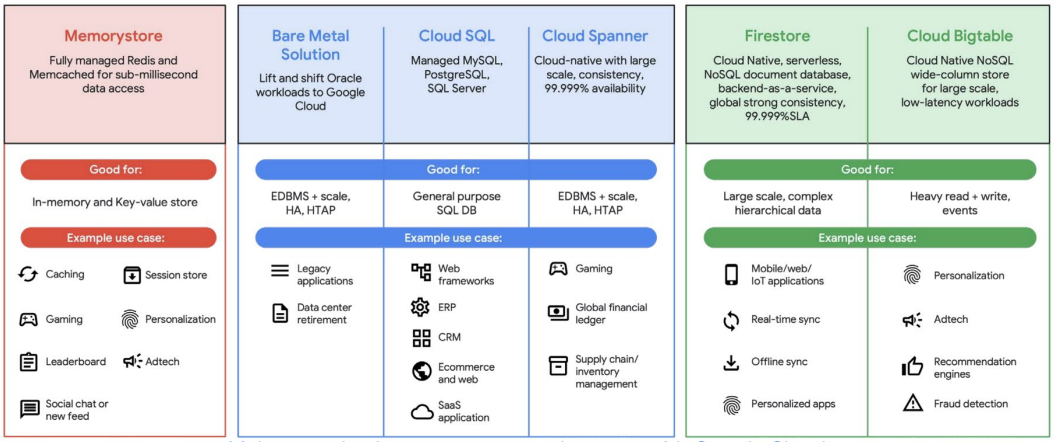
Database Overview



Database overview

- This section provides overviews of the Google Cloud database services and the key points mentioned in the Professional Cloud Developer Exam Guide
  - This is not a detailed discussion on how to use each of these databases
- To learn more about the individual services, go to the documentation
  - Cloud SQL
  - Cloud Spanner
  - Cloud Firestore
  - Cloud Bigtable
  - BigQuery
  - Cloud Memorystore

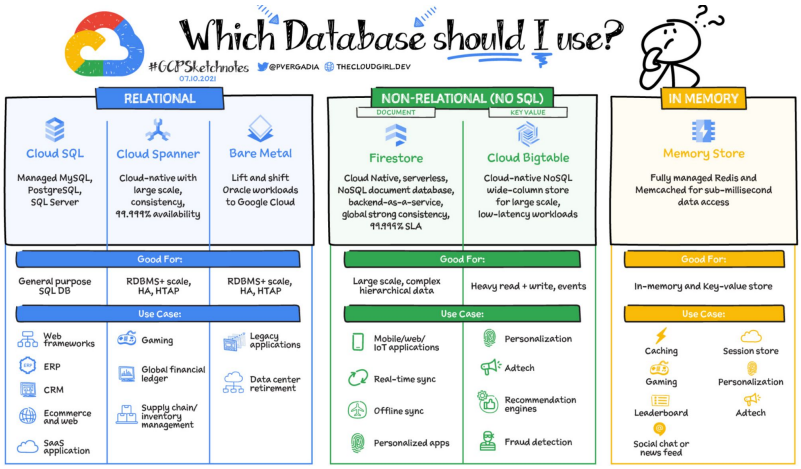
Which database should I use?



Make your database your secret advantage with Google Cloud

Source:
https://services.google.com/fh/files/misc/guide_to_google_cloud_databases.pdf

Database Options - another overview



[Your Google Cloud database options, explained]

Relational vs non-relational databases

Relational databases store information in tables, rows and columns that structure the data. They use relational semantics (i.e. a column in one table can point to data in another table) to ensure data consistency and enable complex queries across multiple tables. Relational databases are used when the structure of the data doesn't change often, such as in banking or supply chain inventory management.

- Cloud SQL
- Cloud Spanner

Non-relational databases (or NoSQL databases) store complex, unstructured data in a non-tabular form such as documents and key-value stores. Non-relational databases are often used when large quantities of complex and diverse data need to be organized, or where the structure of the data is regularly evolving to meet new business requirements, such as personalization and web and mobile applications.

- Cloud Bigtable
- Cloud Firestore

From: Make your database your secret advantage with Google Cloud

What does data in a relational database look like?

- Tables contain fields, indexes, and constraints
- Primary key ensures each row in a table is unique
- Relationships are constraints that ensure a parent row cannot be deleted if there are child rows in another table

Customers

CID	First	Last
12442	Bill	Raymond
12535	Lee	Ying

1 ∞

Orders

OID	CID	OrderDate
2999	12442	20230405
3000	12442	20230105
3001	12535	20230306

1 ∞

OrderDetails

ODID	OID	Item
1000	2999	AFX-3
1001	2999	VKW-k
1003	3000	CST-T

This is a brief (and very, very simplistic) overview of a relational database.

What does data in a NoSQL database look like?

- Depends on the type of database

Firestore

Documents live in collections, which are simply containers for documents. For example, you could have a `users` collection to contain your various users, each represented by a document:

users

alovelace

```
first : "Ada"
last : "Lovelace"
born : 1815
```

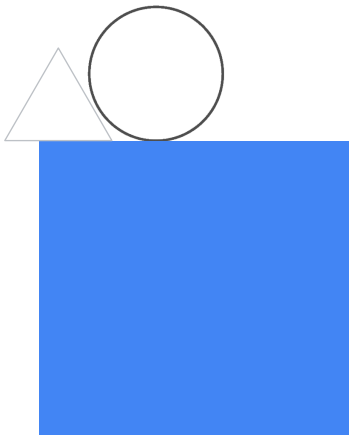
aturing

```
first : "Alan"
last : "Turing"
born : 1912
```

Bigtable

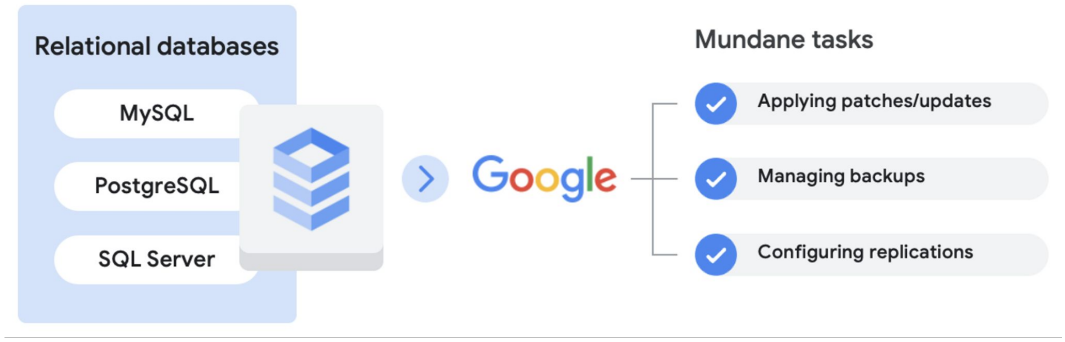
	Flight_Information				
Row Key	Origin	Destination	Departure	Arrival	Passengers
ATL#arrival#20190321-1121	ATL	LON	20190321-0311	20190321-1121	158
ATL#arrival#20190321-1201	ATL	MEX	20190321-0821	20190321-1201	187
ATL#arrival#20190321-1716	ATL	YVR	20190321-1014	20190321-1716	201

Cloud SQL



Cloud SQL

A fully managed, cloud-based alternative to on-premise MySQL, PostgreSQL, and SQL Server databases



The business value of Cloud SQL: how companies speed up deployments, lower costs and boost agility

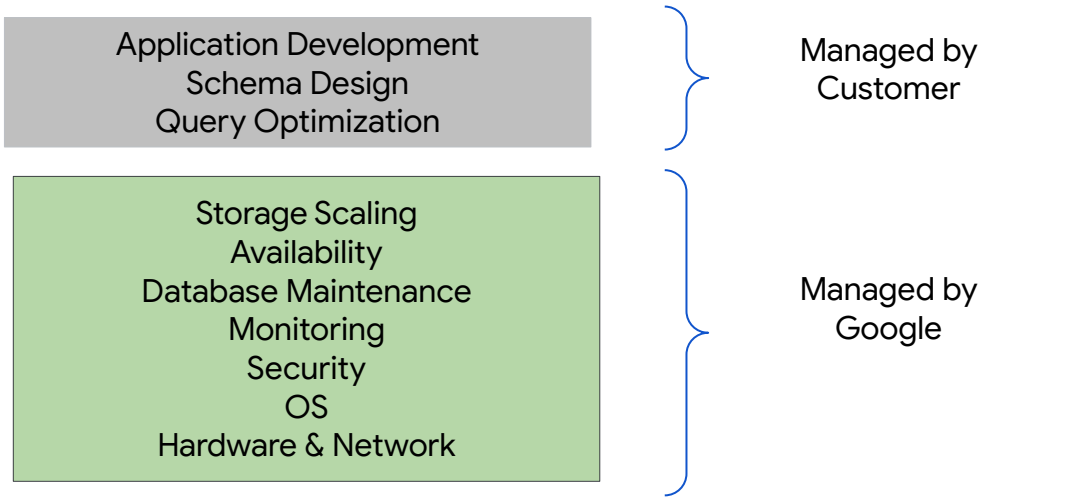
The business value of Cloud SQL: how companies speed up deployments, lower costs and boost agility
https://cloud.google.com/blog/products/databases/the-business-value-of-cloud-sql/

Cloud SQL:
https://cloud.google.com/sql/docs/mysql/introduction

What is Cloud SQL
https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-sql

Cloud SQL offers fully managed relational databases, including MySQL, PostgreSQL, and SQL Server as a service. It's designed to hand off mundane, but necessary and often time-consuming, tasks to Google—like applying patches and updates, managing backups, and configuring replications—so your focus can be on building great applications.

Cloud SQL - shared responsibilities



Google Cloud manages the infrastructure work while the customer focuses on their goals.

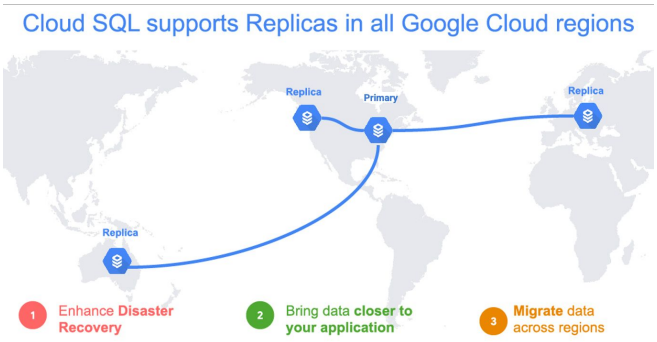
Scaling Cloud SQL Databases

- Cloud SQL can scale in the following ways
  - **Vertical scaling:**
    - Increasing the amount of computing resources (such as CPU and memory) allocated to a single database instance.
    - Can be done with a few clicks in the Cloud Console, and requires no downtime.
  - **Horizontal scaling:**
    - Adding additional read replicas to distribute read workloads and improve performance.
    - Can be added on demand, and there's no limit to the number of replicas that can be added.

Cloud SQL Read Replicas

Fully managed read replica in a different region(s) than that of the primary instance

- Enhance DR
- Bring data closer to your applications (performance and cost implications)
- Migrate data across regions
- Data and other changes on the primary instance are updated in almost real time on the read replicas



Cloud SQL Read Replicas:
https://cloud.google.com/sql/docs/mysql/replication

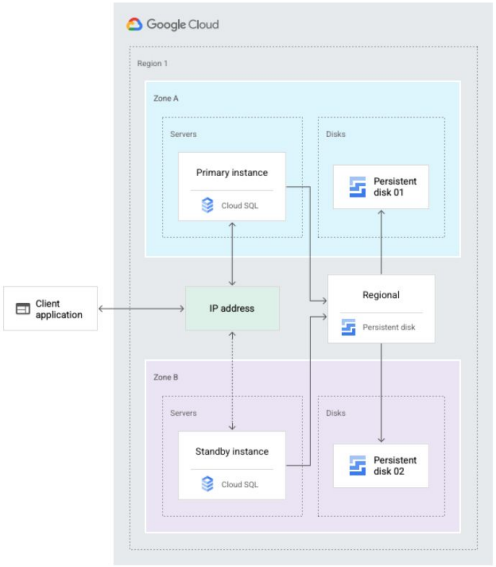
Cloud SQL read replicas provide

- Enhanced Disaster recovery. Can convert a read replica to the Primary (if the primary is no longer available)
- In terms of data closer to your app, you can have different connection strings for an application for different endpoints
  - For example,:
    - Read-Write connection strings will go to the primary
    - Read only connections hit a local read replica
- If reads comprise most of the use case for your app, then accessing the local replica saves on data transfer and egress charges since you are not accessing the primary sitting in the eastern United States in this example

Introducing cross-region replica for Cloud SQL (June 2, 2020)
https://cloud.google.com/blog/products/databases/introducing-cross-region-replica-for-cloud-sql

Cloud SQL - High Availability

- Provides automatic failover if a zone or instance become unavailable
- The primary instance is in one zone in a region
  - The failover instance is in another zone
- Synchronous replication is used to copy all writes from the primary disk to the replica



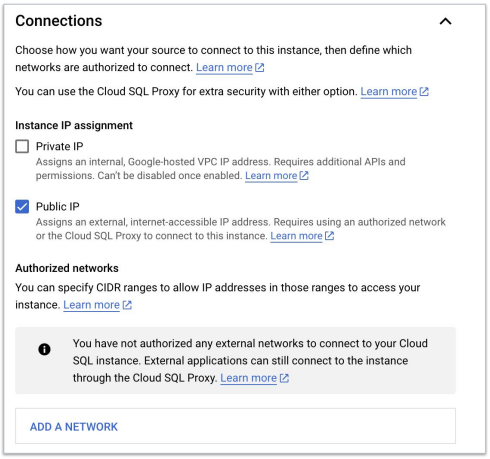
https://cloud.google.com/sql/docs/mysql/high-availability

Cloud SQL supports a variety of use cases

- Storing and managing relational data such as customer information, product inventory, and financial transactions
- Backend database for web and mobile applications
- Migrating on-premises databases to the cloud
- Building and deploying data-driven applications with minimal setup and management overhead
- Replicating data for disaster recovery and high availability
    - Provides automatic failover for high availability, ensuring that databases are always available in case of an unexpected outage or hardware failure

Cloud SQL supports both public and private IP connectivity options

- Can have one or the other or both
- Public IP
  - Access blocked by default
    - Must specify IP ranges (“authorized networks”) that are allowed access
    - Alternatively can use the Cloud SQL Auth Proxy
      - Provides secure access to instances without a need for Authorized Networks
      - Encrypts all traffic using TLS
- Private IP
  - Assigned an internal IP address
  - Allows resources in the VPC network to connect to Cloud SQL instance



Using Private IP

- Used when clients are within the same VPC network that hosts Cloud SQL or connected to the VPC network via a hybrid connection
  - Uses Private Service Access
    - Routing method within Google Cloud that allows workloads in the VPC to access Google-managed infrastructure services privately

Configuring Private Service Access is typically done by a network admin.

