Designing and Building

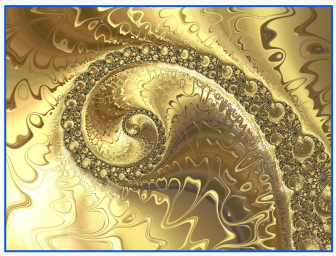
This module covers the first two major parts of the Exam outline: designing and building.

The outline is repetitious. For example, "Building data and data representations" and later "Maintaining data and data representations" and "Optimizing data and data representations."

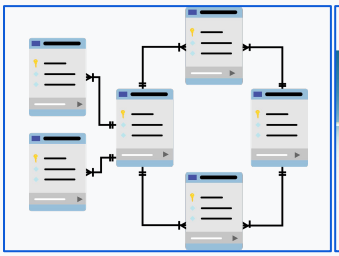
In this course the technology or information is introduced where it makes the most sense in the context of the discussion. And specific aspects or properties of the technology are surfaced later. For example, Cloud SQL is introduced as a database technology in Designing and Building. Access to a Cloud SQL server is covered in the section on security later in the course.

Designing Data Processing Systems

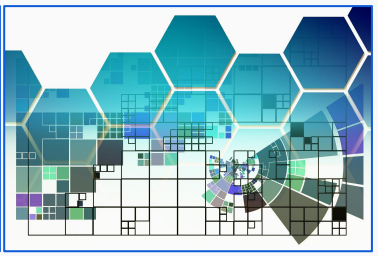
Data processing anatomy



What data?
Representation



What transformations or operations?
Pipeline



By what processing service?
Infrastructure

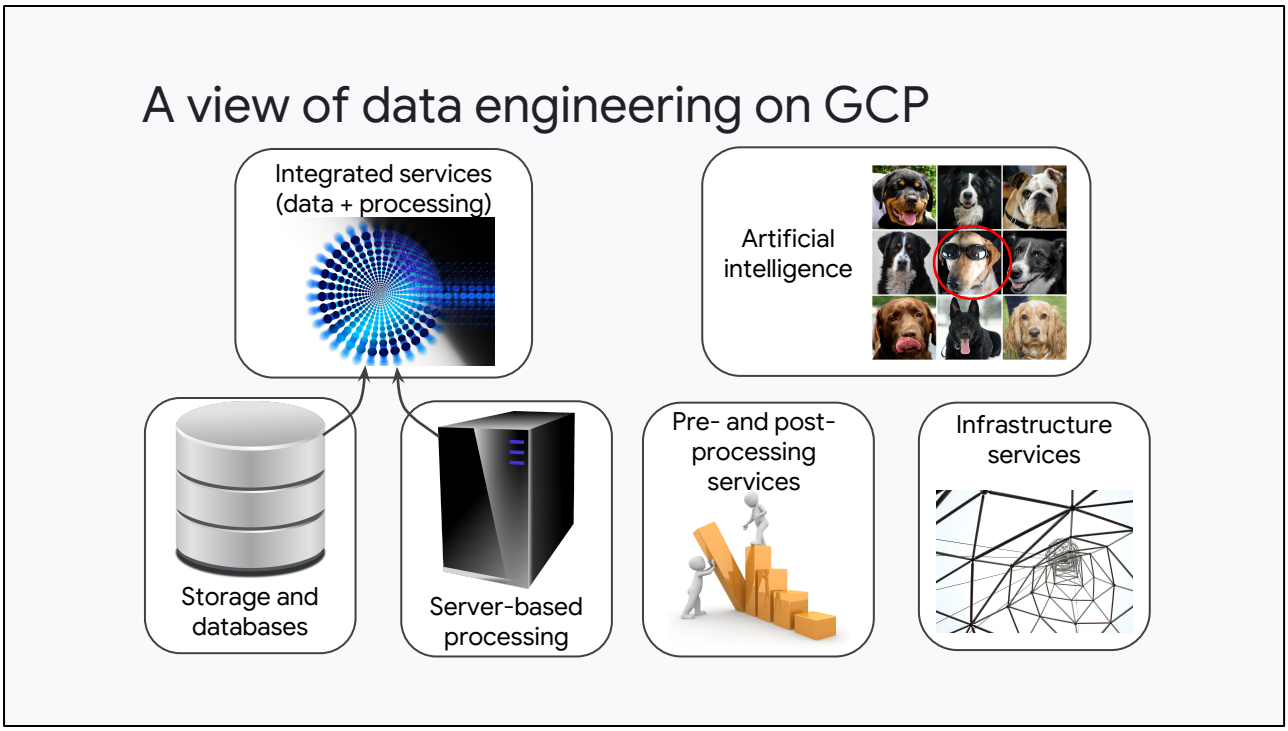
3

Overview. You will see that these three items show up in the first part of the exam with similar, but not identical, considerations.

Same questions or interests show up in different contexts. Data representation, pipelines, processing infrastructure. For example, innovations in the technology could make the data representation of a chosen solution outdated. The data processing pipeline might have implemented a very involved transformation that is now available as a single efficient command. And the infrastructure could be replaced by a service with more desirable qualities.

However, as you'll see, there are additional concerns with each part. For example, "System Availability" is important to pipeline processing, but it is not important to data representation. And "Capacity" is important to processing, but it is not important to abstract structure of a pipeline or to the data representation.

https://pixabay.com/en/database-schema-data-tables-schema-1895779/
https://pixabay.com/en/fractal-golden-background-aesthetic-952750/
https://pixabay.com/en/block-chain-data-records-concept-3145392/



TIP: Think about data engineering in Google Cloud as a platform consisting of components that can be assembled into solutions.

Storage and databases: Services that enable storing and retrieving data; differ in storage and retrieval methods that make them more efficient for specific use cases.

Server-based processing: Services that enable application code and software to run that can make use of stored data to perform operations—actions and transformations—producing results.

Integrated services: Combines storage and scalable processing in a framework design to process data (rather than general applications). More efficient and flexible than isolated server/database solutions.

Artificial intelligence: Method to help **identify** (tag), **categorize**, and **predict**; three actions that are very hard or impossible to accomplish in data processing without machine learning.

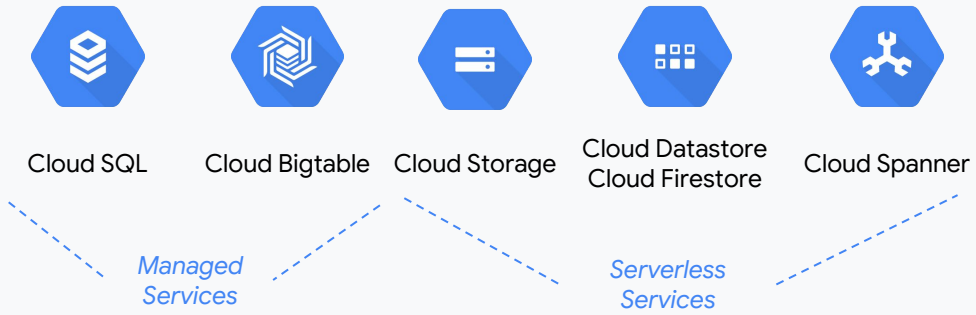
Pre-and-post services: Working with data and pipelines before processing (such as data cleanup) or after processing (such as data visualization). Pre- and post-processing are important parts of a data processing solution.

