



Building and Operationalizing Data Processing Systems

Tom Stern



The next section of the exam guide covers building data processing systems.

So that includes assembling data processing from parts, as well as using full services.



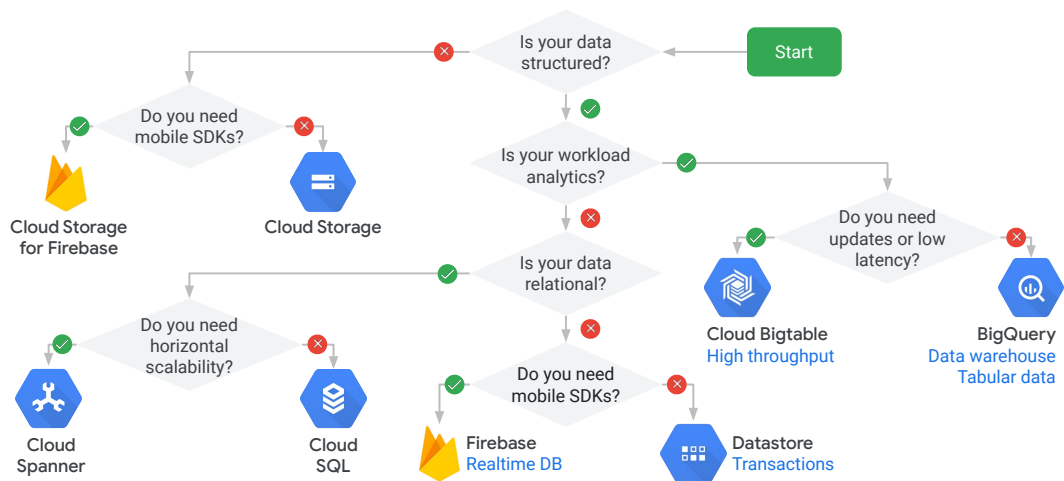
Building and Maintaining Data Structures and Databases



The first area of data processing we will look at is building and maintaining structures and databases.

So, not just selecting a particular kind of database or service, but also thinking about the qualities that are provided and starting to consider how to organize the data.

Selecting storage options



You can familiarize yourself with this diagram as well.

BigQuery is recommended as a data warehouse.

BigQuery is the default storage for tabular data.

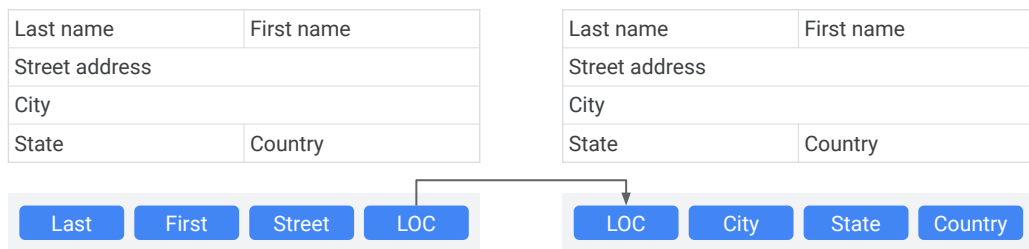
Use Cloud Bigtable if you need transactions.

Use Cloud Bigtable if you want low-latency/high-throughput.

Building and maintaining flexible data representations

Databases are systems for the storage and retrieval of information.

- One approach is to previously define a complex structure that is tailored to the intended operations. This approach may be faster and more efficient for queries and reports.
- Another approach is to retrieve information from documents geared toward the business logic. This approach may be easier for data entry and maintenance.



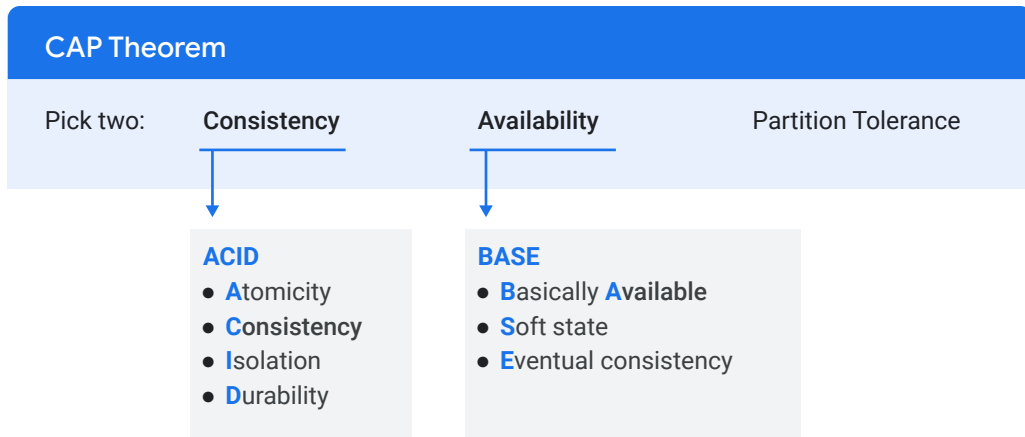
Here is some concrete advice on flexible data representation. You want the data divided up in a way that makes the most sense for your given use case. If the data is divided up too much, it creates additional work.

In the example on the left, each data item is stored separately, making it easy to filter on a specific fields and to perform updates.