No further discussion of the process is provided in this content

Connecting applications to Cloud SQL

- Cloud SQL Auth Proxy
  - Open source library distributed as an executable binary
  - Acts as an intermediary server that listens for incoming connections, wraps them in SSL/TLS, and then passes them to a Cloud SQL instance.
  - Validates connections using credentials for a user or service account
- Cloud SQL Language Connectors
  - Client libraries that provide encryption and IAM authorization when connecting to a Cloud SQL instance
  - Currently available for Java, Python, Go and Node.js
  - Provide the same authentication as the Cloud SQL Auth Proxy without requiring an external process

Using the Cloud SQL Auth Proxy

- Connector that provides secure access to instances without a need for Authorized networks or for configuring SSL
  - Provides automatic encryption traffic to and from the database using TLS 1.3 with a 256-bit AES cipher
    - SSL certificates are provided to verify client and server identities
  - Uses IAM permissions to control who and what can connect to your Cloud SQL instances
    - Handles authentication with Cloud SQL, removing the need to provide static IP addresses

Using the Cloud SQL Auth Proxy

- For example code, see these sites
  - Connect to Cloud SQL for MySQL from Compute Engine
  - Connect from Compute Engine

Using the Cloud SQL Language Connectors

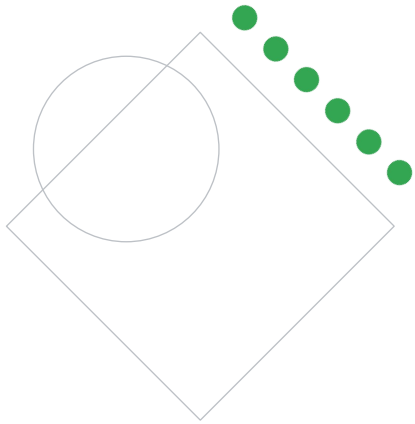
- Libraries that provide encryption and IAM authorization when connecting to a Cloud SQL instance
- A client-side component connects to a proxy server on the Cloud SQL instance
  - Creates a temporary certificate that authorizes the holder to connect to the server-side proxy.
  - The server-side proxy limits access to the Cloud SQL database by requiring a valid TLS certificate in order to connect
- Supported languages
  - Cloud SQL Java connector
  - Cloud SQL Python connector
  - Cloud SQL Go connector
  - Cloud SQL Node.js connector (Preview)

Example using the Python Cloud SQL Language Connectors library

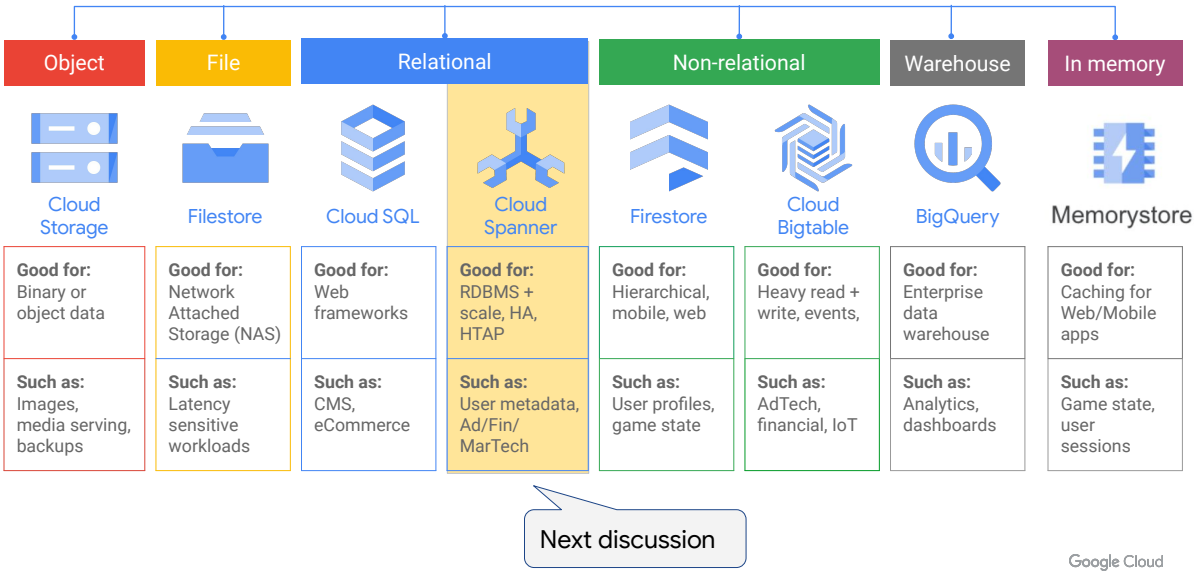
```python
def connect_with_connector() -> sqlalchemy.engine.base.Engine:
    """
    Initializes a connection pool for a Cloud SQL instance of MySQL.

    Uses the Cloud SQL Python Connector package.
    """
    # Note: Saving credentials in environment variables is convenient, but not
    # secure - consider a more secure solution such as
    # Cloud Secret Manager (https://cloud.google.com/secret-manager) to help
    # keep secrets safe.

    instance_connection_name = os.environ[
        "INSTANCE_CONNECTION_NAME"
    ]  # e.g. 'project:region:instance'
    db_user = os.environ["DB_USER"]  # e.g. 'my-db-user'
    db_pass = os.environ["DB_PASS"]  # e.g. 'my-db-password'
    db_name = os.environ["DB_NAME"]  # e.g. 'my-database'

    ip_type = IPTypes.PRIVATE if os.environ.get("PRIVATE_IP") else IPTypes.PUBLIC

    connector = Connector(ip_type)

    def getconn() -> pymysql.connections.Connection:
        conn: pymysql.connections.Connection = connector.connect(
            instance_connection_name,
            "pymysql",
            user=db_user,
            password=db_pass,
            db=db_name,
        )
        return conn

    pool = sqlalchemy.create_engine(
        "mysql+pymysql://",
        creator=getconn,
        # ...
    )
    return pool
```


From:
https://cloud.google.com/sql/docs/mysql/connect-connectors#python

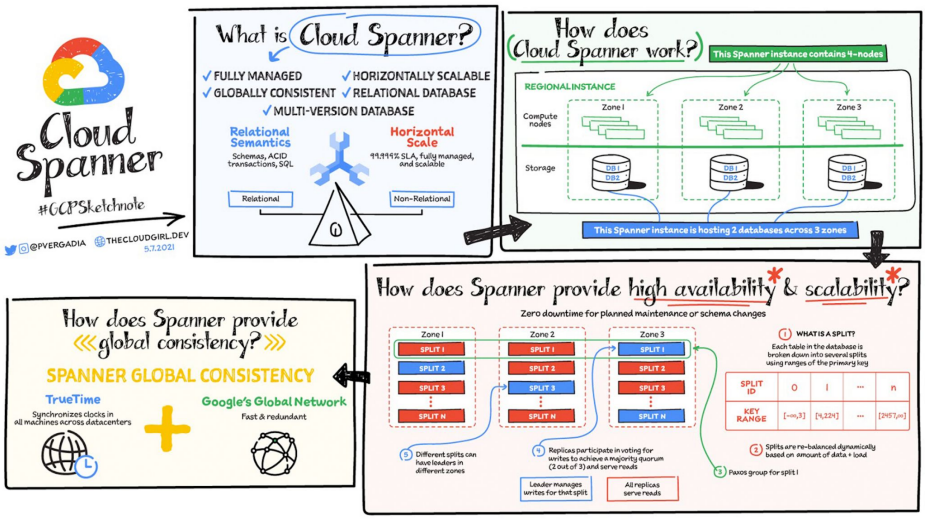
Cloud Spanner



Storage and database services



What is Cloud Spanner?

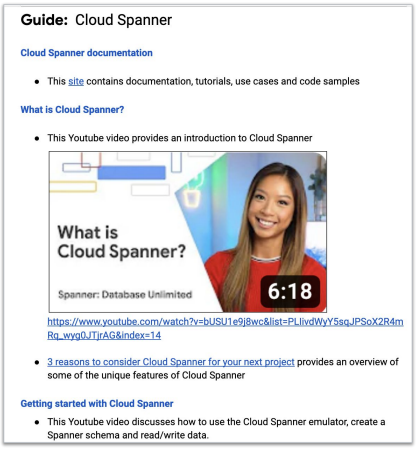


What is Cloud Spanner?

Guide: Cloud Spanner

Topics covered include

- Documentation with links to use cases, tutorials, and more
- How to use the Cloud Spanner emulator
- Schema design best practices
- Strong vs stale reads
- Code samples
- Recommended hands-on lab



Cloud Spanner

Cloud Spanner

A enterprise-grade, globally distributed, externally consistent relational database having unlimited scalability and industry-leading 99.999% availability

- Powers Google's most popular, globally available products, like YouTube, Drive, and Gmail
- Capable of processing more than 1 billion queries per second at peak
- For any workload, large or small, that cannot tolerate downtime, and requires high availability
- Regional and multi-regional deployments
  - SLA: Multi-regional: 99.999%
  - SLA: Regional: 99.99%
- Supports ANSI standard SQL

Cloud Spanner myths busted

Cloud Spanner:
https://cloud.google.com/blog/topics/developers-practitioners/what-cloud-spanner
https://cloud.google.com/spanner

Cloud Spanner myths busted:
https://cloud.google.com/blog/products/databases/cloud-spanner-myths-busted

Cloud Spanner is an enterprise-grade, globally distributed, externally consistent database that offers unlimited scalability and industry-leading 99.999% availability. It requires no maintenance windows and combines the benefits of relational databases with the unmatched scalability and availability of non-relational databases.

Spanner powers Google's most popular, globally available products, like YouTube, Drive, and Gmail, and it can processes more than 1 Billion queries per second at peak

But don't be mislead into thinking that Spanner is only for large, enterprise level applications. Many customers use Spanner for their smaller workloads (both in terms of transactions per second and storage size) for availability and scalability reasons. Spanner is appropriate for any customer that cannot tolerate downtime, and needs high availability for their applications

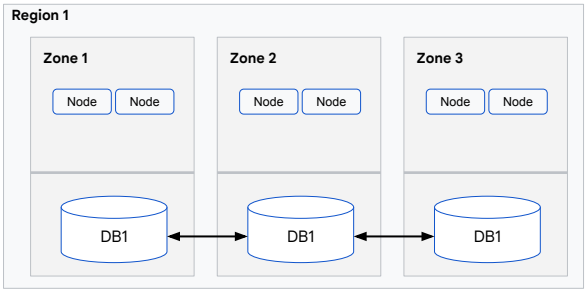
Limitless scalability and high availability is critical in many industry verticals such as gaming and retail, especially when a newly launched game goes viral and becomes

an overnight success or when a retailer has to handle a sudden surge in traffic due to a Black Friday/Cyber Monday sale.

Spanner offers the flexibility to interact with the database via a SQL dialect based on ANSI 2011 standard as well as via a REST or gRPC API interface, which are optimized for performance and ease-of-use. In addition to Spanner's interface, there is a PostgreSQL interface for Spanner, that leverages the ubiquity of PostgreSQL and provides development teams with an interface that they are familiar with. The PostgreSQL interface provides a rich subset of the open-source PostgreSQL SQL dialect, including common query syntax, functions, and operators. It also supports a core collection of open-source PostgreSQL data types, DDL syntax, and information schema views. You get the PostgreSQL familiarity, and relational semantics at Spanner scale.

Spanner regional deployment

- Regional deployment - 99.99 SLA
  - Use when users and services are located within a single region for the lowest-latency reads and writes
  - Create one or more nodes, which are replicated across zones
    - Nodes represent compute capacity and storage



- Nodes are separate from storage and are replicated across zones in a region
- Serve the read/write requests to the storage located within their zones

- Data is stored separately from the nodes and is also replicated across zones
- All instances are read/write in a regional deployment
- When data is written to one instance automatic synchronous replication occurs to the other instances

Multi-regional deployment

- Spanner deployed across 3 or more multiple regions provides 99.999% SLA
  - Replication occurs between the regions as changes are made with immediate consistency
- Benefits
  - High Availability
    - Database remains available even if an entire region experiences an outage (99.999% SLA)
  - Geographic locality
    - Serve data from a location closer to the user
      - Reduces latency and improves user experience
      - Meet data residency requirements where required, e.g., GDPR
  - Load Distribution
    - If have high read and write loads from various global locations, distributing this across multiple regions can help in achieving optimal performance

Spanner supports two types of reads with multi-regional deployments

- Strong reads
  - Used when the absolute latest value needs to be read
  - Default
- Stale reads
  - Allows an applications to read data quickly, even if it may not be the most recent version
  - May be more important for this application's use case if it prioritizes latency over consistency
    - The client does not request the absolute latest version, just the data that is most recent (e.g. up to x seconds old)

In multi-region deployments some nodes are read/write; some are read only

- When a write occurs, it is synchronously replicated to other regions before the write is acknowledged
  - Once acknowledged, it is immediately visible and consistent across all replicas, regardless of their location
- Read-only replicas maintain a full copy of the data, which is replicated from read-write replicas
  - Can specify the maximum staleness of the data you're willing to accept when reading from a read-only replica,
    - This means you can read slightly outdated data (by a matter of seconds) if you're willing to trade off absolute freshness for reduced read latency.