Infrastructure services: All the framework services that connect and integrate data processing and IT elements into a complete solution. Messaging, systems, data

import/export, security, monitoring, etc.

https://pixabay.com/en/database-data-storage-information-309919/
https://pixabay.com/en/server-computer-case-computer-pc-309012/
https://pixabay.com/en/data-computer-internet-online-www-2899899/
https://pixabay.com/en/graph-success-cooperation-together-1019845/
https://pixabay.com/en/dog-collage-canines-looking-1738398/
https://pixabay.com/en/scaffold-frame-tv-tower-reception-1790016/

Storage and databases



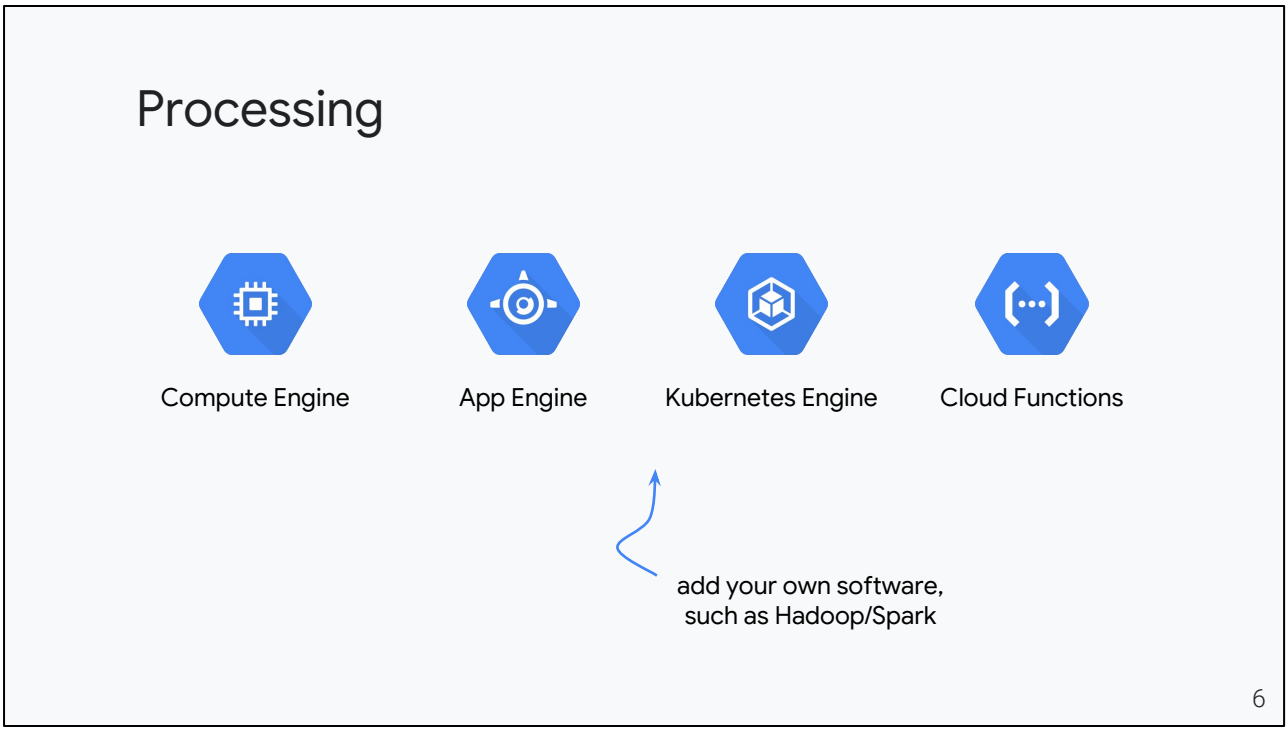
5

Store and retrieve. Organization and access methods of the different services is designed to be efficient for specific cases.
Note that these services do not include data processing or the definition of a processing pipeline.
They support minimal operations that are integral to accessing the data. For example, SQL, is a sophisticated access method, and not focused on data transformation or processing.

Differentiators include access methods, the cost or speed of specific actions, size of data, and organization method.
Details/differences between the services are specified later in this course.

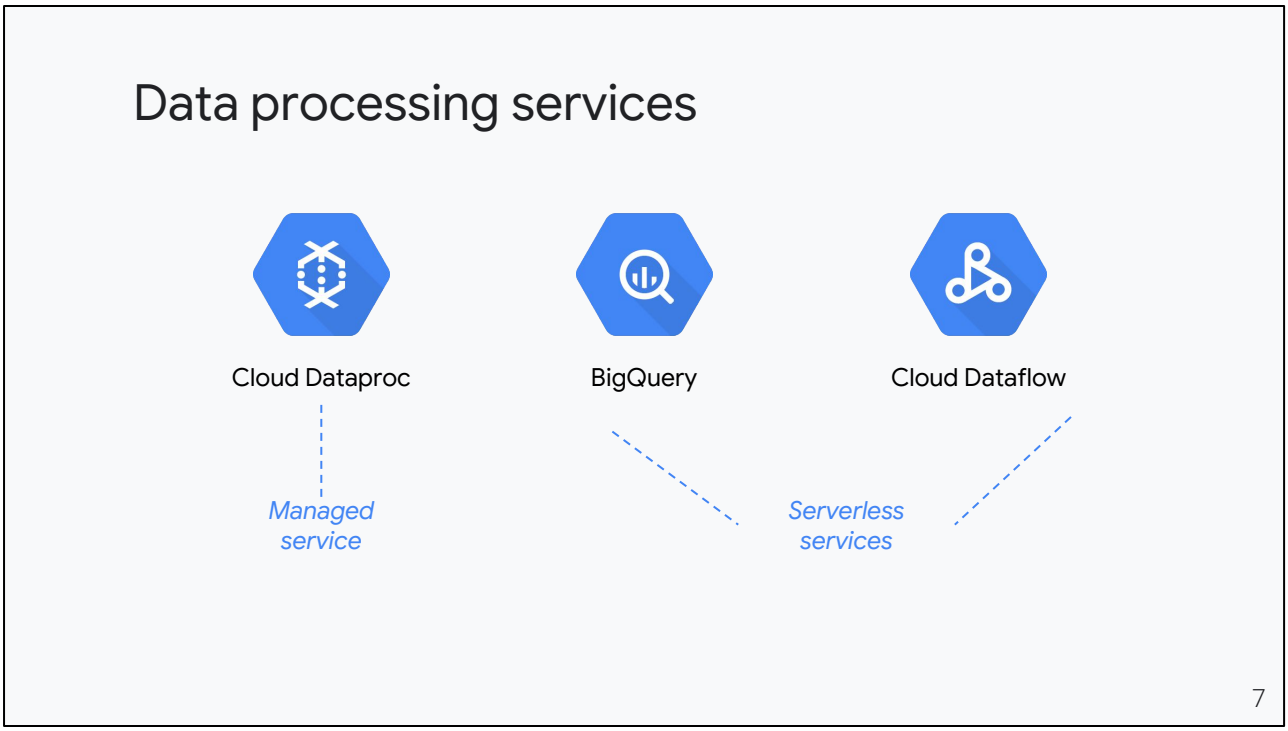
Cloud Firestore is a NoSQL document database built for automatic scaling, high performance, and ease of application development. It includes a "Datastore Mode."
https://cloud.google.com/firestore/

A managed service: You can see the instance or cluster.
A serverless service: You can't see individual instances or clusters.



TIP: Data processing services provide solutions with less IT overhead. Computing solutions in DE are driven mostly by business requirements.

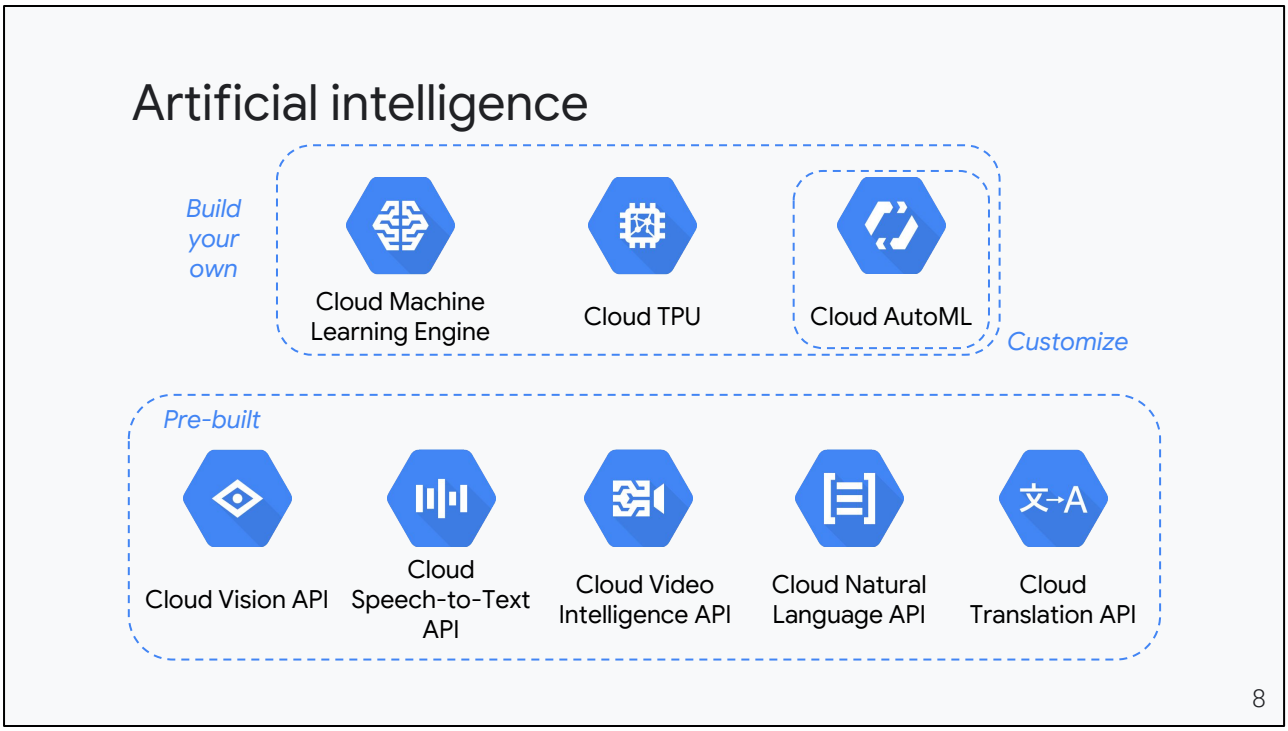
Storage and databases provide limited processing in the context of search and retrieval. But if you need to perform more sophisticated actions and transformations on the data, you will need data processing software. You could use any of the computing platforms to write your own application or parts of an application. Business logic supporting the data engineering solution can be implemented in a computing platform. You could also install software such as MySQL (open source database) or Hadoop (open source data processing).