In the example on the right, all of the data is stored in a single record, like a single string. Editing/updating is difficult. Filtering on a particular field would be hard.

In the example on the bottom, a relation is defined by two tables. This might make it easier to manage and report on the list of locations.

What transaction qualities are required?



ACID vs BASE is essential data knowledge that you will want to be very familiar with so that you can easily determine whether a particular data solution is compatible with the requirements identified in the case.

Example: for a financial transaction, a service that provides only eventual consistency might be incompatible.

Did you know that in some cases an eventually consistent solution can be made strongly consistent for a specific limited use case?

In Datastore, there are only two APIs that provide a strongly consistent view for reading entity values and indexes: (1) the lookup by key method and (2) the ancestor query.

Cloud Storage

Cluster node options

- Persistent storage
- Staging area for other services
- Storage classes

Access

- Granular access control: control access at Project, Bucket, or Object
- IAM roles, ACLs, and Signed URLs

Features

- Versioning
- Encryption options: default (Google), CMEK, CSEK
- Lifecycles
- Change storage class
- Streaming
- Data transfer/synchronization
- Storage Transfer Service
- JSON and XML APIs

Best Practices

- Traffic estimation

Storage classes are now: Standard Storage, Nearline Storage, Coldline Storage, and Archive Storage.



Cloud Storage is persistent. It has storage classes -- nearline, coldline, regional and multiregional.

There is granular access control. You should be familiar with all the methods of access control, including IAM roles and Signed URLs.

Cloud Storage has a ton of features that people often miss. They end up trying to duplicate the function in code, when in fact, all they need to do is use the capacity already available.

For example, you can change storage classes. You can stream data to Cloud Storage. Cloud Storage supports a kind of versioning. And there are multiple encryption options to meet different needs. Also, you can automate some of these features using Lifecycle management. For example, you could change the class of storage for an object or delete that object after a period of time.

Cloud SQL

Familiar: Cloud SQL supports most MySQL statements and functions, even Stored procedures, Triggers, and Views.

Not supported: User-defined functions, MySQL-esque replication, statements, and functions related to files and plugins.

Flexible pricing: You can pay per use or per hour. Backups, replication, and so forth are managed for you.

Connect from anywhere (can assign a static IP address, and use typical SQL connector libraries).

Fast: You can place your Cloud SQL instance in the same region as your App Engine or Compute Engine applications and get great bandwidth.

Google security: Cloud SQL resides in secure Google data centers. There are several ways to securely access a Cloud SQL instance (see *Cloud SQL Access* in the *Security* part of this course).



Cloud SQL is the managed service that provides a MySQL instance.

I want to highlight to you that there are several ways to securely connect to a Cloud SQL instance. And it would be important for you to be familiar with the different approaches and the benefits of each.

Cloud Bigtable

Properties:

- Cloud Bigtable is meant for high throughput data
- Millisecond latency, NoSQL
- Access is designed to optimize for a range of Row Key prefixes

Important features:

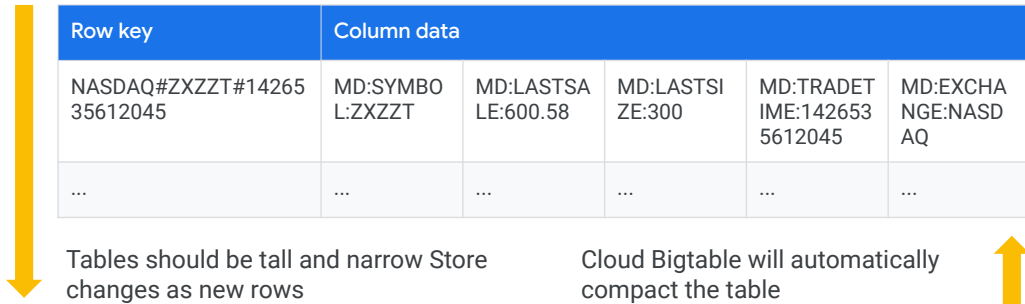
- Schema design and time-series support
- Access control
- Performance design
- Choosing between SSD and HDD



Cloud Bigtable is meant for high throughput data, it has millisecond latency, so it is much faster than BigQuery, for example. It is NoSQL. So this is good for a columnar store.

When would you want to select an SSD for the machines in the cluster rather than HDD? I'd guess if you needed faster performance.

Cloud Bigtable: High throughput data where access is primarily for a range of Row Key prefixes



Row key	Column data				
NASDAQ#ZXZZT#1426535612045	MD:SYMBOL:ZXZZT	MD:LASTSALE:600.58	MD:LASTSIZE:300	MD:TRADETIME:1426535612045	MD:EXCHANGE:NASDAQ
...

Tables should be tall and narrow Store changes as new rows

Cloud Bigtable will automatically compact the table

Tables in Bigtable are tall and narrow. This example shows stock trades.

Each trade is its own row. This will result in 100s of millions of rows per day. Which is fine with Cloud Bigtable.

Cloud Spanner

Properties:

- Global, fully managed, relational database with transactional consistency.
- Data in Cloud Spanner is strongly typed: you must define a schema for each database, and that schema must specify the data types of each column of each table.