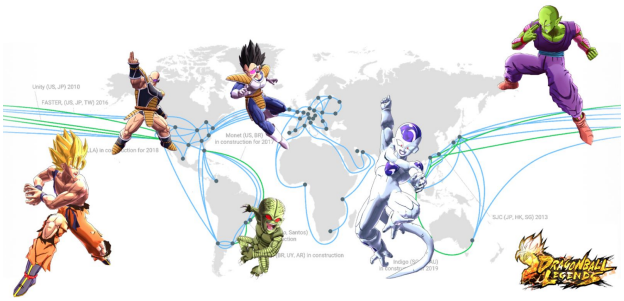
Python example - strong vs stale read

```
def read_data(instance_id, database_id):
    """Reads sample data from the database."""

    spanner_client = spanner.Client()
    instance = spanner_client.instance(instance_id)
    database = instance.database(database_id)

    with database.snapshot() as snapshot:
        keyset = spanner.KeySet(all_=True)
        results = snapshot.read(
            table="Albums",
            columns=("SingerId", "AlbumId",
               "AlbumTitle"),
            keyset=keyset
        )

        for row in results:
            print("SingerId: {}, AlbumId: {},
               AlbumTitle: {}".format(*row))
```

```
def read_stale_data(instance_id, database_id):
    """Reads sample data from the database. The data
     is 15 seconds stale."""
    import datetime

    spanner_client = spanner.Client()
    instance = spanner_client.instance(instance_id)
    database = instance.database(database_id)
    staleness = datetime.timedelta(seconds=15)

    with database.snapshot(exact_staleness=staleness)
      as snapshot:
        keyset = spanner.KeySet(all_=True)
        results = snapshot.read(
            table="Albums",
            columns=("SingerId", "AlbumId",
               "AlbumTitle"),
            keyset=keyset,
        )

        for row in results:
            print("SingerId: {}, AlbumId: {},
               AlbumTitle: {}".format(*row))
```

Cloud Spanner - customer use case

- Dragon Ball Legends - mobile game from Bandai Namco Entertainment
- Requirements were:
  - Global backend that could scale with millions of players and still perform well.
  - Global reliable, low latency network to support multi-region player-versus-player battles
  - Real-time data analytics to measure and evaluate how people are playing the game and adjust it on-the-fly.

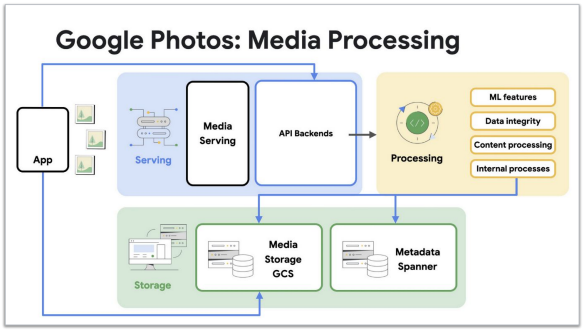


Behind the scenes with the Dragon Ball Legends GC backend

This is another example where multiple Google Cloud services are used together to provide a solution.

Another Cloud Spanner use case - Google Photos

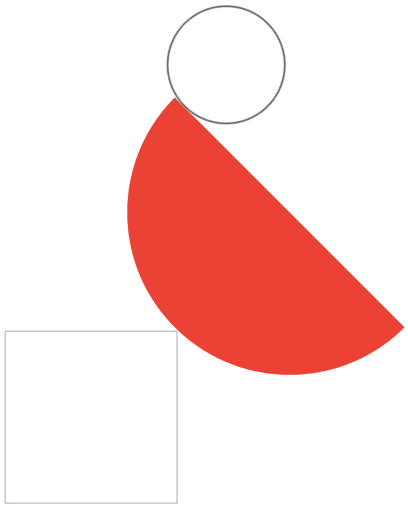
- Google Photos needs storage for more than a billion users having four trillion photos and videos
  - Must also protect personal, private, and sensitive user data
- Needed a database solution that is
  - Highly scalable, reliable, secure
  - Supports large scale data processing workloads conducive to AI/ML applications.



The big picture: How Google Photos scaled rapidly on Spanner

This is another example where multiple Google Cloud services are used together to provide a solution.

Firebase Overview



What is Firebase?

- Firebase is Google's client side app development platform
  - Integrates with many other Google tools such as Analytics and Ads
  - Provides multiple client side SDKs
  - Provides built in authentication, plus security rules to protect user data
  - SDKs provide access to Cloud Storage, Cloud Firestore and Cloud Functions

Supported platforms, frameworks, libraries, and tools

Provides client side SDKs for
iOS (Swift), Android (Java, and Kotlin), Web (JavaScript, TypeScript plus Frameworks such as Angular, React, and Vue), C++ and Unity for game developers, and Flutter

Flutter - Build, test, and deploy beautiful mobile, web, desktop, and embedded apps from a single codebase
https://flutter.dev/

A little history

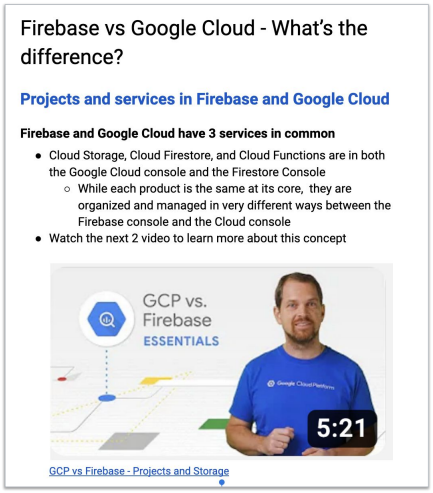
- **2011**: Firebase was a company providing a real-time backend service for building web and mobile applications
- **2012**: Firebase launched its flagship product, Firebase Realtime Database
- **2014**: Google acquired Firebase and began integrating it into its Google Cloud services
- **2016**: Google launched an expanded suite of Firebase services, transforming it into a unified app development platform that includes products for app analytics, messaging, and crash reporting, among others
- **2017**: Google launched Cloud Firestore, a new database within Firebase, designed to offer more scalability and flexibility than Firebase Realtime Database

https://firebase.google.com/

Firebase and Google Cloud

This guide provides a overview of the products and services that are used in both Firebase and Google Cloud including:

- Cloud Storage
- Cloud Functions
- Cloud Firestore

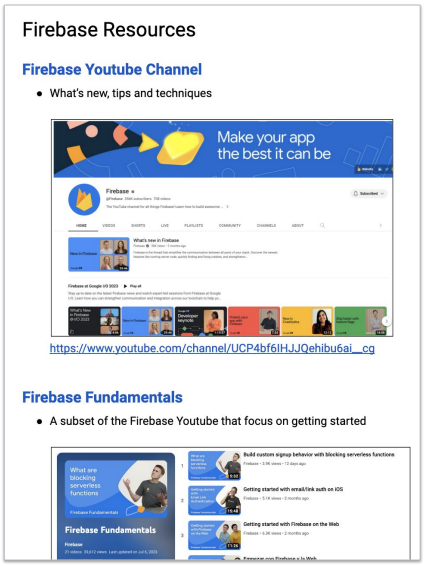


Firebase vs Google Cloud - What's the difference?

Firebase resources

This guide provides links to various Firebase resources including

- Firebase Youtube Channel
- Firebase Fundamentals Youtube playlist
- Firebase sample applications
  - Sample code for mobile and web applications
- Firebase GitHub
  - Quickstarts, plug-ins, SDKs for different languages, documentation and more
- Firebase Solutions Portal
  - Tutorials and documentation

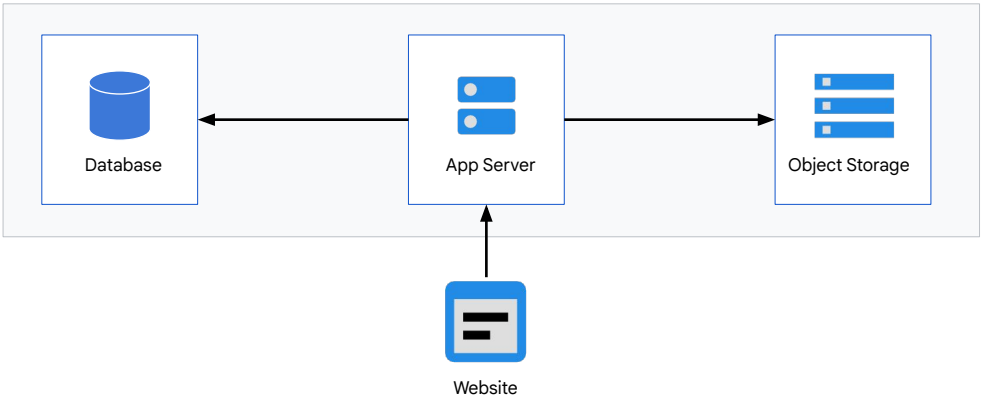


Firebase Resources

Firebase versus traditional web server architecture

Traditional server based architecture

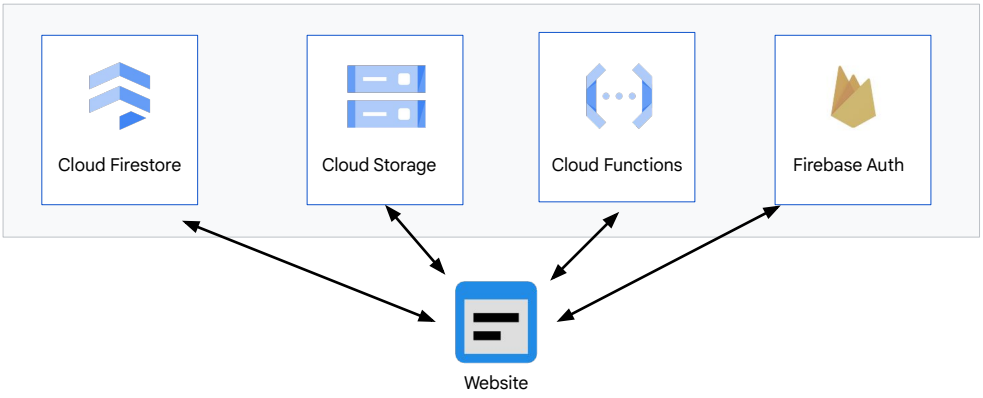
- Server acts as a gateway
  - Application goes through this to get to the storage layer



Firebase architecture

- Client aps talk directly to the backends in Google Cloud
- Using serverless architectures means developers don't have to managed the backend services
  - Fully managed by Google
- Clients are authenticated by Firebase authentication
- Firebase authorization defines what clients can do once authenticated

Upcoming topic



Firebase Authentication vs Authorization

Authentication

What is the application?

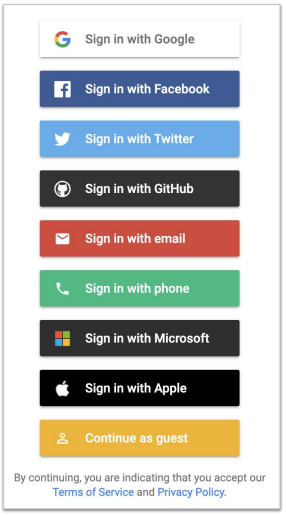
Authorization

What is it allowed to do?

Firebase authentication

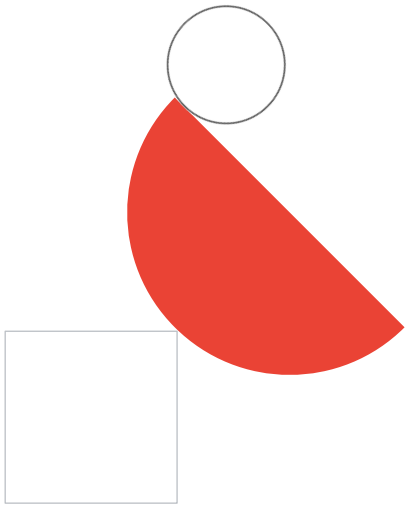
Slide from Module 3

- Multiple options
  - Username/password
  - Sign-in links (No password needed)
    - User enters email address and a link to sign-in is sent to it
    - Email contains a one-time, short-lived authentication code
    - Clicking the link takes the user to a login page where the auth code is exchanged for a normal, long-lived auth token
  - Phone number authentication
    - The authentication code is to the user's phone via SMS
  - “Social” logins with Federated Identity Providers
    - Sign in with Google, Microsoft, Facebook, etc.