TIP: These three data processing services feature in almost every data engineering solution. Each overlaps with the other, meaning that some work could be accomplished in either two or three of these services. Advanced solutions may use one, two, or all three.

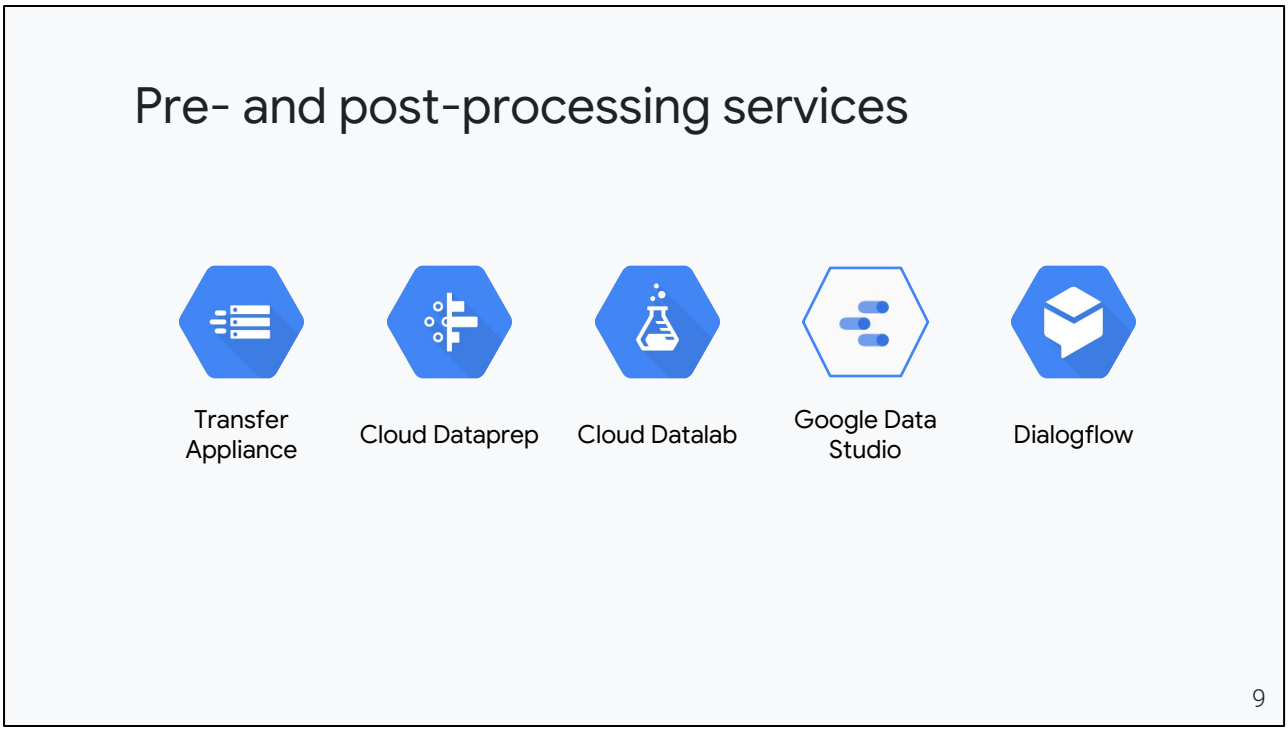
Data processing services combine storage and compute and automate the storage and computing aspects of data processing through abstractions. For example, in Cloud Dataproc, the data abstraction (with Spark) is a **Resilient Distributed Dataset** (RDD), and the processing abstraction is a **Directed Acyclic Graph** (DAG). In BigQuery, the abstractions are **Table** and **Query**. And in Cloud Dataflow, the abstractions are **PCollection** and **Pipeline**.

Implementing storage and processing as abstractions enables the underlying systems to adapt to the workload and the user/data engineer to focus on the data and business problems they are trying to solve.



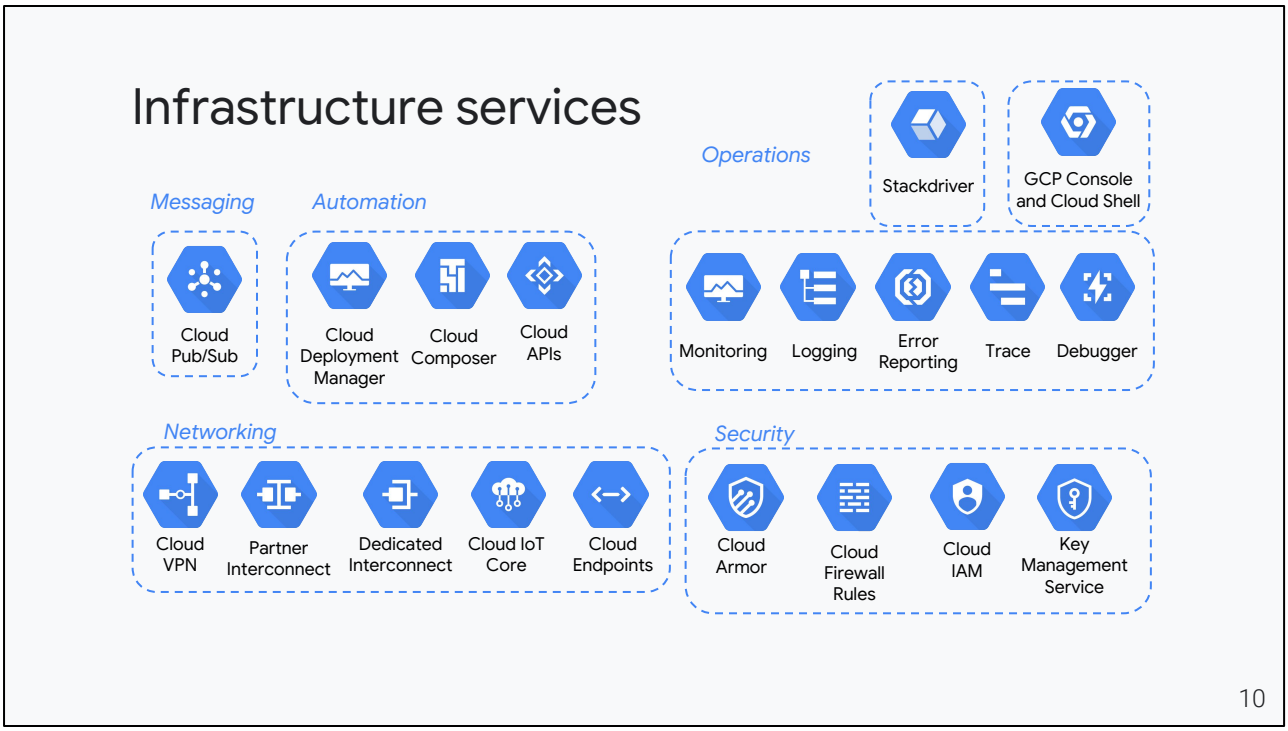
TIP: There is great potential value in product or process innovation by using machine learning to make unstructured data, such as logs, useful by identifying or categorizing the data, enabling business intelligence.
TIP: Recognizing an instance of something that exists is closely related to predicting a future instance based on past experience.