Important features:

- Schema design, Data Model, and updates
- Secondary indexes
- Timestamp bounds and Commit timestamps
- Data types
- Transactions

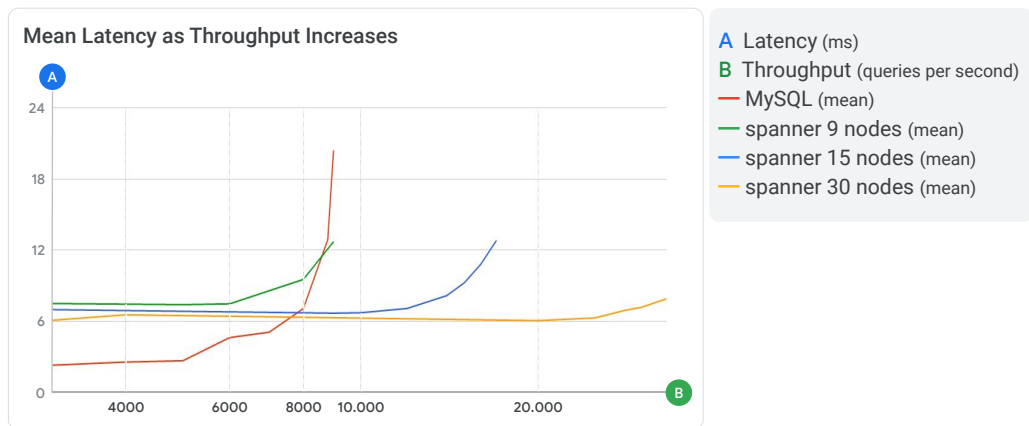


Cloud Spanner is strongly typed and globally consistent.

The two characteristics that distinguish it from Cloud SQL are globally consistent transactions and size.

Cloud Spanner can work with much larger databases than Cloud SQL.

Use Cloud Spanner if you need globally consistent data or more than one Cloud SQL instance



Cloud SQL is fine if you can get by with a single database. But if your needs are such that you need multiple databases, Cloud Spanner is a great choice.

MySQL hits a performance wall. If you look at the 99th percentile of latency, it is clear that performance degrades. Distributing MySQL is hard. However, Spanner distributes easily (even globally) and provides consistent performance. To support more throughput, just add more nodes.

Datastore

Properties:

- Datastore is a NoSQL object database
- **Atomic transactions**
- ACID support
- **High availability of reads and writes**
- **Massive scalability with high performance**
- **Flexible storage and querying of data**
- **Balance of strong and eventual consistency**
- **Encryption at rest**
- **Fully managed with no planned downtime**

Important features:

- Identity and access management
- Storage size calculations
- Multitenancy
- Encryption



Datastore is a NoSQL solution that used to be private to App Engine. It offers many features that are mainly useful to applications, such as persisting state information. It is now available to clients besides App Engine.

Comparing storage options: Use cases

	Datastore	Cloud Bigtable	Cloud Storage	Cloud SQL	Cloud Spanner	BigQuery
Type	NoSQL document	NoSQL wide column	Blobstore	Relational SQL for OLTP	Relational SQL for OLTP	Relational SQL for OLAP
Best for	Getting started, App Engine applications	"Flat" data, Heavy read/write, events, analytical data	Structured and unstructured binary or object data	Web frameworks, existing applications	Large-scale database applications (> ~2 TB)	Interactive querying, offline analytics
Use cases	Getting started, App Engine applications	AdTech, Financial and IoT data	Images, large media files, backups	User credentials, customer orders	Whenever high I/O, global consistency is needed	Data warehousing

Commit this table to memory, and be able to use it backwards.

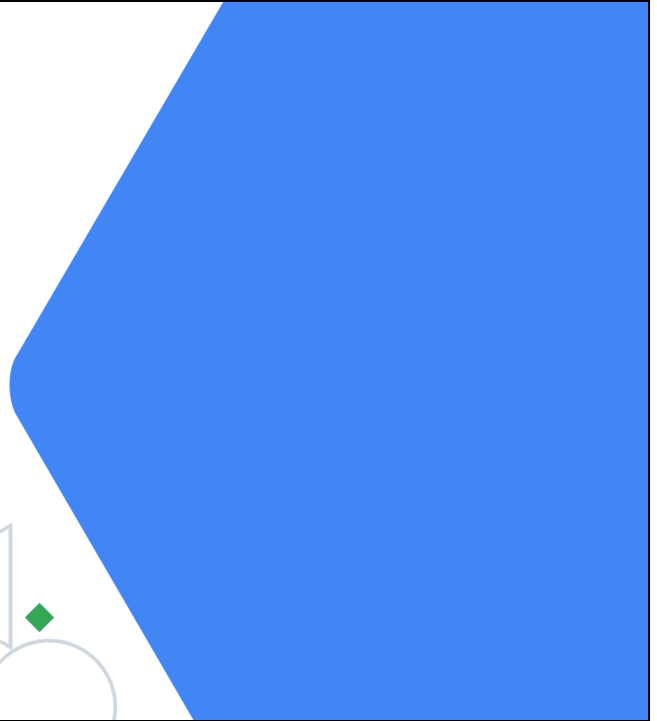
Example: If the exam question contains "Data Warehouse" you should be thinking "BigQuery is a candidate".

If the case says something about large media files, you should immediately be thinking Cloud Storage.