Firebase Overview

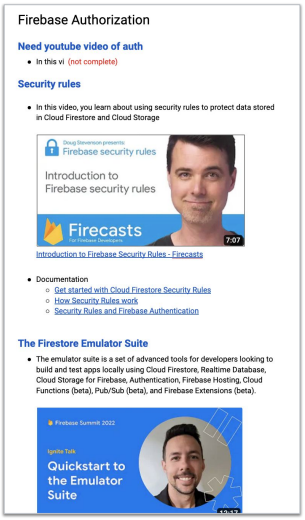
- **Authorization**
- Emulator



Guide: Firebase Authorization

This guide discusses Firebase Authorization and provides an overview of

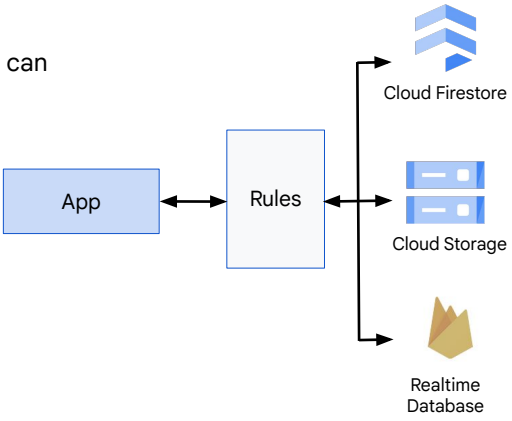
- Security rules
- Using the Firestore Emulator Suite for local testing



Firebase Authorization

Firebase security rules

- Firebase security rules limit the resources a client can access
  - Secures data in
    - Cloud Firestore
    - Cloud Storage
    - Firebase Realtime Database



Firestore security rules example

- Example of
  - Allowing any authenticated user to read document
  - Restricting document writes to documents tagged with their UID

`Service` declaration declares which product is being used

```
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{uid} {
      allow read;
      allow write: if request.auth.uid == uid;
    }
  }
}
```

Within the `match` pattern are `allow` expressions

Users can write to documents tagged with their user UID

Introduction to Firebase Security Rules - Firecasts
https://www.youtube.com/watch?v=QEuu9X9L-MU

Another Firestore example

- Example of
  - Allowing authenticated users to read/write/delete their own data
  - Allowing any authenticated user to create a new document

```
service cloud.firestore {
  match /databases/{database}/documents {
    // Make sure the uid of the requesting user matches name of the user
    // document. The wildcard expression {userId} makes the userId variable
    // available in rules.
    match /users/{userId} {
      allow read, update, delete: if request.auth != null && request.auth.uid == uid;
      allow create: if request.auth != null;
    }
  }
}
```

Any authenticated user can create documents

Authenticated users may read, update and delete their own documents that are tagged with their user UID

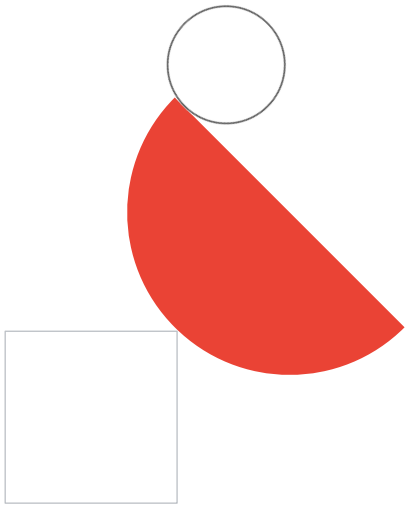
Cloud Storage example

- Example of
  - Allowing any authenticated user to read files
  - Restricting file writes to documents tagged with their UID

```
service firebase.storage {
 // Only a user can upload their file, but anyone can view it
 match /users/{userId}/{fileName} {
   allow read;
   allow write: if request.auth != null && request.auth.uid == userId;
 }
}
```

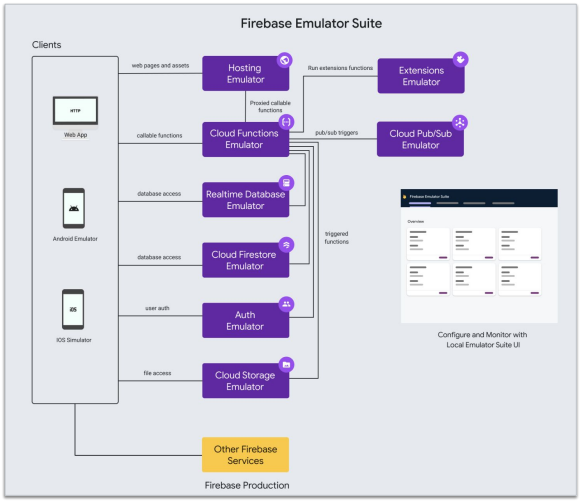
Firebase Overview

- Authorization
- **Emulator**



Firebase Local Emulator Suite

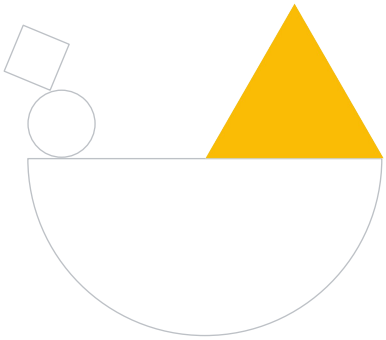
- Consists of individual service emulators built to accurately mimic the behavior of Firebase services
  - Built for accuracy, not performance or security



Introduction to Firebase Local Emulator Suite

Firebase Database Options

- **Overview**
- Firestore in Native Mode
- Firestore in Datastore mode



Firebase offers two NoSQL database products

Firebase Realtime Database vs Cloud Firestore

	Firebase Realtime Database	**Google Cloud Firestore**
Description	Original database	Introduced after Google acquisition
Real-time and Offline Support	Offers offline support for mobile clients	Offers offline support for mobile clients and web clients
Data format	Stores data as JSON	Organizes data into collections and documents, which can make it easier to structure and query data
Storage location	Google Cloud (visible only from the Firebase Console)	Google Cloud (visible from the Cloud Console and the Firebase Console)
Queries	Has some limitations, such as a lack of complex querying and indexing capabilities	Offers more advanced features, such as transaction support, richer querying capabilities, and automatic scaling

While the Realtime Database is stored in Google Cloud, the only way to access it is from the Firebase Console.

Google Cloud Firestore is accessible from both the Firebase Console and the Google Cloud Console.

Firebase Realtime Database vs Cloud Firestore (continued)

	Firebase Realtime Database	**Google Cloud Firestore**
Scalability	Requires sharding to scale. It can scale to about 200,000 concurrent connections and 1,000 writes/second. For scaling beyond this, data sharding across multiple databases is required.	Scales automatically. Currently, scaling is limited to approx 1 million concurrent connections and 10,000 writes/second. These limits are planned to be increased in the future.
Reliability and Performance	Deployed to a single Google Cloud region. It has very low latency and is ideal for frequent state sync.	Multi-region solution that automatically scales. Data is stored in multiple zones in different regions to ensure global scalability and reliability.
When to use	Certain use cases that require the lowest possible latency	Recommended for new projects due to its more advanced features and scalability

[Choose a Database: Cloud Firestore or Realtime Database]

Cloud Firestore has two modes*

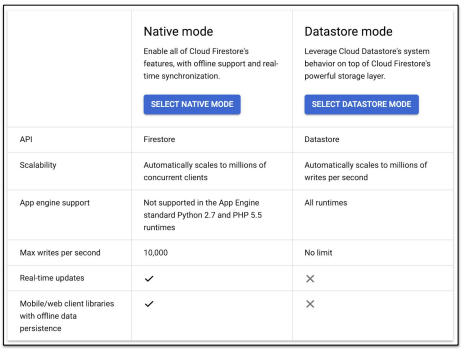
	Firestore in native-mode	**Firestore in Datastore mode**
Description	A combination of the best features of Datastore and the Firebase Realtime Database	Datastore is a NoSQL database that existed in GC prior to the acquisition of Firebase
Data model	Document database organized into documents and collections	Entities organized into kinds and entity groups
API support	Firestore v1 API	Datastore v1 API
Real-time updates	Yes	No
Offline data persistence	The mobile and web client libraries support offline data persistence.	No
Performance	Automatically scales to millions of concurrent clients	Automatically scales to millions of writes per second
When to use	Mobile and web apps	NoSQL database storage for applications running in Google cloud (server side)

Choosing between Native mode and Datastore mode

*Can change from one to the other but only if the database is empty

Native vs Datastore Firestore Modes

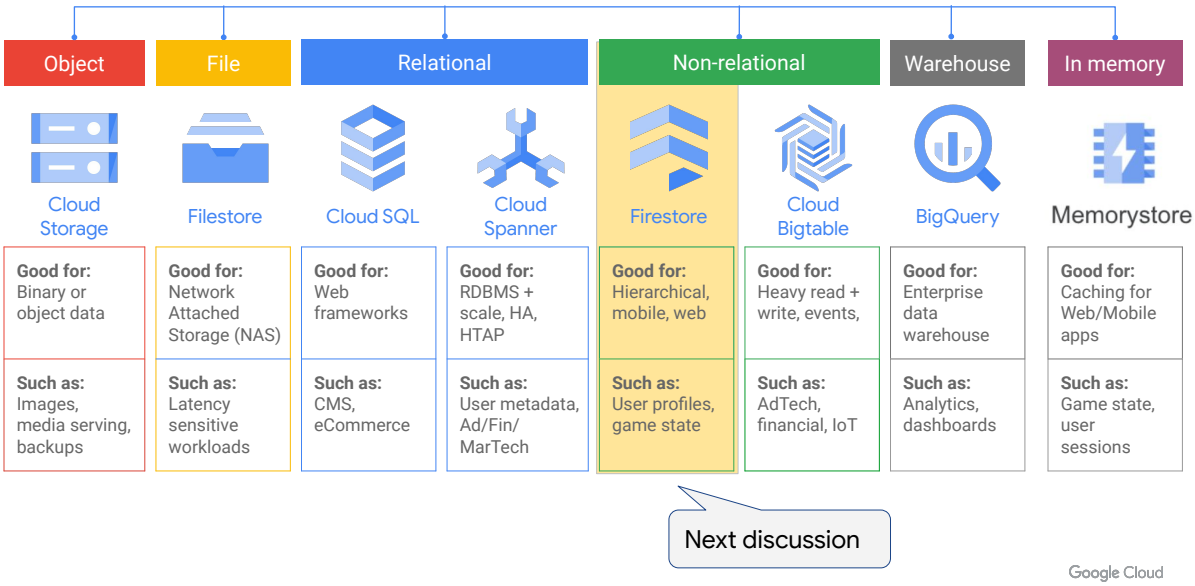
- Native mode
  - Native mobile and web client libraries
  - Real-time updates and offline features
  - Uses Firebase APIs
  - Document database similar to MongoDB
- Datastore mode
  - Optimized for serving as a backend for services running in Google Cloud
  - Does not support some Firestore features like offline support for mobile devices and synchronization
  - Compatible with Datastore APIs



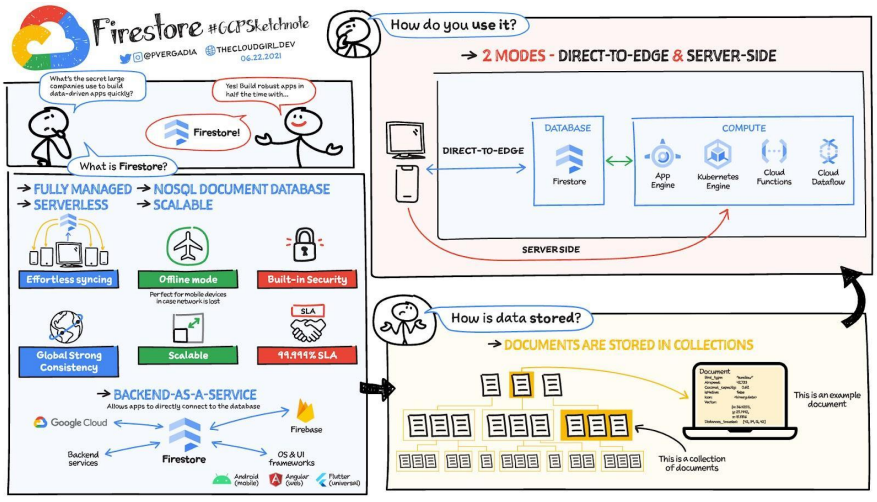
	Native mode Enable all of Cloud Firestore's features, with offline support and real-time synchronization. SELECT NATIVE MODE	Datastore mode Leverage Cloud Datastore's system behavior on top of Cloud Firestore's powerful storage layer. SELECT DATASTORE MODE
API	Firestore	Datastore
Scalability	Automatically scales to millions of concurrent clients	Automatically scales to millions of writes per second
App engine support	Not supported in the App Engine standard Python 2.7 and PHP 5.5 runtimes	All runtimes
Max writes per second	10,000	No limit
Real-time updates	✓	✗
Mobile/web client libraries with offline data persistence	✓	✗

Choosing between Native mode and Datastore mode
https://cloud.google.com/datastore/docs/firestore-or-datastore

Storage and database services



Cloud Firestore

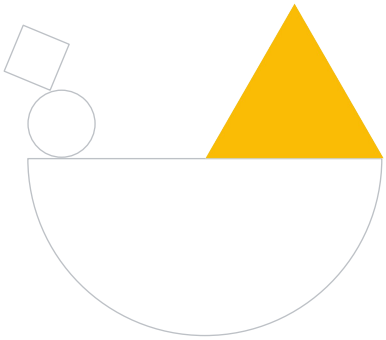


[All you need to know about Firestore]

All you need to know about Firestore: https://cloud.google.com/blog/topics/developers-practitioners/all-you-need-know-about-firestore-cheatsheet

Firebase Database Options

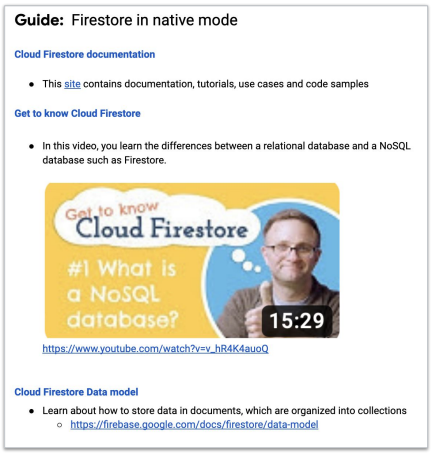
- Overview
- **Firestore in Native Mode**
- Firestore in Datastore mode



Firestore in native-mode

This guide discusses Firestore in native mode and provides:

- Overview video
- Documentation links to quickstarts, tutorials, etc.
- Data model overview
- How to query data

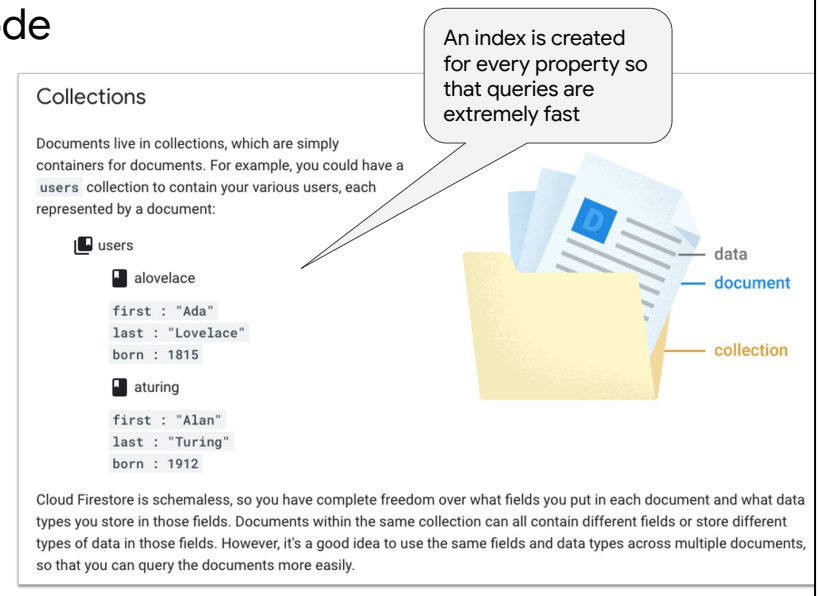


Cloud Firestore in native-mode

For iOS: https://firebase.google.com/docs/auth/ios/firebaseui
For Android: https://firebase.google.com/docs/auth/android/firebaseui
For web: https://firebase.google.com/docs/auth/web/firebaseui