Machine learning is used for identifying, categorizing, and predicting. It can make unstructured data useful.



TIP: A data engineering solution involves data ingest, management during processing, analysis, and visualization. These elements can be critical to the business requirements.

Data Transfer Services/Data Appliance: Getting/synchronizing data with external source.
Data Studio: Visualization after the processing.
Cloud Dataprep: Prepare/condition data and/or pipelines before processing.
Cloud Datalab: Notebook
Dialogflow - Chatbot: Iinteraction with AI and data. Provides a method for direct human interaction with data.



TIP: Familiarize yourself with services that show up commonly in data engineering solutions. Often they are employed because of key features they provide. For example, Cloud Pub/Sub can hold a message for up to seven days, providing resiliency to data engineering solutions that otherwise would be very difficult to implement.

Every service in Google Cloud Platform could be used in a data engineering solution. However, some of the most common and important services are listed. **Cloud Pub/Sub**, a messaging service, features in virtually all live or streaming data solutions because it decouples data arrival from data ingest. **Cloud VPN**, **Partner Interconnect**, or **Dedicated Interconnect** play a role whenever there is data on-premises that must be transmitted to services in the cloud. **Cloud IAM**, **Firewall Rules**, and **Key Management** are critical to some verticals, such as the healthcare and financial industries. And every solution needs to be monitored and managed, which usually involves panels displayed in **GCP Console** and data sent to **Stackdriver Monitoring**.

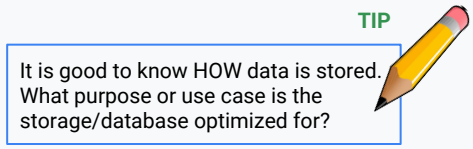
TIP: It is a good idea to examine sample solutions that use data processing or data engineering technologies and pay attention to the infrastructure components of the solution. It is important to know what the services provide to data solutions, but also to be familiar with key features and options.

Design flexible data representations

The key concept we will explore is understanding how data is stored and therefore how it is processed.
There are different abstractions for storing data. And if you store data in one abstraction instead of another, it makes different processes easier or faster.

For example, if you store data in a file system, it makes it easier to retrieve that data by name.
If you store data in a database, it makes it easier to find data by logic, such as SQL.
And if you store data in a processing system, it makes it easier and faster to transform the data, not just retrieve it.

Your data in...



Your data in **Cloud Storage** is an **Object** stored in a **Bucket**.

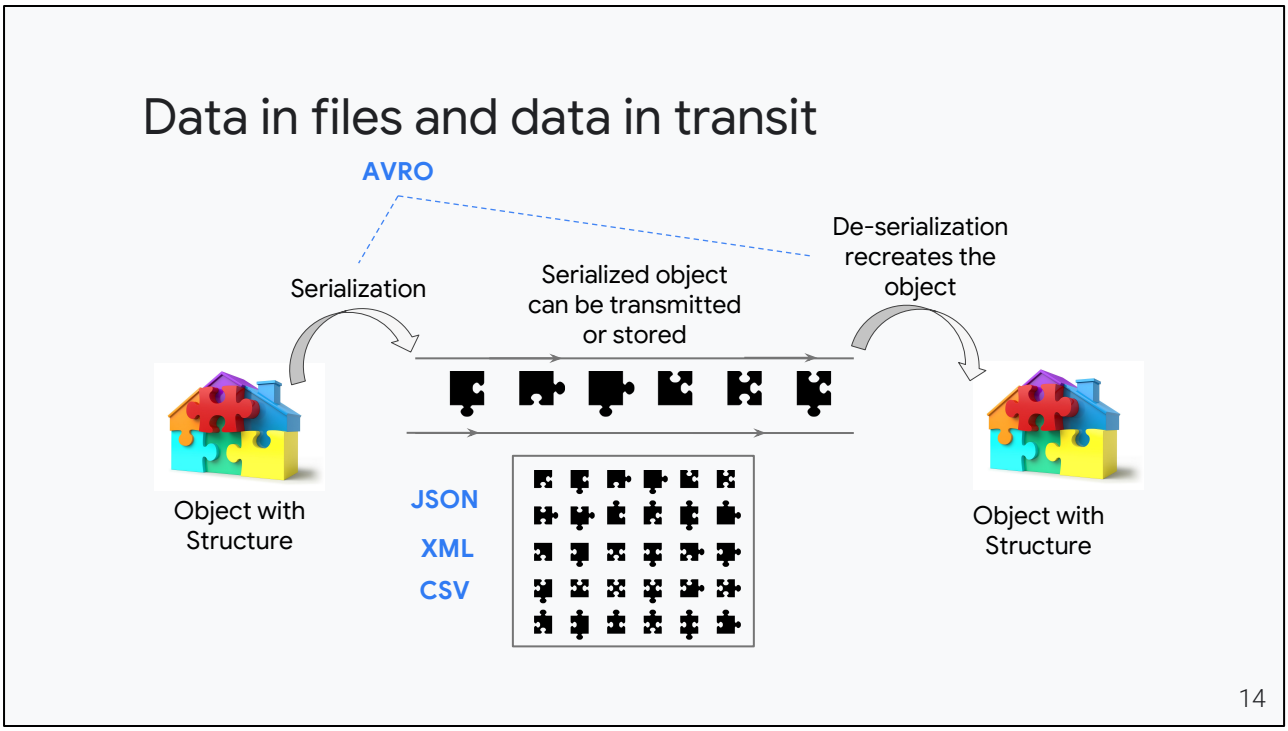
Your data in **Cloud Datastore** is a **property,** contained in an **Entity** and is in a **Kind** category.

Your data in **Cloud SQL** consists of **Values** stored in **Rows** and **Columns** in a **Table** in a **Database**.

Your data in **Cloud Spanner** consists of **Values** stored in **Rows** and **Columns** in a **Table** in a **Database**.

For the job, and therefore to help you on the exam, you need to know the terminology and basic concepts of data representation.
For example, if a problem is described using the terms rows and columns, since those concepts are used in SQL, you might already be thinking about a SQL database such as Cloud SQL or Cloud Spanner.

If an exam question describes an Entity and a Kind—which are concepts used in Cloud Datastore—and you don't know what they are, you will have a difficult time answering the question.
You won't have time or resources to look these up during the exam. You need to know them going in.



Avro is a remote procedure call and data serialization framework. It was developed within Apache's Hadoop project. It uses JSON for defining data types and protocols and it serializes data in a compact binary format. Its primary use is in Apache Hadoop, where it can provide a serialization format for persistent data. It can also provide a wire format for communication between Hadoop nodes and from client programs to the Hadoop services.

CSV is a simple file format used to store tabular data.
XML was designed to store and transport data and was designed to be self-descriptive.
JSON (JavaScript Object Notation) is a lightweight data-interchange format based on name/value pairs and an ordered list of values, which maps easily to common objects in many programming languages.

https://pixabay.com/en/puzzle-pieces-house-shape-2648214/
https://pixabay.com/en/jigsaw-puzzle-pieces-game-32249/

Flat, serialized data is easy to work with, but it lacks structure and therefore meaning. If you want to represent data that has meaningful relationships, you need a method that represents not only the data but also the relationships. The most popular and important of these representation methods are CSV, JSON, and XML. CSV stands for *comma-separated values*, and it is a natural way to represent data in a matrix or table. Data is separated into columns by commas. An end-of-line sequence means

the end of the row. JSON uses curly braces to delimit data and then represents data in pairs or tuples that consist of a key and an associated value. JSON can really represent a wide range of structures. It maps to a Python Dictionary variable, if you are familiar with those. So it is a very powerful abstraction, human readable, and easy to work with in code. XML uses markup language as delimiters, so it maps very well to HTML for web pages.