Building and Operationalizing Pipelines

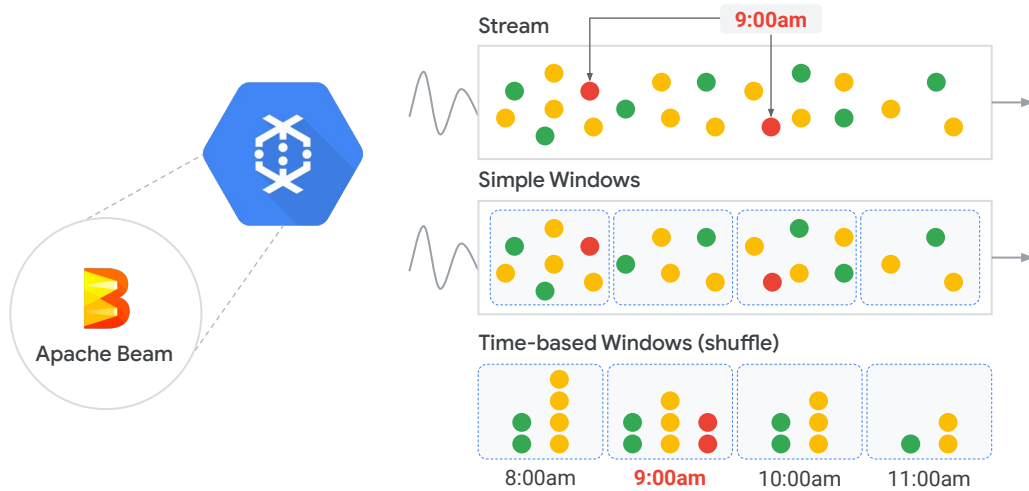
Tom Stern



The next section is on Building and Maintaining pipelines.

We've already covered a lot of this information in the design section.

Dataflow does batch and streaming



Apache Beam is an open programming platform for unifying batch and streaming.

Before Apache Beam, you needed two pipelines to balance latency, throughput, and fault tolerance.

Dataflow is Apache Beam as a service; a fully-managed autoscaling service that runs Beam pipelines.

Continuous data can arrive out of order. Simple windowing can separate related events into independent windows, losing relationship information. Time-based windowing (shuffling) overcomes this limitation.

Dataflow solves many stream processing issues

Size	Scalability and Fault-tolerance	Programming Model	Unbounded data
Autoscaling and rebalancing handles variable volumes of data and growth.	On-demand and distribution of processing scales with fault tolerance.	Efficient pipelines (Apache Beam) + efficient execution (Dataflow).	Windowing, triggering, incremental processing, and out-of-order/late data are addressed in the streaming model.

Dataflow resources are deployed on demand, per job, and work is constantly rebalanced across resources.

Dataflow solves many stream processing issues, including changes in size (spikes) and growth over time. It can scale while remaining fault tolerant. And it has a flexible programming model and methods to work with data arriving late or out of order.

Dataflow windowing for streams

To compute averages on streaming data, we need to bound the computation within time windows.

Triggering controls how results are delivered to the next transforms in the pipeline.

Watermark is a heuristic that tracks how far behind the system is in processing data from the event time. Where in event time does processing occur?

Fixed, **sliding**, and **session-based** windows.

Updated results (late), or **speculative results** (early).

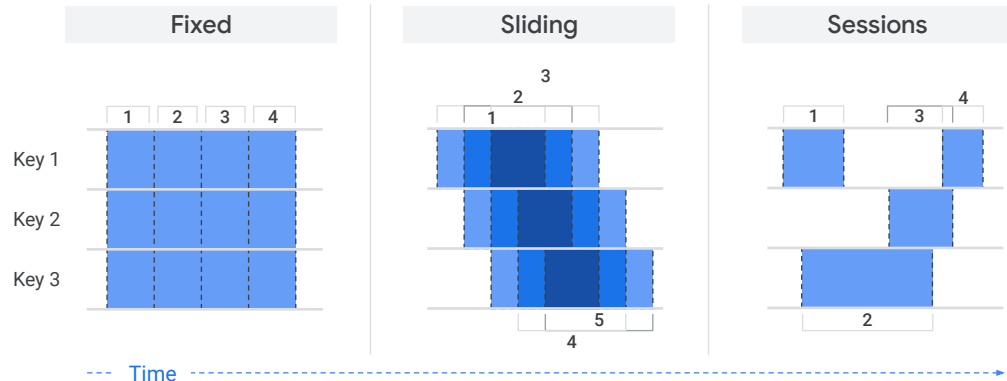
All data processing is behind or lags events simply due to latency in the delivery of the event message.

Windowing is too complicated to explain here. I just want to highlight that you might need to know it, so make sure you understand it.

There really is no replacement for the Dataflow windowing capability for streaming data.

Windows are the answer to "Where in event time?"

Windowing divides data into event time-based finite chunks.



Often required when doing aggregations over unbounded data.

Windowing creates individual results for different slices of event time.