Firestore in native mode

- Fully managed, serverless, scalable NoSQL database
- Built to support native mobile and web client libraries
- Provides real-time updates and offline features
- Data is stored as documents in Collections



From: https://firebase.google.com/docs/firestore/data-model#hierarchical-data

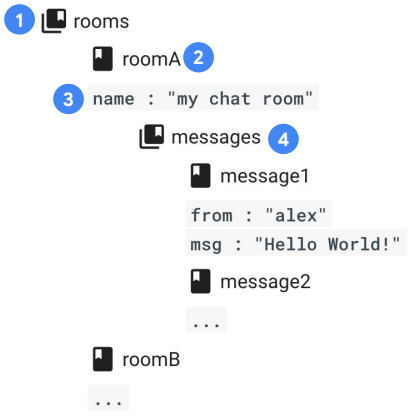
Python code example - adding users

```
from google.cloud import firestore

# The `project` parameter is optional and represents which project the client
# will act on behalf of. If not supplied, the client falls back to the default
# project inferred from the environment.

db = firestore.Client(project="bt-firestore-native-mode")

#Create a new collection and a document using the following example code.
doc_ref = db.collection("users").document("alovelace")
doc_ref.set({"first": "Ada", "last": "Lovelace", "born": 1815})

#Now add another document to the users collection.
#Notice that this document includes a key-value pair (middle name) that does not appear in the first document.
#Documents in a collection can contain different sets of information.
doc_ref = db.collection("users").document("aturing")
doc_ref.set({"first": "Alan", "middle": "Mathison", "last": "Turing", "born": 1912})
```

Python code example - retrieving the collection

```python
from google.cloud import firestore


# The `project` parameter is optional and represents which project the client
# will act on behalf of. If not supplied, the client falls back to the default
# project inferred from the environment.

db = firestore.Client(project="bt-firestore-native-mode")

#retrieve the collection
#Must setup ADC

users_ref = db.collection("users")
docs = users_ref.stream()

for doc in docs:
    print(f"{doc.id} => {doc.to_dict()}")
```

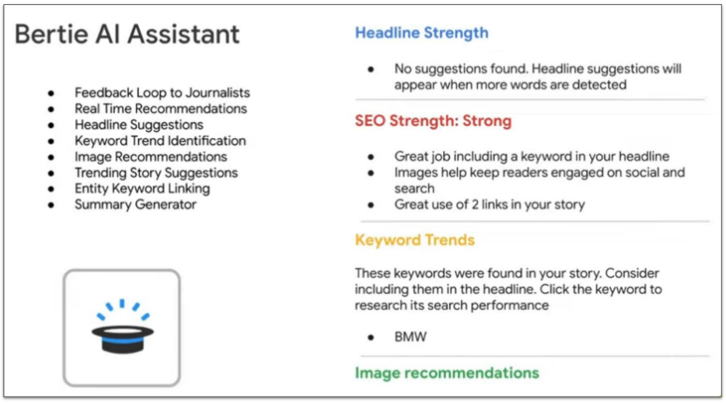
Modeling Firestore Documents



1. Data is organized in Collections
2. Collections contain Documents
3. Documents have properties
4. Documents can also have Subcollections

Firestore - customer use case

- Forbes created Bertie - an AI assistant for journalists
- Journalists upload their content and Bertie provides feedback
  - Strength of the article's headline
  - Keywords needed for search engine optimization
  - Words to add to the headline to increase search performance

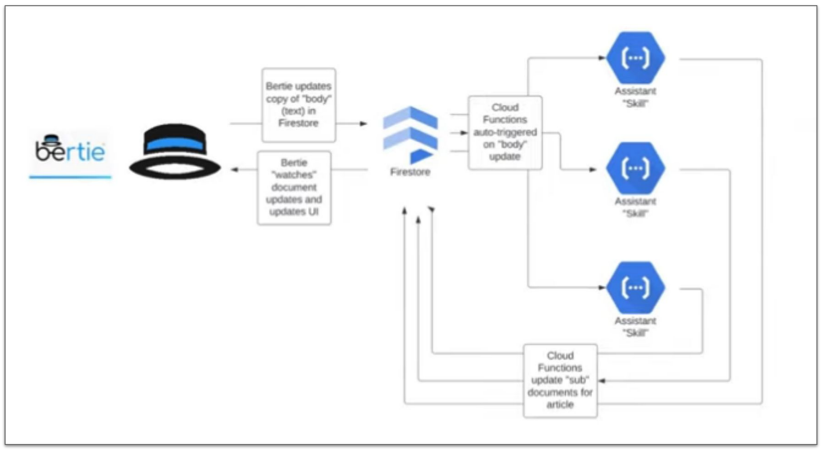


YouTube video: https://www.youtube.com/watch?v=KVRxsRPhmoo

Forbes website (for anyone that wants to know more about the company) https://www.forbes.com/

Forbes - Bertie AI Architecture

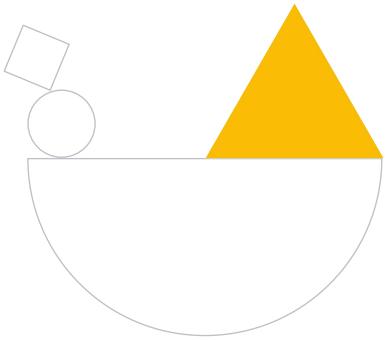
- Content is stored in Firestore
- Data updates trigger Cloud Functions
  - Each Cloud Function performs a different task
  - Results are written back to Firestore
  - Website is refreshed with the recommendations



YouTube video: https://www.youtube.com/watch?v=KVRxsRPhmoo

Firebase Database Options

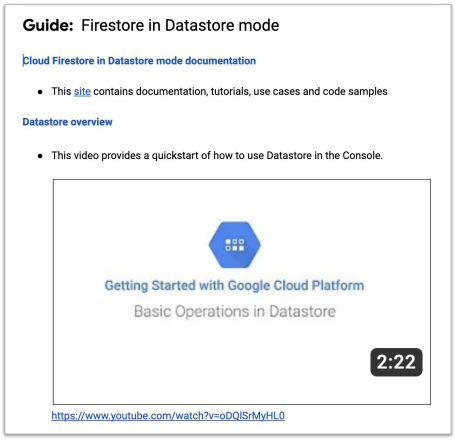
- Overview
- Firestore in Native Mode
- **Firestore in Datastore mode**



Firestore in Datastore mode

This guide discusses Firestore in Datastore mode and provides:

- Overview video
- Documentation links to quickstarts, tutorials, etc.
- Data model overview
- How to query data
- How to use the emulator
- And more



Firestore in Datastore mode

Firestore in Datastore mode

- Cloud Datastore was the original Google Cloud NoSQL database service designed to hold server-side data from applications running in Google Cloud
- Cloud Firestore was introduced in 2017
  - Scalable NoSQL cloud database designed to store and sync data for mobile and web
- To simplify its product lineup, Google merged the two products.
  - Cloud Datastore is now part of the Cloud Firestore product and is referred to as "Firestore in Datastore mode

Firestore in Datastore mode uses Datastore system behavior but stores and reads data from Firestore's improved storage layer.

Firestore in Datastore mode is backward compatible with Datastore, but Firestore's native data model, real-time updates, and mobile and web client library features cannot be used with Datastore mode.

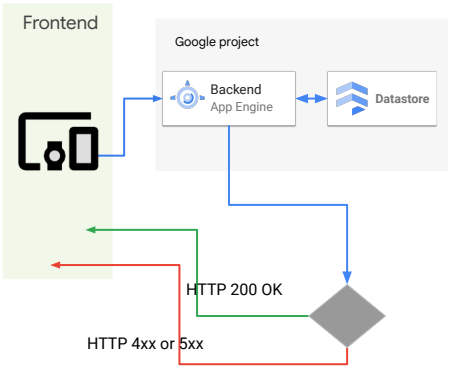
To access all of the new Firestore features, you must use Firestore in Native mode. When you create a new Firestore database, you must select a database mode, and you cannot use both Native mode and Datastore mode in the same project.

Datastore mode should typically be used for server applications. Native mode should be used for new mobile and web apps, or when requiring real-time and offline features.

Firestore in Datastore mode today

- Fully backward compatible with original Datastore but uses Firestore's improved storage layer
- As of 2023, a single project can contain both a Firestore Native mode database and a Datastore mode database
- How to decide:
  - Choose Datastore mode when using it with apps running in Google Cloud
    - Automatically scales to millions of writes per second.
  - Choose Native mode for new mobile and web apps or when requiring real-time and offline features.
    - Automatically scales to millions of concurrent clients.
    - Offers real-time synchronization

Handling errors



Error	Recommended Action
• ALREADY_EXISTS • FAILED_PRECONDITION • INVALID_ARGUMENT • NOT_FOUND • PERMISSION_DENIED • UNAUTHENTICATED	Do not retry without fixing the problem.
• RESOURCE_EXHAUSTED	Fix quota if exceeded. If quota was not exceeded, retry using exponential backoff.
• DEADLINE_EXCEEDED • UNAVAILABLE	Retry using exponential backoff.
• INTERNAL	Do not retry this request more than once.
• ABORTED	For a non-transactional commit: • Retry the request or structure your entities to reduce contention. For requests that are part of a transactional commit: • Retry the entire transaction or structure your entities to reduce contention.

Errors and Error Handling

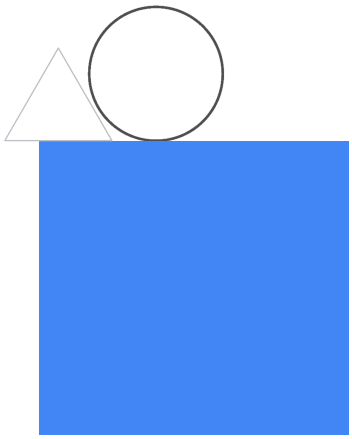
When a Datastore request succeeds, the API will return a 200 OK status code, with the requested data in the body of the response.

Failures return a 4xx or 5xx status code with more specific information about the errors that caused the failure.

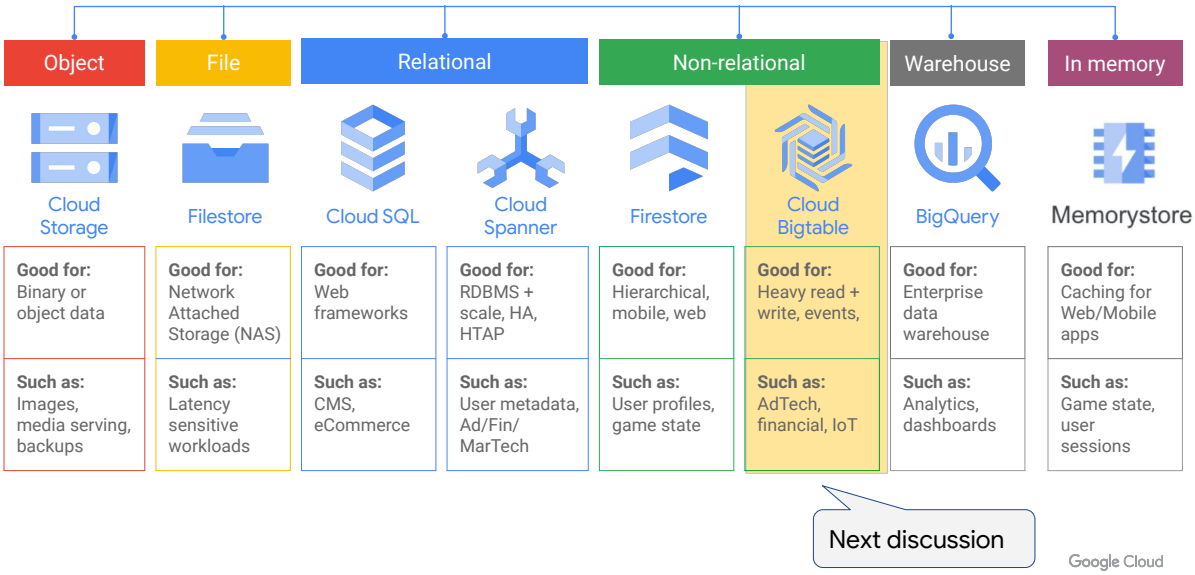
Errors should be classified by inspecting the value of the canonical error code. The table displays recommended actions per error code.