Presently, networking transmits serial data—a stream of bits—in zeroes and ones. Also, presently, storage medium stores data as a series of zeroes and ones. That means if you have a data object with structure in memory and you need to store it or transmit it, the object will have to be flattened and serialized first. And it has to be done in such a way that the object can be recreated with structure when it is later received or retrieved. AVRO is one example of a framework that does this. It is used mainly in Hadoop. And it provides both a format for serialization of persistent data and a format for wire transfer of data.

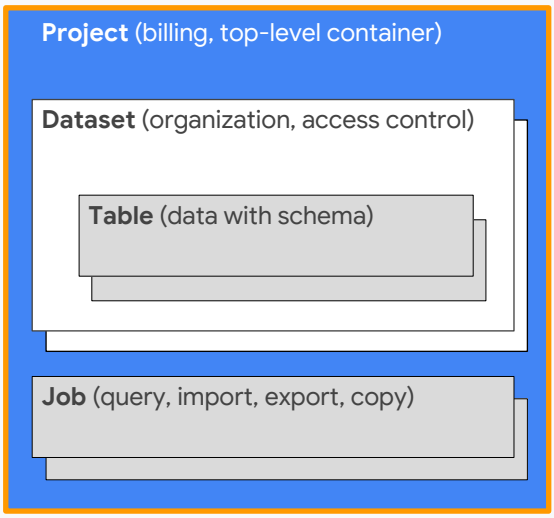
Standard SQL data types

Data type	Possible value
`STRING`	Variable-length character (Unicode) data
`INT64`	64-bit integer
`FLOAT64`	Double-precision (approximate) decimal values
`BOOL`	True or false (case-insensitive)
`ARRAY`	Ordered list of zero or more elements of any non-ARRAY type
`STRUCT`	Container of ordered fields, each with a type (required) and field name (optional)
`TIMESTAMP`	Represents an absolute point in time, with precision up to microseconds. Values range from the years 1 to 9999, inclusive.

For more information on all data types supported by standard SQL, see: https://cloud.google.com/bigquery/sql-reference/data-types.

It helps to understand the data types supported in different representation systems. For example, there is a data type in modern SQL called NUMERIC. NUMERIC is similar to floating point. However, it provides a 38-digit value with 9 digits to represent the location of the decimal point. NUMERIC is very good at storing common fractions associated with money. NUMERIC avoids the rounding error that occurs in a full floating point representation, so it is used primarily for financial transactions.

BigQuery datasets, tables, and jobs



A project contains users and datasets
Use a project to:

- Limit access to datasets and jobs
- Manage billing

A dataset contains tables and views
Access Control Lists for Reader/Writer/Owner
Applied to all tables/views in dataset

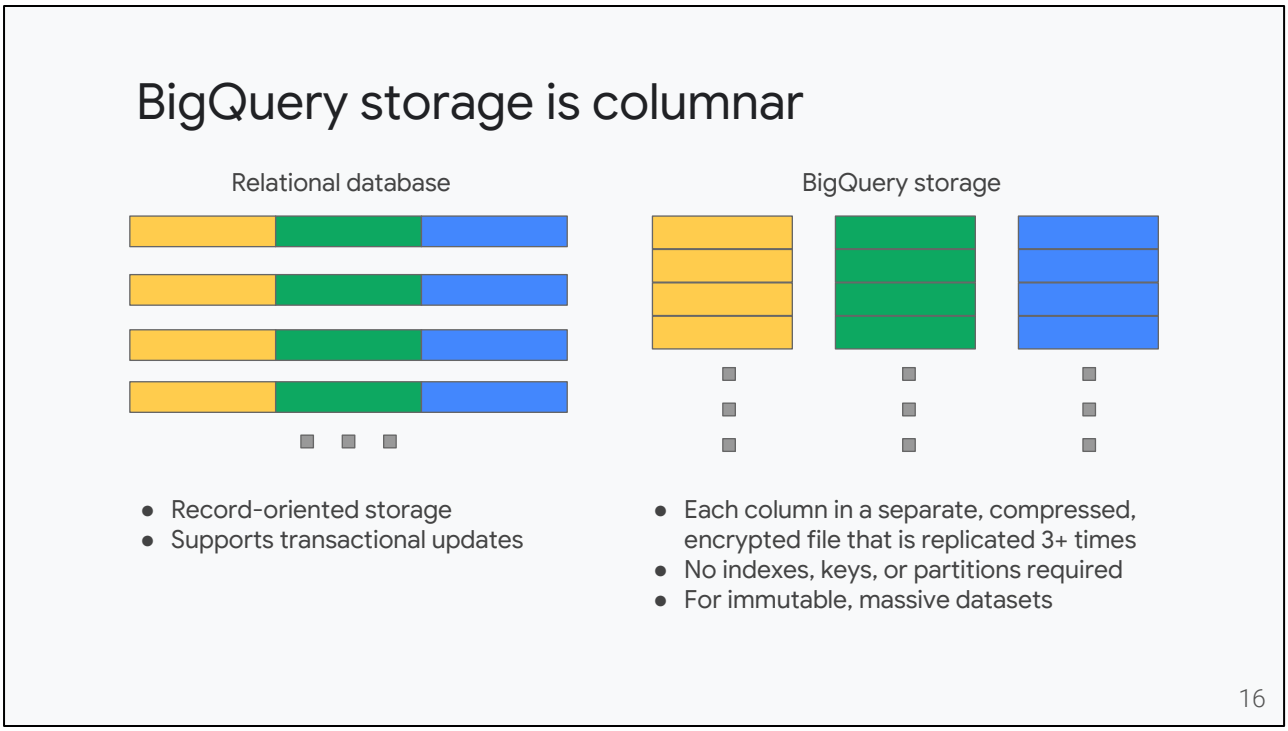
A table is a collection of columns
Columnar storage
Views are virtual tables defined by SQL query
Tables can be external (e.g., on Cloud Storage)

A job is a potentially long-running action
Can be cancelled

TIP: Your data in BigQuery is in Tables in a Dataset.

Here is an example of the abstractions associated with a particular technology. You should already know that every resource in GCP exists inside a Project. And in addition to security and access control, a Project links usage of a resource to a credit card: it is what makes a resource billable.

Then in BigQuery, data is stored inside datasets. And datasets contain tables. And tables contain columns. When you process the data, BigQuery creates a job. Often the job runs an SQL query. Although there are some update and maintenance activities supported."



TIP: Access in BigQuery is optimized by column.

BigQuery stores data in columns.

Most queries only work on a small number of fields, and BigQuery only needs to read those relevant columns to execute a query. Because each column has data of the same type, BigQuery could compress the column data much more effectively.

You can stream (append data) easily to BigQuery tables, but not change existing values.
Replicating the data 3+ times also helps in finding optimal compute nodes to do filtering, mixing, etc.

BigQuery is called a "columnar store," meaning that it is designed for processing columns, not rows. Column processing is very cheap and fast in BigQuery, and row processing is slow and expensive.

Spark hides data complexity with an abstraction: RDDs

17

TIP: Your data in Cloud Dataproc/Spark is in Resilient Distributed Datasets, or RDDs.

RDDs hide the complexity of the location of data within the cluster and also hide the complexity of replication.
Spark partitions data in memory across the cluster and knows how to recover through an RDD's lineage, if anything goes wrong.
Spark can direct processing to occur where there is a processing resource available.
You treat your data as a single entity; Spark knows the truth.
Data partitioning, data replication, data recovery, pipelining of processing: all are automated by Spark so you don't have to worry about them.

https://pixabay.com/en/books-door-entrance-italy-colors-1655783/
https://pixabay.com/en/book-red-closed-library-education-34014/

Cloud Dataflow terms and concepts: PCollection

Each step is a Transform

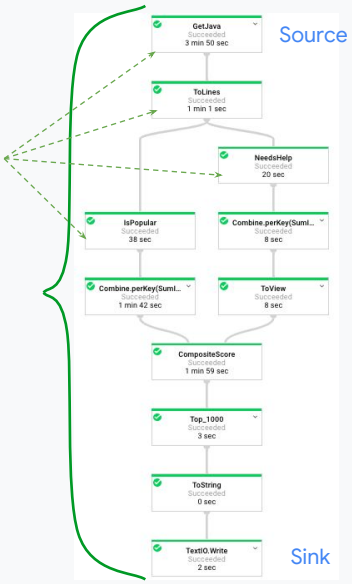
Each step is elastically scaled.

Each Transform is applied on a PCollection.

The result of an `apply()` is another PCollection.