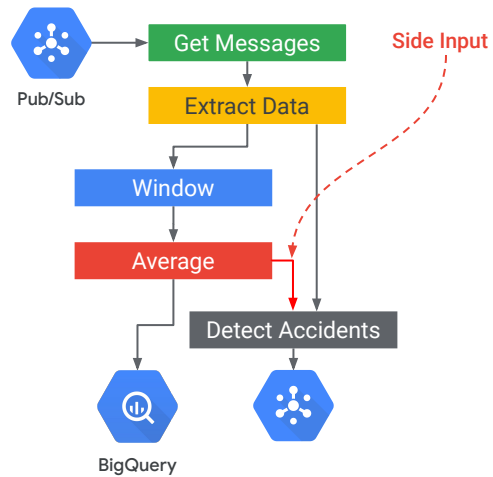
Windowing divides a PCollection up into finite chunks based on the event time of each message. It can be useful in many contexts but is required when aggregating over infinite data.

Do you know the basic windowing methods, including Fixed time (such as a daily window), Sliding and overlapping windows (such as the last 24 hours), and session-based windows that are triggered to capture bursts of activity?

Side inputs in Dataflow

Pipeline to detect accidents

DetectAccidents uses the average speed at each location as a side input.

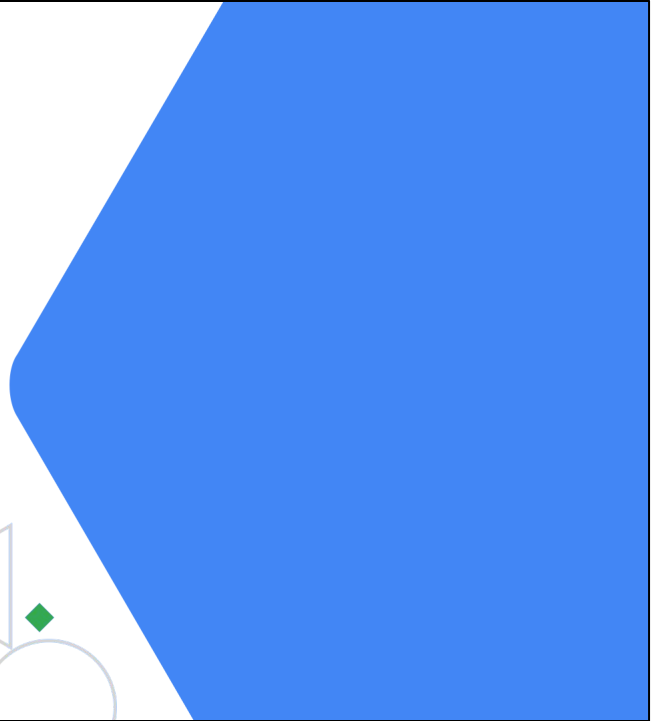


Remember to study side-inputs. If you understand side-inputs you will necessarily understand many dependent concepts that are part of Dataflow.



Building and Operationalizing Processing Infrastructure

Tom Stern



We have covered a lot of information about the processing infrastructure already.

Just a few points about building and maintaining.

Building a streaming pipeline

Stream from Pub/Sub into BigQuery; BigQuery can provide streaming ingest to unbounded data sets.

BigQuery provides streaming ingestion at a rate of 100,000 rows/table/second.

Pub/Sub guarantees delivery, but not the order of messages. "At least once" means that repeated delivery of the same message is possible. Dataflow stream processing can remove duplicates based on internal Pub/Sub ID and can work with out-of-order messages when computing aggregates.

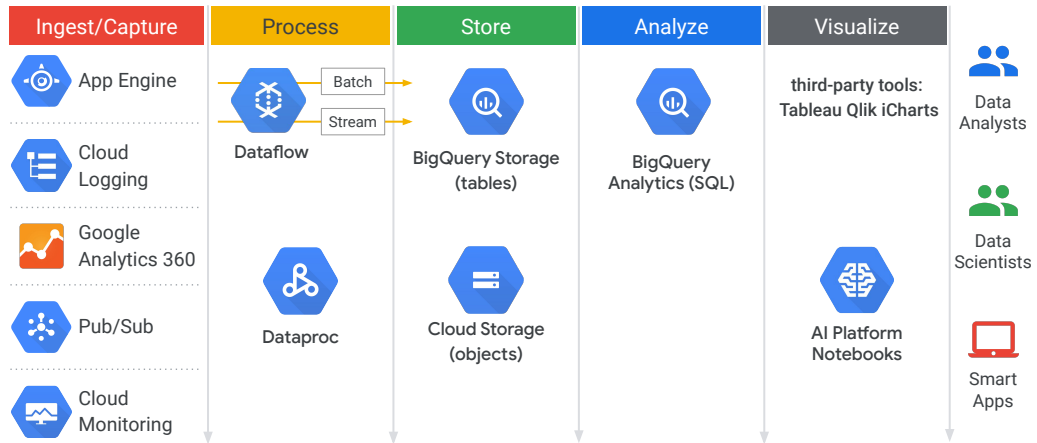
All data processing is behind or lags events simply due to latency in the delivery of the event message.

You can stream unbounded data into BigQuery. But it maxes out at 100,000 rows per table per second.

Pub/Sub guarantees delivery but might deliver the messages out of order.

If you have a timestamp, then Dataflow can remove duplicates and work out the order of messages.

Data processing solutions



BigQuery is an inexpensive data store for tabular data. It is cost-comparable with Cloud Storage, so it makes sense to ingest into BigQuery and leave the data there.

This diagram is useful because it shows the progression and options for input and visualization on the edges of the common solution design.