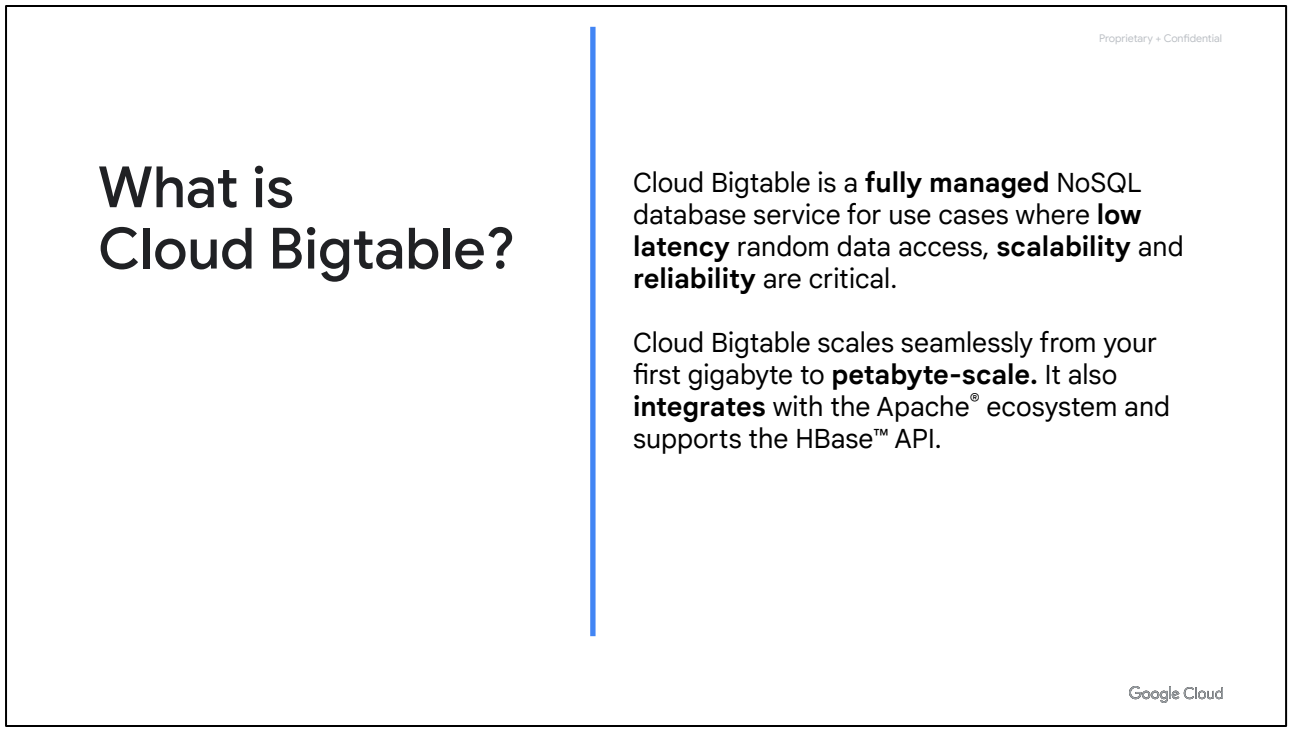
For more information, see: https://cloud.google.com/datastore/docs/concepts/errors

Cloud Bigtable



Storage and database services





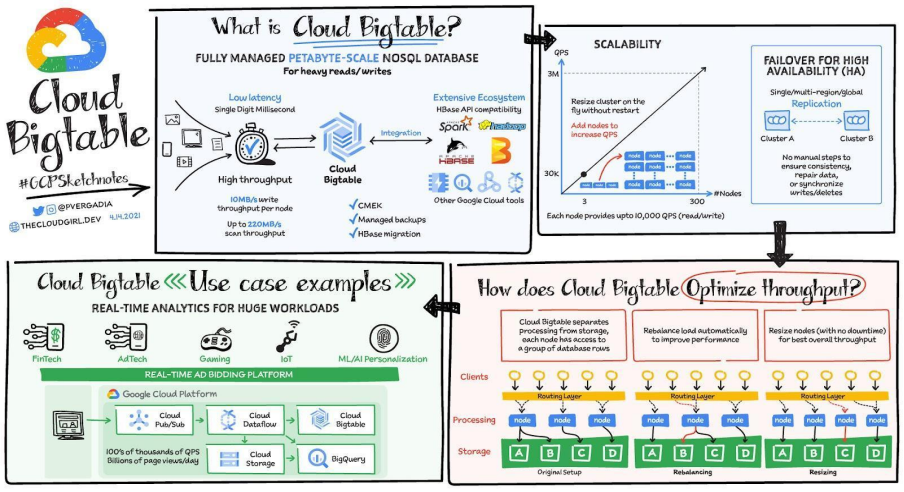
Cloud Bigtable is based on the original Bigtable technology that supports several billion-user Google products (search, gmail etc.)

Scalability and high throughput are key - your application will not exhaust the capacity of Cloud Bigtable, we allow you to easily scale to meet your demands, and Cloud Bigtable is today supporting customers with millions of queries per second (like Cognite)

As a managed service, it frees your team from database and infrastructure management tasks, allowing them to dedicate their time to where your business adds value

Completely compatible with an existing Apache HBase setup, enabling seamless migration from an existing HBase setup, and compatibility with many other Apache services such as Spark, Storm etc.

Cloud Bigtable



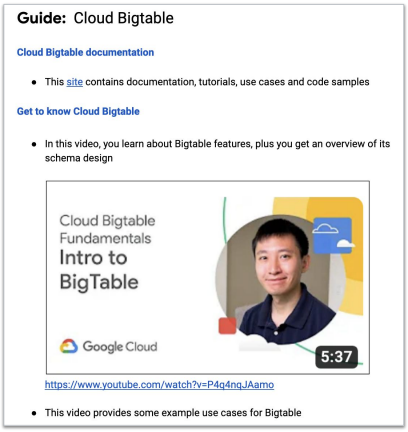
https://cloud.google.com/blog/topics/developers-practitioners/how-big-cloud-bigtable

How BIG is Cloud Bigtable:
https://cloud.google.com/blog/topics/developers-practitioners/how-big-cloud-bigtable

Guide: Cloud BigTable

Topics covered include

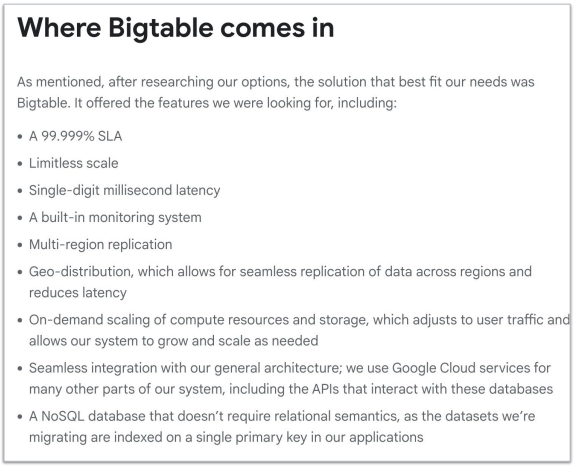
- Documentation with links to use cases, tutorials, and more
- Schema design best practices
- Bigtable emulator
- Example use cases
- Links to code samples
- Recommended hands-on lab



Cloud Bigtable

Bigtable scales to billions of rows and thousands of columns

- Stores terabytes to petabytes of data
- Built for fast key-value lookup and scanning over a defined key range
  - Similar to a spreadsheet that gives you access to any set of columns from contiguous rows by searching only the value in the first column (the key)
- Updates to individual rows are atomic
- Ideal for storing large amounts of single-keyed data and MapReduce operations



Where Bigtable comes in

As mentioned, after researching our options, the solution that best fit our needs was Bigtable. It offered the features we were looking for, including:

- A 99.999% SLA
- Limitless scale
- Single-digit millisecond latency
- A built-in monitoring system
- Multi-region replication
- Geo-distribution, which allows for seamless replication of data across regions and reduces latency
- On-demand scaling of compute resources and storage, which adjusts to user traffic and allows our system to grow and scale as needed
- Seamless integration with our general architecture; we use Google Cloud services for many other parts of our system, including the APIs that interact with these databases
- A NoSQL database that doesn't require relational semantics, as the datasets we're migrating are indexed on a single primary key in our applications

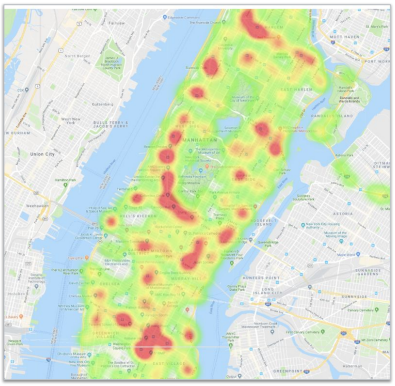
From MySQL to NoSQL: Bitly's big move to Bigtable

Cloud Bigtable example use cases

Healthcare	Data streaming in from connected healthcare devices, e.g., pacemakers
Financial Services	Storage of large amounts of transaction, risk management, and compliance data
Retail	Streaming data used to generate product recommendations given items in a shopping cart
Gaming	Real-time player data for game analytics
IoT	Streaming data from connected devices and sensors, e.g., electric car battery status
Logs and Metrics	Storage of log data and metrics for analysis
Time series data	Storage of resource consumption like CPU and memory usage over time for multiple servers

Cloud Bigtable codelab

- Introduction to Cloud Bigtable codelab imports New York city bus data
  - Generates a heat map of where buses are at a given moment in time



Example of the raw data

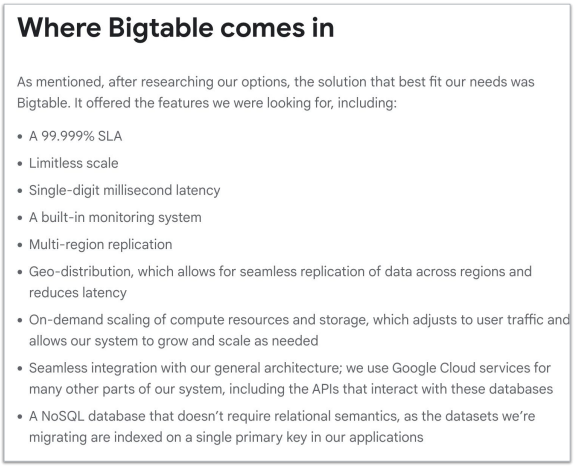
	RecordedAtTime	Dire	PublishedLin	OriginName	OriginLat	OriginLong	DestinationN	DestinationL	DestinationL	VehicleRef	VehicleLocat	VehicleLocat
2	12/1/17 0:05	0	B67	MC DONALD	40.63816	-73.978939	DNTWN BKL	40.700253	-73.98703	NYCT_406	40.671578	-73.977672
3	12/1/17 0:06	1	Bx7	RIVERDALE /	40.912363	-73.902699	WASHINGTC	40.839813	-73.939745	NYCT_4223	40.866243	-73.925258
4	12/1/17 0:05	0	S51	LINCOLN AV,	40.581245	-74.11199	ST GEORGE	40.643585	-74.07261	NYCT_7080	40.581364	-74.112033
5	12/1/17 0:05	0	M8	WEST ST/CH	40.732847	-74.010081	AVENUE "D"	40.724689	-73.974548	NYCT_3809	40.734194	-73.999677
6	12/1/17 0:06	0	M101	ASTOR PL/3	40.729567	-73.990052	FT GEORGE	40.855666	-73.925259	NYCT_5902	40.773784	-73.957659
7	12/1/17 0:06	0	M5	6 AV/W 31 S	40.748042	-73.988957	WASHINGTC	40.848264	-73.937455	NYCT_6365	40.76087	-73.979718
8	12/1/17 0:06	0	B41	E 70 ST/VETI	40.619935	-73.908708	EMPIRE BL	40.663012	-73.962211	NYCT_5045	40.619318	-73.920946
9	12/1/17 0:05	1	S53	4 AV/86 ST	40.622312	-74.028688	PT RICHMON	40.640296	-74.13133	NYCT_8249	40.608292	-74.088821
10	12/1/17 0:06	0	M101	ASTOR PL/3	40.729567	-73.990052	96 ST	40.785724	-73.948791	NYCT_5268	40.730379	-73.989303
11	12/1/17 0:05	1	B6	LIVONIA AV/	40.666382	-73.883617	BENSONHUF	40.592947	-73.993383	NYCT_5090	40.653041	-73.889844

Image from codelab: Introduction to Cloud Bigtable

The dataset is from the NYC MTA buses data stream service. In roughly 10 minute increments the bus location, route, bus stop and more is included in each row. The scheduled arrival time from the bus schedule is also included, to give an indication of where the bus should be (how much behind schedule, or on time, or even ahead of schedule).

Bigtable - customer use case: Bitly

- Bitly, the link & QR Code management platform, migrated 80 billion rows of link data to Cloud Bigtable.
- Data was moved from a MySQL database to Bigtable in just six days



Where Bigtable comes in

As mentioned, after researching our options, the solution that best fit our needs was Bigtable. It offered the features we were looking for, including:

- A 99.999% SLA
- Limitless scale
- Single-digit millisecond latency
- A built-in monitoring system
- Multi-region replication
- Geo-distribution, which allows for seamless replication of data across regions and reduces latency
- On-demand scaling of compute resources and storage, which adjusts to user traffic and allows our system to grow and scale as needed
- Seamless integration with our general architecture; we use Google Cloud services for many other parts of our system, including the APIs that interact with these databases
- A NoSQL database that doesn't require relational semantics, as the datasets we're migrating are indexed on a single primary key in our applications

From MySQL to NoSQL: Bitly's big move to Bigtable

About Bitly
https://www.macysinc.com/about

Bitly
https://cloud.google.com/blog/products/databases/bitly-migrates-link-data-from-mysql-to-bigtable-for-scalability/

Strong vs Eventual consistency in distributed storage systems

- Strong consistency
  - Once a write is acknowledged, all subsequent reads reflect that write
  - Use case: Systems where data accuracy and integrity are paramount, such as financial systems or databases enforcing ACID properties
- Eventual consistency
  - After a write operation, reads might return the old value for some indefinite time
  - Given enough time without new writes, all reads will eventually return the last written value

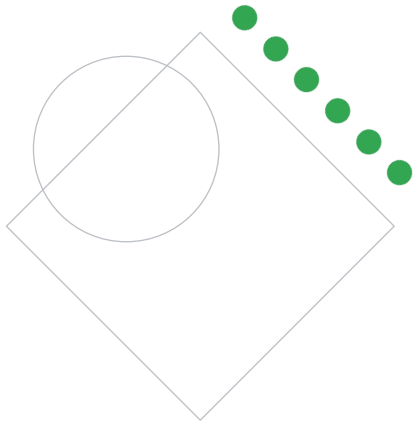
Strong vs Eventual consistency in Google Cloud databases

Strong Consistency	Eventual Consistency
Cloud SQL	Cloud SQL read replicas (“within seconds”)
Cloud Spanner	Bigtable read replicas
Firestore in native mode	Firestore in Datastore mode*
Bigtable (single row updates)	

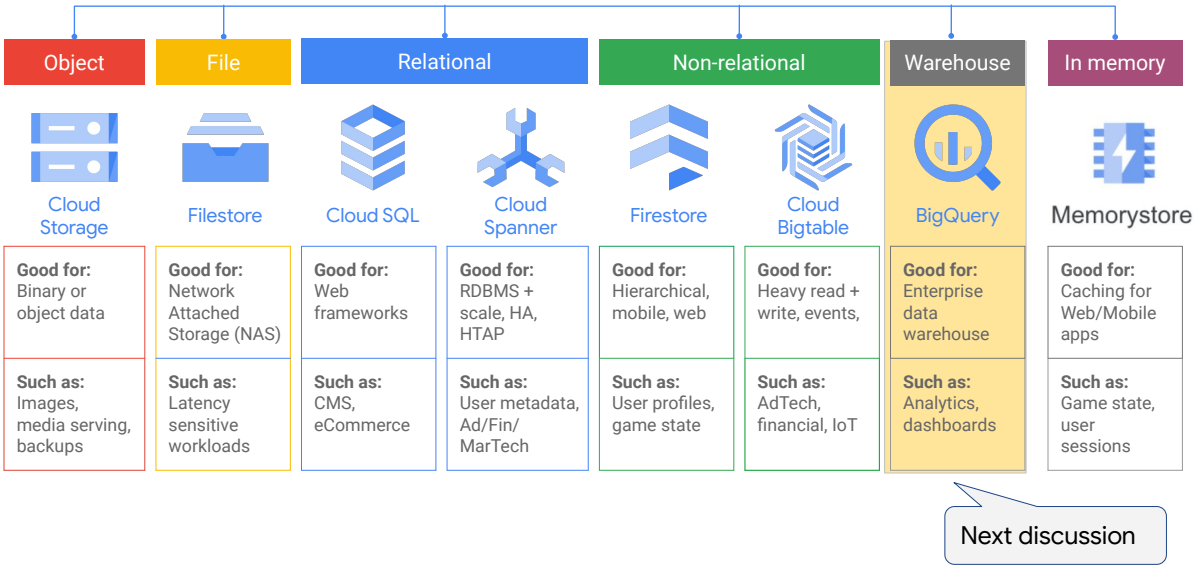
*Some queries provide strong consistency; Others provide eventual consistency. See docs

Firestore - Eventual Consistency
https://cloud.google.com/datastore/docs/articles/balancing-strong-and-eventual-consistency-with-google-cloud-datastore#eventual-consistency-in-datastore

BigQuery

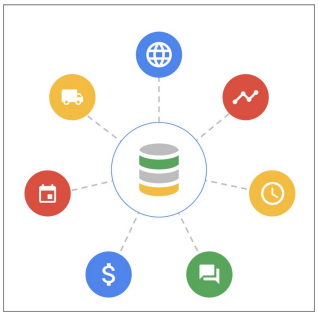


Storage and database services



What is a Data Warehouse?