Together, they form a Pipeline

The Pipeline is executed on the cloud by a Runner



Cloud Dataflow involves several concepts that you should know about. In Cloud Dataflow, each step is a transformation, and the collection of transforms makes a pipeline. The entire pipeline is executed by a program called a *runner*. For development there is a local runner, and for production there is a cloud runner. When the pipeline is running on the cloud, each step, each transform, is applied to a PCollection and results in a PCollection. So the PCollection is the unit of data that traverses the pipeline. And each step scales elastically.

This diagram identifies key concepts to be familiar with in Cloud Dataflow.

TIP: Your data in Cloud Dataflow is in PCollections.

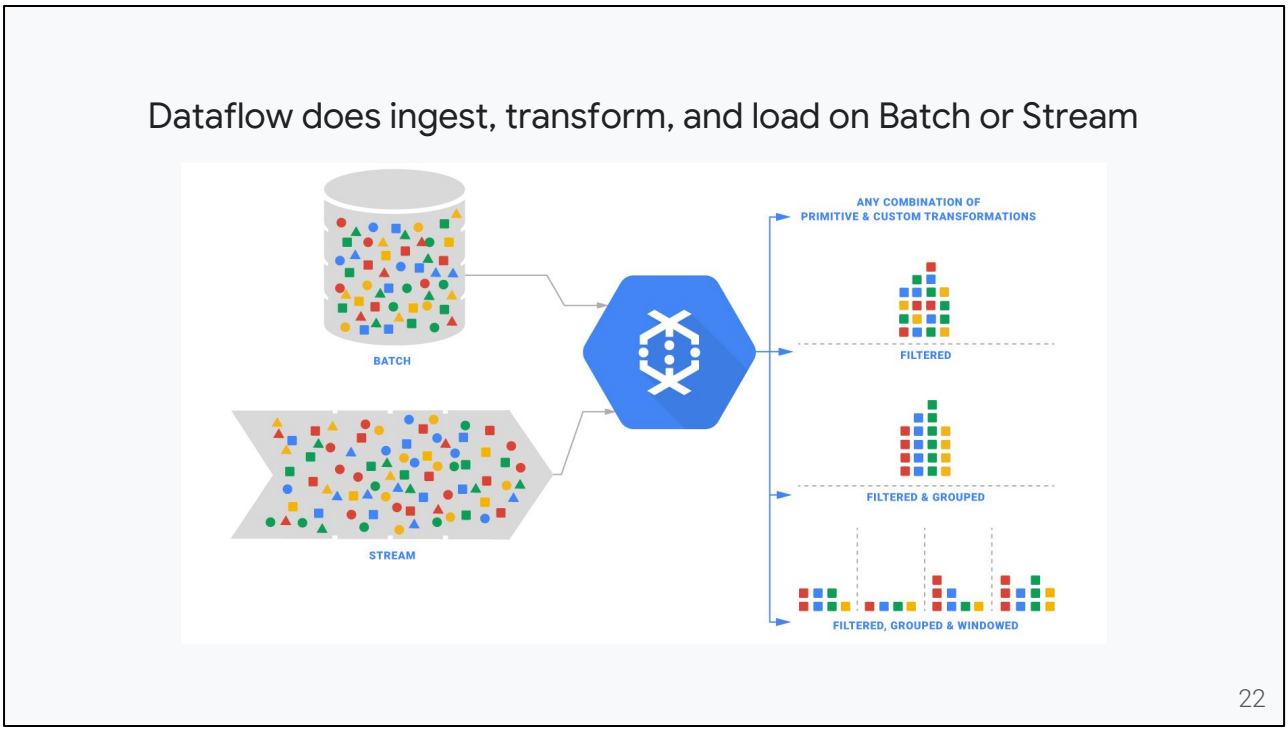
The idea is to write Java (or Python code) and deploy it to Cloud Dataflow, which then executes the pipeline.

The pipeline here reads data from BigQuery, does some processing, and writes its output to Cloud Storage.

Elastic: unlike with Cloud Dataproc, there is no need to launch a cluster. It is like BigQuery in that respect.

In a Cloud Dataflow pipeline, all the data is stored in a PCollection. The input data is a PCollection. Transformations make changes and output a PCollection.
A PCollection is immutable. That means you don't modify it. Every time you pass data

through a transformation, it creates another PCollection.

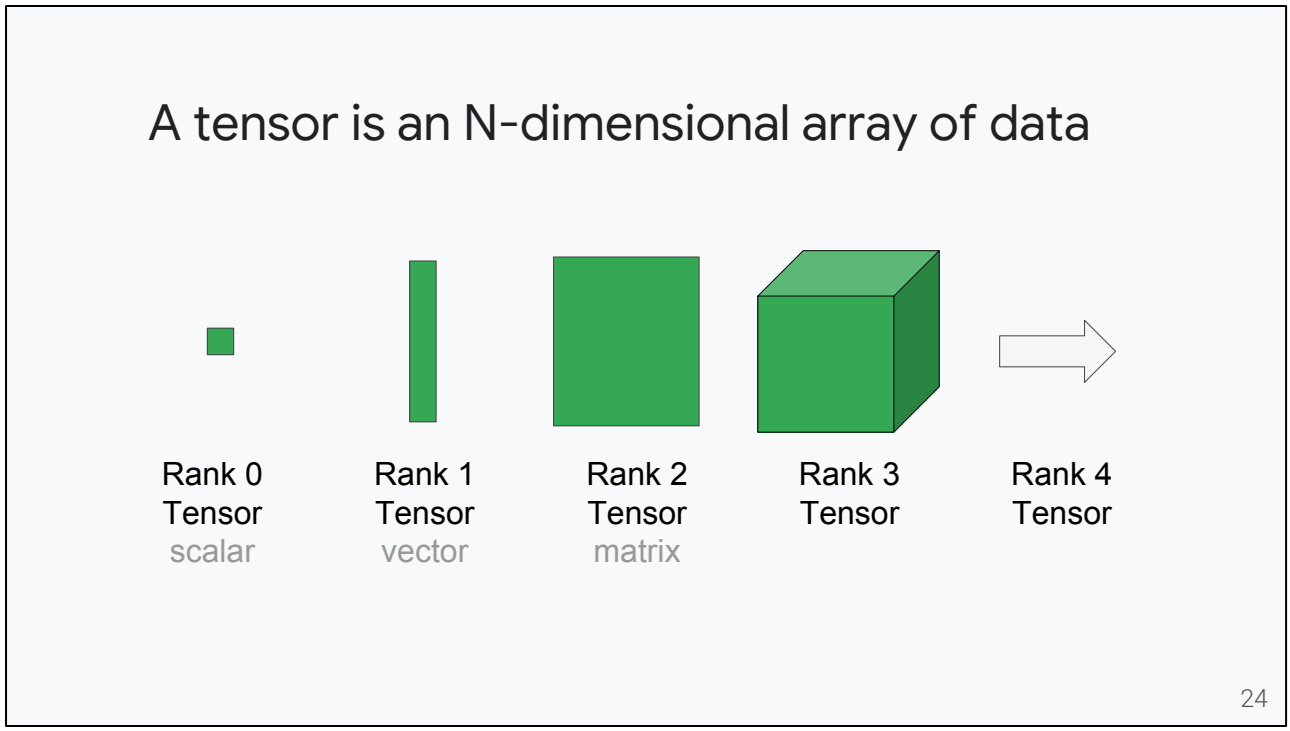


Dataflow uses the same pipeline, the same code, to work on batch and stream-type data. Remember that batch data is also called *bounded data*, and it is usually a file. Batch data has a finite end. Streaming data is also called *unbounded data*, and it might be dynamically generated. For example, it might be generated by sensors or by sales transactions. Streaming data just keeps going, day after day, year after year, with no defined end. Algorithms that rely on a finite end won't work with streaming data; for example, a simple average, where you add up all the values and divide by the number of values. That is fine with batch data, because eventually you will have all the values. But that doesn't work with streaming data because there may be no end. So Cloud Dataflow allows you to define a period or window and to calculate the average within the window. That's an example of how both kinds of data can be processed with one kind of code. Filtering and grouping are also supported.

TIP: Cloud Dataflow is designed to use the same operations for both batch and stream, also called *bounded* and *unbounded* data.