Scaling streaming beyond BigQuery

BigQuery

Easy, inexpensive

- latency in order of seconds
- 100k rows/second streaming

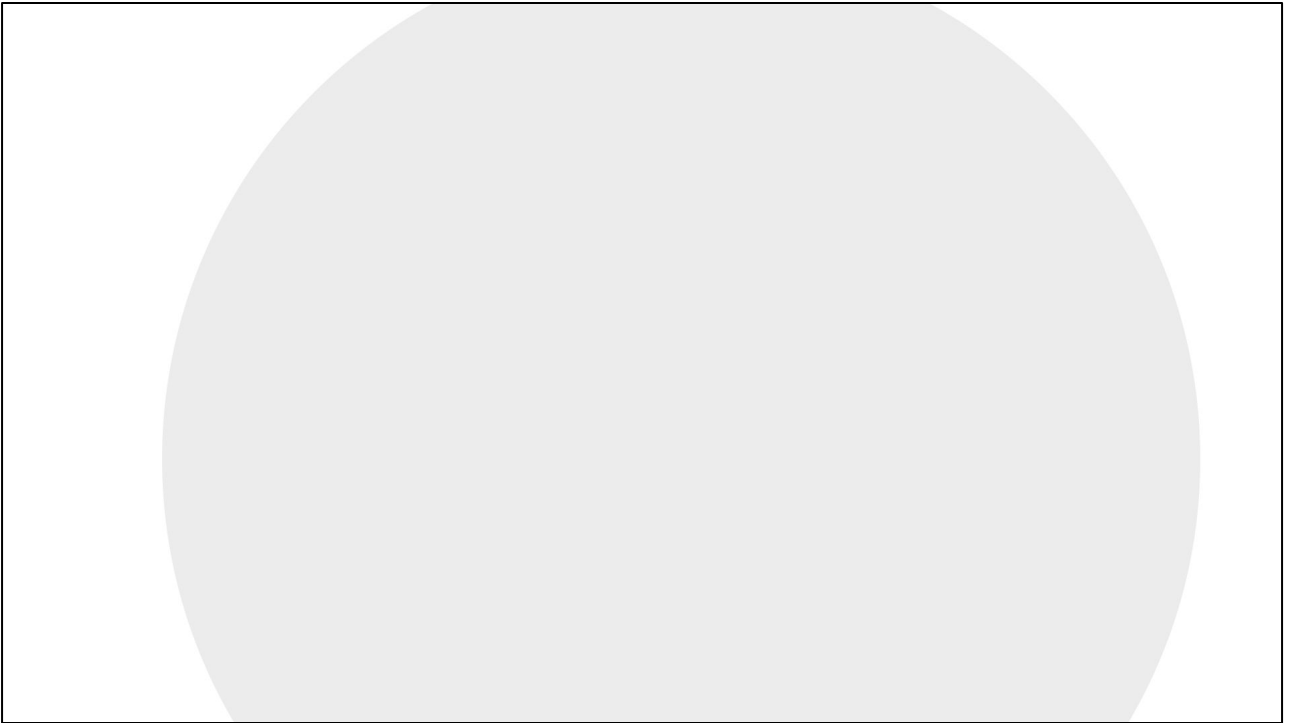
Cloud Bigtable

Low latency/high-throughput

- 100,000 QPS at 6ms latency for a 10-node cluster

Why Bigtable and not Cloud Spanner? Cost!

Note that we can support 100,000 qps with 10 nodes in Bigtable, but would need ~150 nodes in Cloud Spanner.



Time for some more practice exam questions.

Here's the first one.

Scenario #1

Question

An application that relies on Cloud SQL to read infrequently changing data is predicted to grow dramatically. How can you increase capacity for more read-only clients?

- A. Configure high availability on the Master node.
- B. Establish an external replica in the customer's data center.
- C. Use backups, so you can restore if there is an outage.
- D. Configure read replicas.

An application that relies on Cloud SQL to read infrequently changing data is predicted to grow dramatically.

How can you increase capacity for more read-only clients?

- Configure high availability on the Master node.
- Establish an external replica in the customer's data center.
- Use backups, so you can restore if there is an outage.
- Configure read replicas.

Do you have your answer?

Scenario #1

Answer

An application that relies on Cloud SQL to read infrequently changing data is predicted to grow dramatically. How can you increase capacity for more read-only clients?

- A. Configure high availability on the Master node.
- B. Establish an external replica in the customer's data center.
- C. Use backups, so you can restore if there is an outage.
- D. Configure read replicas.

The answer is D, configure Read Replicas.

Do you know why?

Scenario #1

Rationale

D is correct.

A: High Availability does nothing to improve throughput; it makes the service more accessible.

B: An external replica is more of a backup/D.R. activity; it doesn't add to throughput on the cloud.

C: Backups would not make sense in this scenario.

Refer to the link in the course notes for more information.

The clue is that the clients are read-only and need the challenge is scale.

Read Replicas increase capacity for simultaneous reads.

Note that a high availability configuration wouldn't help in this scenario because it would not necessarily increase throughput.

Ready for another question?

<https://cloud.google.com/sql/docs/mysql/replication#read-replicas>

Scenario #2

Question

A BigQuery dataset was located near Tokyo. For efficiency reasons, the company wants the dataset duplicated in Germany.