- Allows data from multiple sources to be combined and analyzed
  - Historical archive of data
- Data sources could be:
  - Relational databases
  - Logs
  - Web data
  - Etc.
- Optimized for analytical processing
  - Can handle large amounts of data and complex queries, and is well-suited for reporting and data analysis
- **BigQuery is Google Cloud's data warehouse (OLAP*)**



A data warehouse is a central hub for business data. Different types of data can be transformed and consolidated into the warehouse for analysis

*OLAP: Online Analytical Processing

Online Transactional Processing (OLTP) vs Data warehouse

Aspect	Online Transaction Processing (OLTP) Spanner & Cloud SQL	Data Warehouse BigQuery
Purpose	Manage transactional data in real-time	Support analytical queries and reporting based on historical data BigQuery can also process streaming data in near real-time
Data Volume	Handles relatively small amounts of data (with exceptions, e.g., Spanner)	Stores and manages larger volumes of data, often in the terabytes or even petabytes
Data Latency	Data is available immediately after it is entered into the system	May have some latency, since data is extracted from OLTP systems on a regular basis and transformed before being loaded into the data warehouse
Usage	Used by operational staff to support day-to-day business processes	Used by business analysts and decision-makers to perform complex analysis and reporting, such as forecasting, trend analysis, and predictive modeling
Example Use cases	Point of Sale systems, inventory management, airline reservation systems, hospital medical records	Identify patterns and trends, and develop predictive models to anticipate future events; Develop targeted marketing campaigns to improve customer engagement; Analyze healthcare data to improve patient outcomes and optimize healthcare operations

BigQuery

- Fully managed, serverless, highly scalable data warehouse that can run analytics over vast amounts of data in near real time.
- Processes multi-terabytes of data in minutes
- Automatic high availability
- Supports federated queries
  - Cloud SQL & Cloud Spanner
  - Cloud Bigtable
  - Files in Cloud Storage
- Use cases:
  - Near real-time analytics of streaming data to predict business outcomes with built-in machine learning, geospatial analysis and more
  - Analysis of historical data

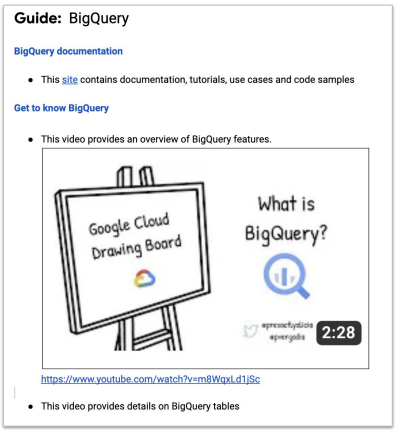
BigQuery

Cloud data warehouse to power your data-driven innovation
https://cloud.google.com/bigquery

Guide: BigQuery

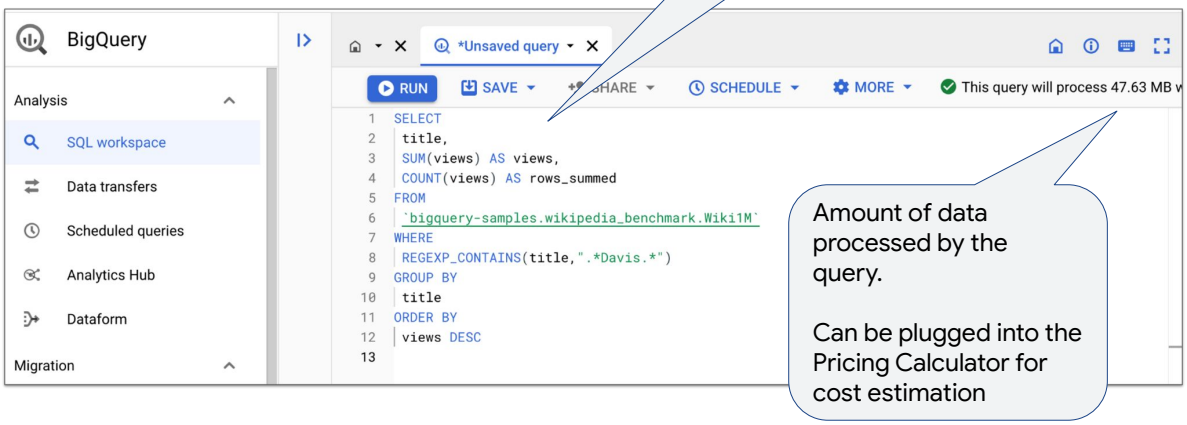
Topics covered include

- Documentation with links to use cases, tutorials, and more
- How to create datasets, tables and views
- Code samples
- Recommended hands-on lab

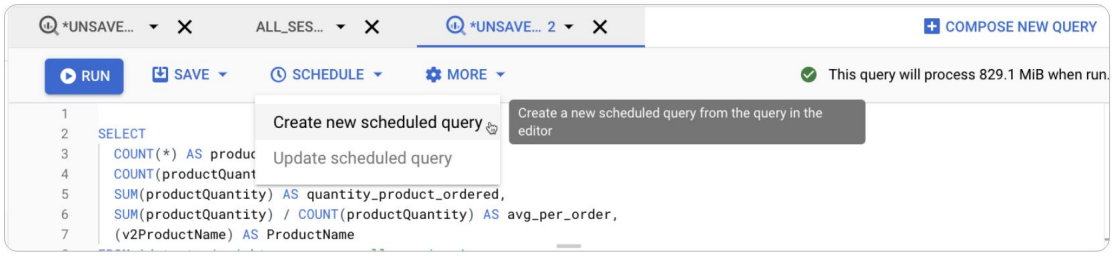


Big Query

BigQuery User Interface



Can also scheduled queries to run periodically

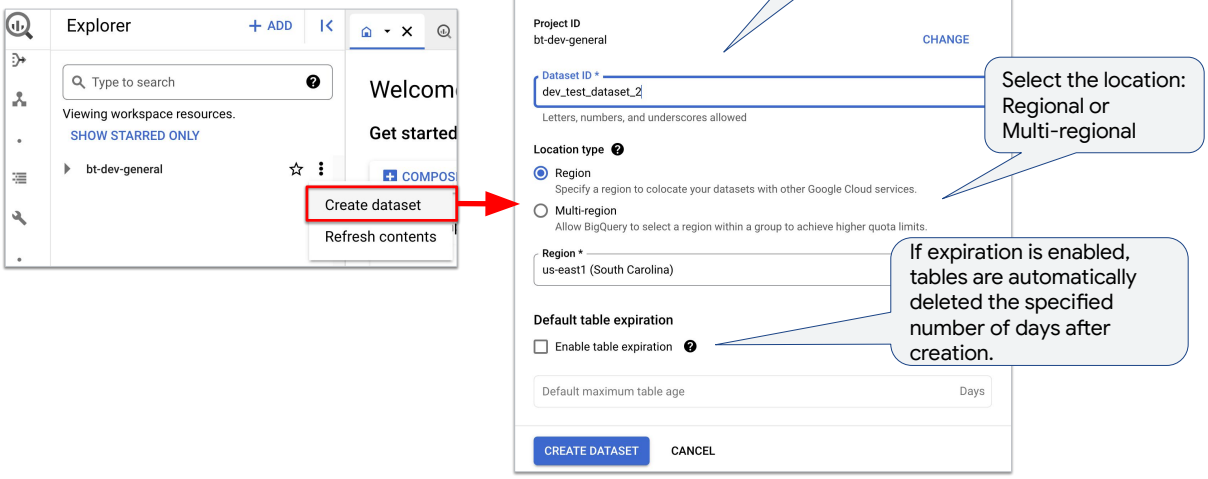


https://cloud.google.com/bigquery/docs/scheduling-queries

BigQuery Datasets

- Datasets are top-level containers that are used to organize and control access to tables and views
  - Must create at least one dataset before loading data into BigQuery
- Within a dataset, can be
  - Tables
    - Similar to tables in other database systems
      - Individual records are organized into columns, not rows
  - Views
    - A virtual table defined by a SQL query
      - Query them the same as standard tables
        - Views don't store the data directly
          - Instead, they reference data stored in other tables.

Creating a dataset in the Console



Creating a dataset with code

```
from google.cloud import bigquery

# Construct a BigQuery client object.
client = bigquery.Client()

# Supply the name of the dataset
dataset_id = "{}.dev_test_dataset_1".format(client.project)


# Construct a full Dataset object to send to the API.
dataset = bigquery.Dataset(dataset_id)

#Specify the geographic location where the dataset should reside.
dataset.location = "us-east1"

# Send the dataset to the API for creation, with an explicit timeout.
# Raises google.api_core.exceptions.Conflict if the Dataset already
# exists within the project.
dataset = client.create_dataset(dataset, timeout=30)  # Make an API request.
print("Created dataset {}.{}".format(client.project, dataset.dataset_id))
```

Using the BigQuery client

Name of the dataset will be in the format of `projectID.datasetname`

Creating a regional dataset

Making the API request

Creating a table with code

```
from google.cloud import bigquery

# Construct a BigQuery client object.
client = bigquery.Client()

# Set table_id to the ID of the destination table.
dataset_id = "{}.dev_test_dataset_1".format(client.project)
table_id = "{}.dev_test_dataset_1.shakespeare".format(client.project)

job_config = bigquery.QueryJobConfig(destination=table_id)

sql = """
    SELECT word, word_count, corpus
    FROM `bigquery-public-data.samples.shakespeare;`
"""

# Start the query, passing in the extra configuration.
query_job = client.query(sql, job_config=job_config)  # Make an API request.
query_job.result()  # Wait for the job to complete.

print("Query results loaded to the table {}".format(table_id))
```

Name of the dataset will be in the format of `projectID.datasetname`

Name of table

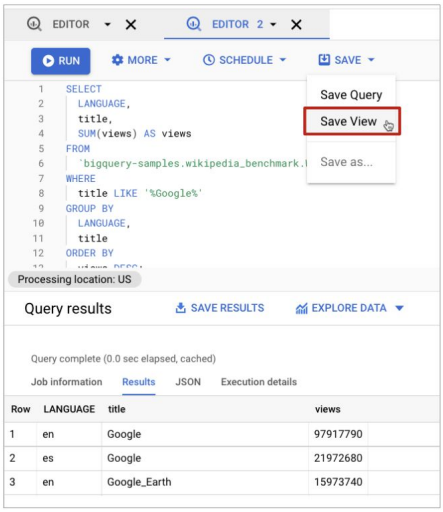
Populating the table via a query of existing data

Storing results in a view

- View = Saved SQL query (a virtual table)
- The underlying query is executed each time the view is accessed

Can also create views via SQL

```
CREATE VIEW ecommerce.shirts_vw AS
SELECT DISTINCT
  v2ProductName
FROM
`data-to-insights.ecommerce.all_sessions`
WHERE LOWER(v2ProductName) LIKE '%shirt%'
LIMIT 100
```



A view is a virtual table defined by a SQL query. You can query views in BigQuery using the web UI, the command-line tool, or the API. You can also use a view as a data source for a visualization tool such as Looker Studio.

BigQuery's views are logical views, not materialized views. Because views are not materialized, the query that defines the view is run each time the view is queried. Queries are billed according to the total amount of data in all table fields referenced directly or indirectly by the top-level query. For more information, see query pricing.

SQL queries used to define views are subject to the standard query quotas.

Limitations

BigQuery views are subject to the following limitations:

- You cannot run a BigQuery job that exports data from a view.
- You cannot use the TableDataList JSON API method to retrieve data from a view. For more information, see Tabledata: list.
- You cannot mix standard SQL and legacy SQL queries when using views. A standard SQL query cannot reference a view defined using legacy SQL syntax.
- The schemas of the underlying tables are stored with the view when the view is created. If columns are added, deleted, and so on after the view is created,

- the reported schema will be inaccurate until the view is updated. Even though the reported schema may be inaccurate, all submitted queries produce accurate results.
- You cannot update a legacy SQL view to standard SQL in the BigQuery web UI. You can change the SQL language using the command-line tool bq update --view command or by using the update or patch API methods.
- You cannot include a user-defined function in the SQL query that defines a view.
- You cannot reference a view in a wildcard table query.
- BigQuery supports up to four levels of nested views in legacy SQL. If there are more than four levels, an INVALID_INPUT error returns. In standard SQL, you are limited to 100 levels of nested views.
- You are limited to 1,000 authorized views per dataset.