You can replace all the various data handling tools with just Cloud Dataflow. Many Hadoop workloads can be run more easily and are easier to maintain with Cloud Dataflow.

We use a PCollection in Cloud Dataflow to represent both batch data and streaming data. There is a benefit here because you can use the same code and the same pipeline for dealing with both kinds of data.



Where does the name TensorFlow come from? Well, the "flow" is a pipeline, just like we discussed in Cloud Dataflow. But the data object in TensorFlow is not a PCollection, but something called a *Tensor*. A Tensor is a special mathematical object that unifies scalars, vectors, and matrixes. Tensor-zero is just a single value, a scalar. Tensor-one is a vector, having direction and magnitude. Tensor-two is a matrix. Tensor-three is a cube shape. Tensors are very good at representing certain kinds of math functions, such as coefficients in an equation. And TensorFlow makes it possible to work with tensor data objects of any dimension.

TIP: Your data in TensorFlow is in tensors.

So where does the name (‘tensorflow’) come from?

In math, a simple number like 3 or 5 is called a scalar.

A vector is a one-dimensional array of numbers. In physics, a vector is something with magnitude and direction. But in computer science, we use *vector* to mean 1D arrays.

A two-dimensional array is a matrix.

A three-dimensional array? We just call it a 3D tensor.

So scalar, vector, matrix, 3D tensor, 4D tensor, etc.

A tensor is an n-dimensional array of data.

So, your data in TensorFlow is tensors.
They flow through the graph. Hence, TensorFlow.

TensorFlow is the open source code that you use to create machine learning models. In TensorFlowm the data is called a *tensor*. In TensorFlow, you define a process graph. The transformation occurs when the data is passed through or flows through the graph. What is a tensor? A tensor is an object that can have any kind of dimensions. For example, if you have a value that is a simple number or simple boolean or a single character, it is stored in a scalar, and object of dimension zero. If you have two values, it is a vector. And if you have three values, one could be the x coordinate, one could be the y coordinate, and the third could be the value at that location. So that is a matrix. You could have something with 5 dimensions or 7 dimensions. And we don't have special words for those. The idea of the tensor links together these relationships into a data type. A tensor is a powerful abstraction because it relates different kinds of data types. And there are transformations in tensor algebra that apply to any dimension of tensor. So it makes solving some problems much easier.

Design data pipelines

Cloud Dataproc

Cluster node options

Single node (for experimentation)

Standard (1 master only)

High availability (3 masters)

Benefits

Hadoop: familiar

Automated cluster mgmt

Fast cluster resize

Flexible VM configurations

HDFS

Use Cloud Storage for a stateless solution.

HBASE

Use Cloud Bigtable for a stateless solution.

Objectives:

Shut down the cluster when it is not actually running jobs.

Start a cluster per job or for a particular kind of work.

Data storage

Don't use hdfs to store input/output data; use it for temporary working storage

Disk performance SCALES with size!

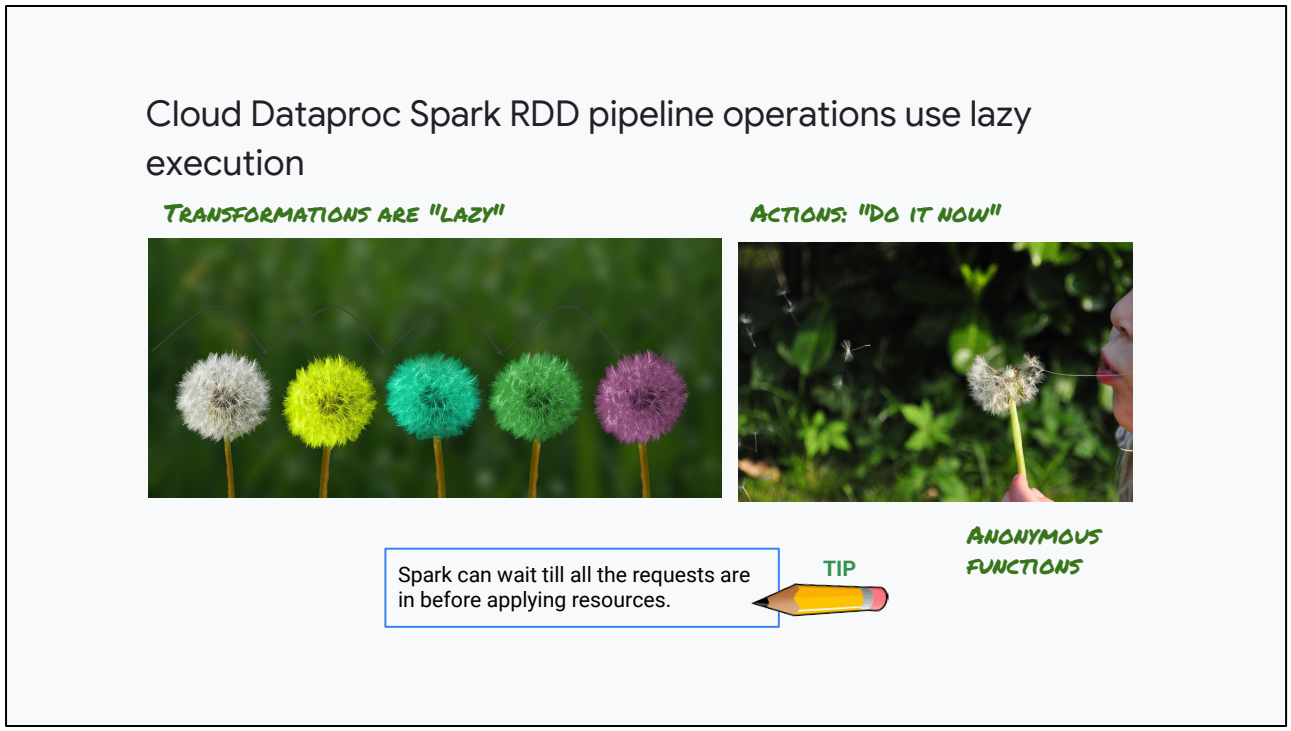
Cloud storage

Match your data location with your compute location; zone matters.

Machine types and preemptible VMs

\Cloud Dataproc is a managed Hadoop service. And there are a number of things you should know, including standard software in the Hadoop ecosystem and components of Hadoop. However, the main thing you should know about Cloud Dataproc is how to use it differently from standard Hadoop. If you store your data externally from the cluster, storing HDFS-type data in Cloud Storage and storing HBASE-type data in Cloud Bigtable, then you can shut your cluster down when you are not actually processing a job. That is very important. What are the two problems with Hadoop? First, trying to tweak all of its settings so it can run efficiently with multiple different kinds of jobs, and second, trying to cost-justify utilization. So you search for users to increase your utilization. And that means tuning the cluster. And then if you succeed in making it efficient, it is probably time to grow the cluster. You can break out of that cycle with Cloud Dataproc by storing the data externally and starting up a cluster and running it for one type of work. Then shut it down when you are done.

Stateless clusters in <90 seconds
Supports Hadoop, Pig, Hive, Spark



Remember that with SPARK you have transforms and actions. Spark builds its pipeline with transforms, called a Directed Graph. But it doesn't execute the pipeline until it sees an Action. Very simply, waiting until it has the whole story, all the information, allows Spark to choose the best way to distribute the work and run the pipeline. That's called *lazy execution*.

Transformation
Input is an RDD and output is an RDD.
Registered in DAG awaiting an action (lazy).

Action
Output is a result format, such as a text file.
Triggers Spark to process the pipeline.

Transformations and Actions are API calls that reference the functions you want them to perform.

Anonymous functions (in Python, Lambda functions) are commonly used for the following reasons:

- A self-contained way to make a request to Spark.
- Lambda functions are defined "inline" making it easy to read and understand in sequence.
- They are limited to a single specific purpose.