- A. Change the dataset from a regional location to multi-region location, specifying the regions to be included.
- B. Export the data from BigQuery into a bucket in the new location, and import it into a new dataset at the new location.
- C. Copy the data from the dataset in the source region to the dataset in the target region using BigQuery commands.
- D. Export the data from BigQuery into a nearby bucket in Cloud Storage. Copy to a new regional bucket in Cloud Storage. Import into the new dataset in the new location.

A BigQuery dataset was located near Tokyo. For efficiency reasons, the company wants the dataset duplicated in Germany.

- Change the dataset from a regional location to multi-region location, specifying the regions to be included.
- Export the data from BigQuery into a bucket in the new location, and import it into a new dataset at the new location.
- Copy the data from the dataset in source region to the dataset in the target region using BigQuery commands.
- Export the data from BigQuery into nearby bucket in Cloud Storage. Copy to a new regional bucket in Cloud Storage. Import into the new dataset in the new location.

Ready for the answer?

Scenario #2

Answer

A BigQuery dataset was located near Tokyo. For efficiency reasons, the company wants the dataset duplicated in Germany.

- A. Change the dataset from a regional location to multi-region location, specifying the regions to be included.
- B. Export the data from BigQuery into a bucket in the new location, and import it into a new dataset at the new location.
- C. Copy the data from the dataset in the source region to the dataset in the target region using BigQuery commands.
- D. Export the data from BigQuery into a nearby bucket in Cloud Storage. Copy to a new regional bucket in Cloud Storage. Import into the new dataset in the new location.

Export the data from BigQuery to Cloud Storage, copy to another location in Cloud Storage, and import into the new dataset in the new location.

Scenario #2

Rationale

D is correct. BigQuery imports and exports data to local or multi-regional buckets in the same location. So you need to use Cloud Storage as an intermediary to transfer the data to the new location.

A, B, and C are incorrect.

A: Datasets are immutable, so the location can't be updated.

B: BigQuery writes and reads from nearby buckets, so the new location can't read the old location data.

C: BigQuery doesn't provide a location-to-location move or copy command.

Refer to the link in the course notes for more information.

BigQuery imports and exports data to local or multi-regional buckets in the same location. So you need to use Cloud Storage as an intermediary.

Ready for one more?

<https://cloud.google.com/bigquery/docs/dataset-locations>

Scenario #3

Question

Your client wants a transactionally consistent global relational repository.
You need to be able to monitor and adjust node count for unpredictable traffic spikes.

- A. Use Cloud Spanner. Monitor storage usage and increase node count if more than 70% utilized.
- B. Use Cloud Spanner. Monitor CPU utilization and increase node count if more than 70% utilized for your time span.
- C. Use Cloud Bigtable. Monitor data stored and increase node count if more than 70% utilized.
- D. Use Cloud Bigtable. Monitor CPU utilization and increase node count if more than 70% utilized for your time span.

A transactionally consistent global relational repository where you can monitor and adjust node count for unpredictable traffic spikes.

- Use Cloud Spanner. Monitor storage usage and increase node count if more than 70% utilized.
- Use Cloud Spanner. Monitor CPU utilization and increase node count if more than 70% utilized for your time span.
- Use Cloud Bigtable. Monitor data stored and increase node count if more than 70% utilized.
- Use Cloud Bigtable. Monitor CPU utilization and increase node count if more than 70% utilized for your time span.

Got your answer?

Scenario #3

Answer

Your client wants a transactionally consistent global relational repository.
You need to be able to monitor and adjust node count for unpredictable traffic spikes.

- A. Use Cloud Spanner. Monitor storage usage and increase node count if more than 70% utilized.
- B. Use Cloud Spanner. Monitor CPU utilization and increase node count if more than 70% utilized for your time span.
- C. Use Cloud Bigtable. Monitor data stored and increase node count if more than 70% utilized.
- D. Use Cloud Bigtable. Monitor CPU utilization and increase node count if more than 70% utilized for your time span.

Use Cloud Spanner, monitor CPU utilization and increase the number of nodes as needed.

Scenario #3

Rationale

B is correct because of the requirement to globally scalable transactions—use Cloud Spanner. CPU utilization is the recommended metric for scaling, per Google best practices, linked below.

A is not correct because you should not use storage utilization as a scaling metric.

C, D are not correct because you should not use Cloud Bigtable for this scenario.

Refer to the links in the course notes for more information.

B is correct because of the requirement to globally scalable transactions—use Cloud Spanner.

CPU utilization is the recommended metric for scaling, per Google best practices.

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

<https://cloud.google.com/bigtable/docs/monitoring-instance>

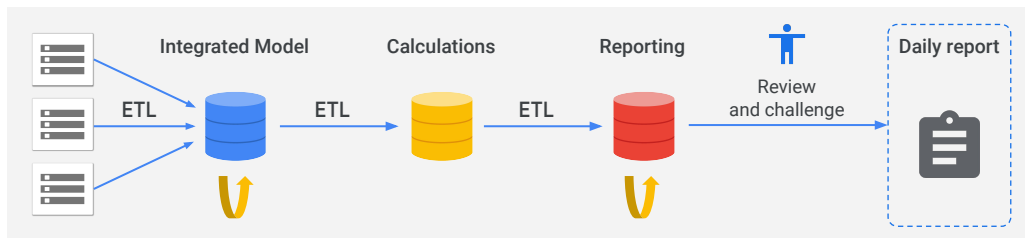
Data Engineer Case Study 01:

A customer had this interesting business requirement...