BigQuery also supports materialized views

- Views that periodically cache the results of a query for increased performance and efficiency
  - Incremental data changes from the base tables are automatically added to the materialized views, with no user action required
  - If changes to base tables might invalidate the materialized view, then data is read directly from the base tables
  - If any part of a query against the base table can be resolved by querying the materialized view, then BigQuery reroutes the query to use the materialized view for better performance and efficiency.

```
CREATE MATERIALIZED VIEW
myproject.mydataset.my_mv_table AS (
  SELECT
    product_id,
    SUM(clicks) AS sum_clicks
  FROM
    myproject.mydataset.my_base_table
  GROUP BY
    product_id
);
```


In BigQuery, materialized views are precomputed views that periodically cache the results of a query for increased performance and efficiency. BigQuery leverages precomputed results from materialized views and whenever possible reads only delta changes from the base table to compute up-to-date results. Materialized views can be queried directly or can be used by the BigQuery optimizer to process queries to the base tables.

Queries that use materialized views are generally faster and consume fewer resources than queries that retrieve the same data only from the base table. Materialized views can significantly improve the performance of workloads that have the characteristic of common and repeated queries.

Further information on materialized views:
https://cloud.google.com/bigquery/docs/materialized-views-intro

Jobs are actions that BigQuery runs on your behalf

- Load Jobs:
  - Used to load data from files into BigQuery tables and supports various data formats like CSV, JSON, Avro, Parquet, and ORC.
  - Data can be loaded from Google Cloud Storage or from a local file.
- Query Jobs:
  - Used to execute SQL queries.
  - The results can be saved to a new table, appended to an existing table, or overwritten in an existing table.
- Export Jobs:
  - Used to export data from BigQuery tables into files in Google Cloud Storage.
  - Supports formats like CSV, JSON, and Avro.
- Copy Jobs:
  - Used to copy data from one table to another, either within the same dataset or across datasets.
  - Can be used to create backups or to replicate data.
- Extract Jobs:
  - Similar to export jobs, they are used to extract data from a table to Google Cloud Storage.
- BigQuery also provides a Jobs API, which you can use to programmatically manage and track your jobs. This is especially useful when you're running automated data pipelines or workflows.

Using BigQuery from the command line

- Examine the shakespeare table in the samples dataset:

```
bq show bigquery-public-data:samples.shakespeare >out.txt
```

- Show the first 5 rows to see what the data looks like
  - Can also use the “Preview” feature in the console to see a preview of the data

```
bq query --use_legacy_sql=false \
    'SELECT * FROM `bigquery-public-data.samples.shakespeare`
     LIMIT 5'
```

- Determine how many times the substring ‘raisin’ appears in Shakespeare's works

```
bq query --use_legacy_sql=false \
    'SELECT word, SUM(word_count) AS count
     FROM `bigquery-public-data.samples.shakespeare`
     WHERE word LIKE "%raisin%"
     GROUP BY word;'
```

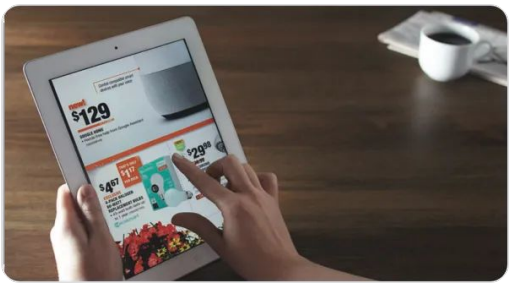
Using BigQuery from a client library

```
from google.cloud import bigquery

def query_shakespeare():
    client = bigquery.Client()
    query_job = client.query(
        """
        SELECT word, SUM(word_count) AS count
             FROM `bigquery-public-data.samples.shakespeare`
             WHERE word LIKE "%raisin%"
             GROUP BY word;"""
    )

    results = query_job.result()  # Waits for job to complete.

    for row in results:
        print("{} : {} views".format(row.word, row.count))


if __name__ == "__main__":
    query_shakespeare()
```

Same query that was run in the CLI

BigQuery - customer use case

- The Home Depot is the world's largest home improvement retailer
  - 2,300 stores in North America + online retail
  - Annual sales > $100 billion
- BigQuery provides timely data to help keep 50,000+ items stocked at over 2,000 locations, to ensure website availability, and provide relevant information through the call center
- No two Home Depots are alike, and the stock in each has to be managed at maximum efficiency.
  - Migrating to Google Cloud, THD's engineers built one of the industry's most efficient stock replenishment systems

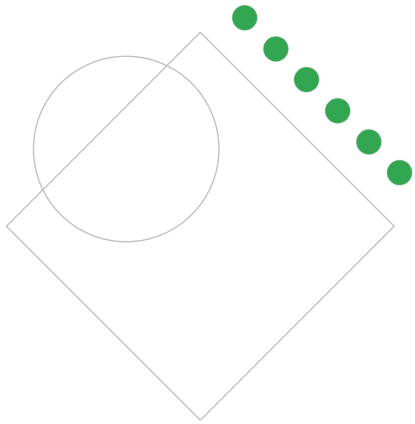


The Home Depot's data-driven focus on customer success

About The Home Depot
https://corporate.homedepot.com/page/about-us

The Home Depot's data-driven focus on customer success
https://cloud.google.com/customers/featured/the-home-depot

Storage summary



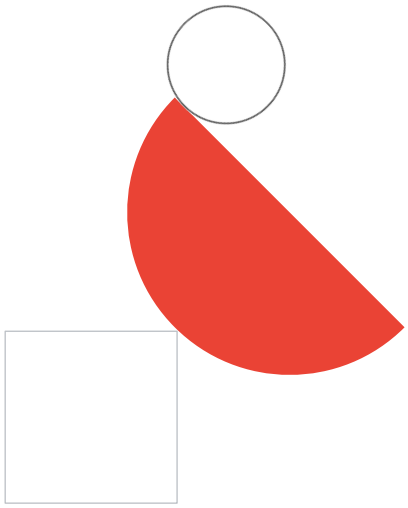
Storage at a glance

Product	Simple Description	Ideal for	Not Ideal for
Cloud Storage	Binary/object store	Large or rarely accessed unstructured data	Structured data, building fast apps
Firestore	Real-time NoSQL database to store and sync data	Mobile, web, multi-user, IoT & real-time applications	Analytic data, heavy writes
Datastore	Scalable store for structured data	App Engine and server apps, heavy read/write	Relational or analytic data
Bigtable	High-volume, low-latency database	"Flat," heavy read/write, or analytical data	High structure or transactional data
Cloud SQL	Well-understood VM-based RDBMS	Web frameworks, existing applications	Scaling, analytics, heavy writes
Spanner	Relational database service	Low-latency transactional systems	Analytic data
BigQuery	Auto-scaling analytic data warehouse	Interactive analysis of static datasets	Building fast apps

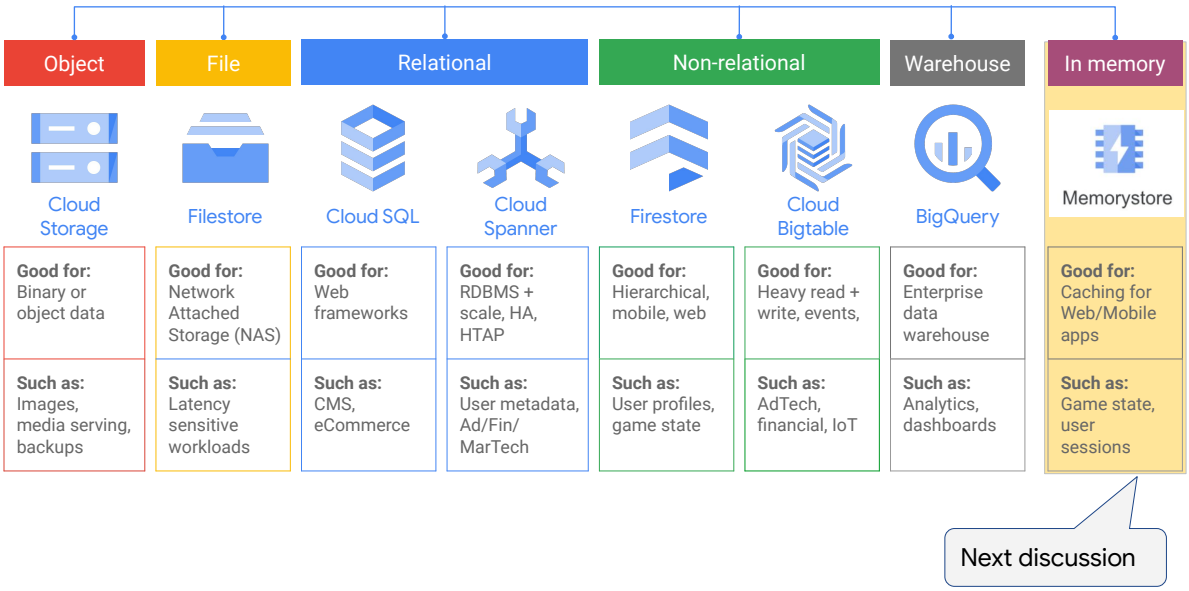
Here are the Google Cloud storage options at a glance. When you are choosing the right storage option for your application, it's important to understand what a product is and isn't ideal for by design.

This slide includes a simple description of the products, as well as use cases that are ideal for each product. Use cases that are not ideal for each product are also listed.

Cloud Memorystore

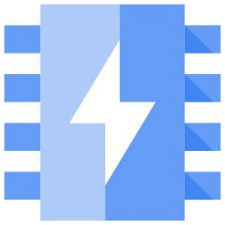


Storage and database services



Memorystore

- Fully managed implementation of the open source in-memory databases Redis and Memcached
- High availability, failover, patching and monitoring
- Sub-millisecond latency
- Instances up to 300 GB
- Network throughput of 12 Gbps



Memorystore

Memorystore
https://cloud.google.com/memorystore

Applications that you run on Google Cloud can achieve high levels of performance by leveraging either Redis or Memcached without the burden of managing complex deployments. Memorystore supports both of these highly scalable, available, and secure open source caching engines, and is fully protocol compatible with each engine.

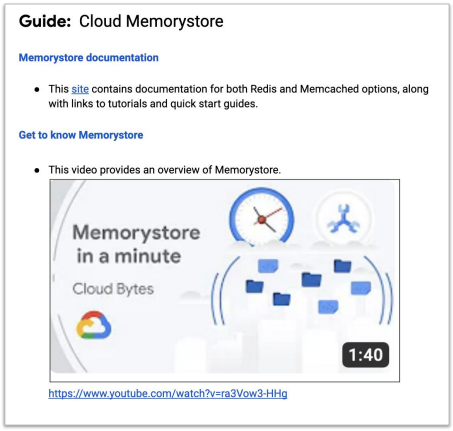
Memorystore is ideal for scalable web applications, gaming, and stream processing, where a distributed in-memory data store allows for the fast, real-time processing of data.

As a fully managed service, provisioning, replication, failover, and patching are all automated. You can also monitor instances and set up alerts with Cloud Monitoring. Memorystore can be protected from the internet through the use of VPC networks and private IP addresses. Memorystore also integrates with Cloud Identity and Access Management.

Guide: Cloud Memorystore

Topics covered include

- Documentation with links to use cases, tutorials, and more
- Videos providing an overview of Memorystore and key features
- Links to the client libraries
- Recommended hands-on lab

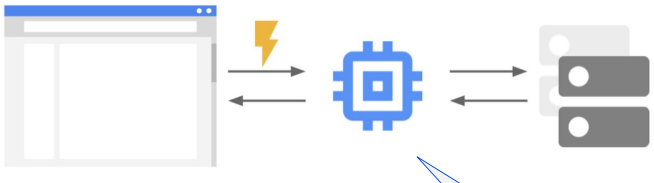


Cloud Memorystore

Example use cases

- Caching
  - Provides low latency access and high throughput for heavily accessed data
    - Compared to accessing the data from a disk based backend store
  - Common examples: Session management, frequently accessed queries
- Gaming
  - Everyone wants to see how they are progressing and where they stand on the leaderboard
    - Making this experience fast is critical
      - Memorystore makes it easy by storing a sorted list of scores
  - Player profiles is another example of data that may be accessed frequently
    - Memorystore makes it fast and easy to store and access profile data
- Stream Processing
  - Whether processing feeds or stream of data from IoT devices, Memorystore for Redis is a perfect fit for streaming solutions.
    - Combined with Pub/Sub and Dataflow, Memorystore provides a scalable, fast in-memory store for storing intermediate data that clients can access with very low latency

Use case: Storing user sessions



Application

Web browser

Cache

A caching layer is a high-speed storage layer that stores a subset of data.

Databases

Can store 100's of thousands of sessions

Processing is done in memory for very fast response times

Benefit

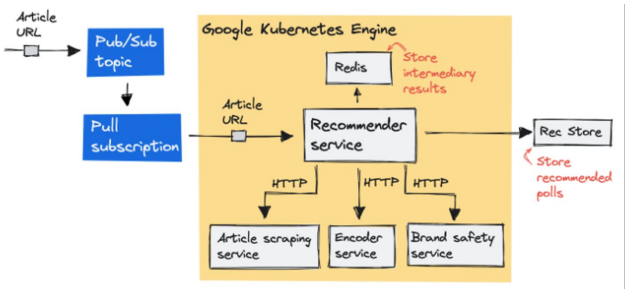
- Reduce latency
- Reduce back-end load

Main use case

- Gaming leaderboard
- Real-time application
- Social Media

Memorystore - customer use case

- Opinary creates polls that appear alongside news articles on various sites around the world
  - Machine learning is used to decide which poll to display by which article
- The polls let users share their opinion with one click and see how they compare to other readers.
- Publishers benefit by increased reader retention, and increased subscriptions
- Advertisers benefit from high-performing interaction with their audiences



Opinary generates recommendations faster on Cloud Run

Opinary generates recommendations faster on Cloud Run
https://cloud.google.com/blog/topics/developers-practitioners/opinary-generates-recommendations-faster-cloud-run/

More about Opinary (if interested)
https://opinary.com/


Google Cloud