A daily reporting pipeline with multiple sources and complex dependencies

Human intervention to data check quality, inputs, and proceed to next stage

Need daily updates on yesterday's data, but takes >24 hours to run.



Case Study 01

I worked with a client that had a very complex reporting pipeline. They had a very large amount of data that had to be processed into a report that was going to regulators on a daily basis. They had to demonstrate that the risk in the financial data from the previous day indicated that they were following the regulatory rules. They put the data through multiple systems or stages. Each stage had a separate Extract-Transform-Load sequence and then performed unique processing on the data.

A customer had this interesting business requirement...

A daily reporting pipeline with multiple sources and complex dependencies

- Human intervention to data check quality, inputs, and proceed to next stage
- Need daily updates on yesterday's data, but takes >24 hours to run

The complexity in the simplified diagram makes it look like it was a linear progression from start-to-finish. But this is meant to be symbolic. The actual processes were much more complicated and it took 30 hours from start to finish to generate one report. The processes were actually more of a spider-web, with many dependencies. So one part would run and then halt, waiting until other dependent parts were complete before proceeding. And there were some processes that could run in parallel.

Data Engineer Case Study 01:

We mapped that to technical requirements like this...

BigQuery and Cloud Composer (aka Apache Airflow)

BigQuery: Reduce overall time to run with BigQuery as data warehouse and analytics engine.

Apache Airflow: Control to automate pipeline, handle dependencies as code, start query when preceding queries were done.

First we diagrammed out all the processes. And then we started to look at how to implement this on Google Cloud using the available services. We initially considered using Dataproc or Dataflow. However, the customer already had analysts that were familiar with BigQuery and SQL. So if we developed in BigQuery it was going to make the solution more maintainable and usable to the group. If we developed in Dataproc, for example, they would have had to rely on another team that had Spark programmers. So this is an example where the technical solution was influenced by the business context.

To make this solution work, we needed some automation. And for that we chose Apache Airflow. In the original design we ran Airflow on a Compute Engine instance. You might be familiar with the Google service called Cloud Composer, which provides a managed Apache Airflow service. Cloud Composer was not yet available when we began the design.

BigQuery and Cloud Composer (aka Apache Airflow).

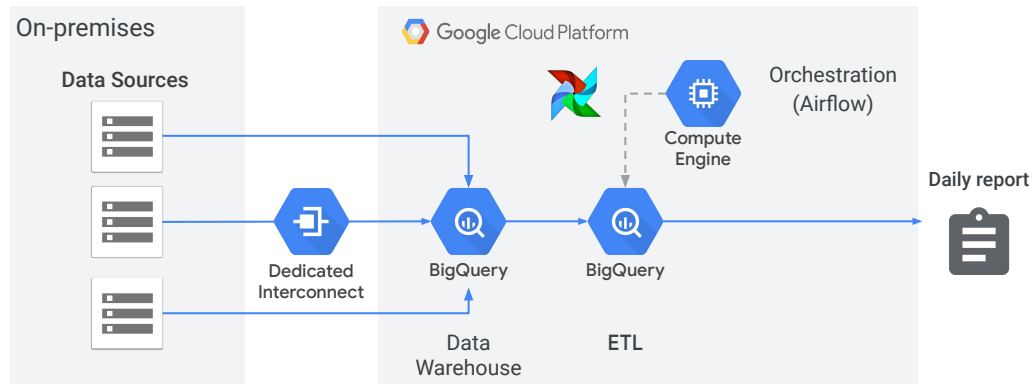
BigQuery: Reduce overall time to run with BQ as data warehouse and analytics engine.

Apache Airflow: Control to automate pipeline, handle dependencies as code, start query when preceding queries were done.

Data Engineer Case Study 01:

And this is how we implemented that technical requirement

Common data warehouse in BigQuery. Apache Airflow to automate query dependencies.



And this is how we implemented that technical requirement.

Common data warehouse in BigQuery. Apache Airflow to automate query dependencies. In this particular case we used open source Apache Airflow. But if we were implementing it today we would use Cloud Composer.

Cloud Composer / Apache Airflow allowed us to establish the dependencies between different queries that existed in the original reporting process. BigQuery served as both the data storage solution and the data processing / query solution.

We were able to implement all their processing as SQL queries in BigQuery. And we were able to implement all the dependencies through Airflow.

One of the time-sinks in their original process had to do with that 30 hour start-to-finish window. What they would do is start processing jobs and sometimes they would fail because the data from a previous dependency wasn't yet available. And they had a manual process for restarting those jobs. We were able to automate away that toil and the re-work by implementing the logic in Apache Airflow.

Challenge Lab 01

PDE Prep—BigQuery Essentials:
Challenge Lab

 :45



A Challenge Lab has minimal instructions. It explains the circumstance and the expected results; you have to figure out how to implement them.

This is a timed lab.

The lab will expire after 45 minutes.

The lab can be completed in 30 minutes.

The Qwiklabs system will track and verify your progress in completing the lab.