- They don't clutter the namespace with function names for code that is only used in one place.

https://pixabay.com/en/dandelion-colorful-people-of-color-2817950/

https://pixabay.com/en/blowball-dandelion-girl-blowing-384598/

Cloud Dataproc can augment BigQuery

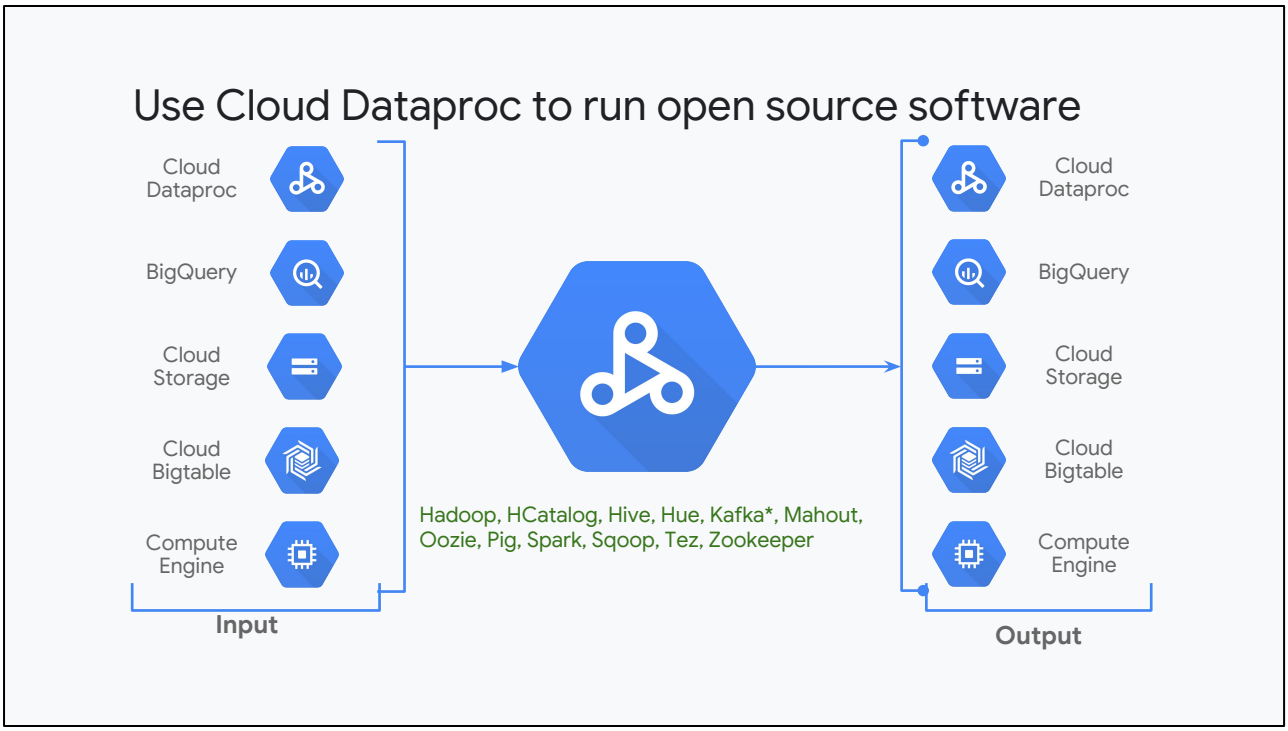
Extract data in BigQuery and pull the data into Spark cluster for further analysis.

	year	month	day	weight_pounds
1	1969	10	2	9.37626000286
2	1969	7	30	6.8122838958
3	1969	7	1	7.68751907594
4	1969	10	8	8.062304921339999
5	1969	82	24	6.686620406459999

```
projectId = <your-project-id>

sql = "
SELECT
  n.year,
  n.month,
  n.day,
  n.weight_pounds
FROM
   `bigquery-public-data.samples.natality` AS n
ORDER BY
  n.year
LIMIT 50"

print "Running query..."
data =
gbq.read_sql.gb1(sql,projectid=projectId)
data [:5]

Running query...
Requesting query… ok.
Query running...
Query done.
Processed 3.5Gb

Retrieving results.
Got 50 rows.
Time taken 1.14 s.
Finished at 2018-02-12 22:20:13
```

You can use Cloud Dataproc and BigQuery together in several ways. BigQuery is great at running SQL queries. But what it isn't built for is modifying data; real data-processing work. So if you need to do some kind of analysis that is really hard to accomplish in SQL, sometimes the answer is to extract the data from BigQuery into Cloud Dataproc and let Spark run the analysis. Also, if you needed to alter or process the data, you might read from BigQuery into Cloud Dataproc, process the data, and write it back out to another dataset in BigQuery.

TIP: If the case has data in BigQuery and perhaps the business logic is better expressed in terms of functional code instead of SQL, you may want to run a Spark job on the data.



Cloud Dataproc has connectors to all kinds of GCP resources. You can also run open source software from the Hadoop ecosystem on the cluster. It would be wise to be at least familiar with the most popular Hadoop software and to know whether alternative services exist in the cloud. For example, Kafka is a messaging service. And the alternative on GCP would be Cloud Pub/Sub.

TIP: Open source software for data engineering is always a Cloud GCP option. You can install/run Hadoop ecosystem software on the Cloud Dataproc cluster.

You can read from GCP sources and write to GCP sources and use Cloud Dataproc as the intermingling glue.

Use initialization actions to install custom software and cluster properties to configure Hadoop

Initialization actions

Optional executable scripts (Shell, Python, etc.) that run when your cluster starts.

Allow you to install additional components, stage files, or change the node.

We provide a set of common initialization actions on GitHub.

Cluster properties

Allow you to modify properties in common configuration files like core-site.xml.

Remove the need to manually change property files by hand or initialization action.

Specified by file_prefix:property=value in gcloud SDK.

TIP: Modify the cluster. Initialization action to load/install software. Not all properties can be changed. If you need to modify the cluster, consider whether you have the right data processing solution.

Cluster properties are not currently available on web UI.

If you are migrating to Cloud Dataproc from on-premises Hadoop or Hadoop hosted on VMs, you may already have customized Hadoop settings that you want to apply to the cluster within Cloud Dataproc. This is supported in a limited way via Cluster properties. Although Cloud Dataproc automatically manages the installation of software packages and cluster settings, you may want to customize these configurations in specific cases to make sure that the Cloud Dataproc cluster works similarly to your customized environment.

You can see which properties are configurable here: https://cloud.google.com/dataproc/docs/concepts/configuring-clusters/cluster-properties

Cloud Dataflow

Write Java or Python code and deploy it to Dataflow, which then executes the pipeline.

Open-source API (Apache Beam) can also be executed on Flink, Spark, etc.

Parallel tasks are autoscaled by execution framework.

Same code does real-time and batch.

You can get input from any of several sources, and you can write output to any of several sinks. The pipeline code remains the same.

You can put code inside a servlet, deploy it to App Engine, and schedule a cron task queue in App Engine to execute the pipeline periodically.

Dataflow supports side inputs, which allows different transformations on data in the same pipeline.

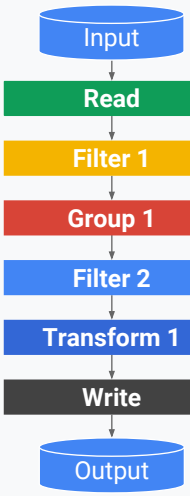
TIP

For Cloud Dataflow users, use roles to limit access to only Dataflow Resources, not just the Project.

27

Security and Permissions:
https://cloud.google.com/dataflow/security-and-permissions
Advanced access control: https://cloud.google.com/dataflow/access-control

Cloud Dataflow pipeline—documented—easier to maintain




```
Pipeline p = Pipeline.create();
p
   .apply(TextIO.read().from("gs://…"))
   .apply(ParDo.of(new Filter1()))
   .apply(new Group1())
   .apply(ParDo.of(new Filter2())
   .apply(new Transform1())
   .apply(TextIO.write().to("gs://…"));
p.run();
```

```
class Filter1 extends DoFn<...> {
  public void
  processElement(ProcessContext c)
{
      ... = c.element();
      ...
      c.output(...);
  }
```

TIP: The Cloud Dataflow pipeline not only appears in code, but also in the GCP Console as a diagram. Pipelines reveal the progression of a data processing solution, and the organization of steps makes it much easier to maintain than other code solutions.

Distinguish between the API (Apache Beam) and the implementation/execution framework (Cloud Dataflow).

Each step of the pipeline does a filter, group, transform, compare, join, etc.. Transforms can be done in parallel.

c.element() gets the input. c.output() sends the output to the next step of the pipeline.

Cloud Dataflow operations

ParDo: Allows for parallel processing

Map: 1:1 relationship between input and output in Python

FlatMap: For non 1:1 relationships, usually with generator in Python

.apply(ParDo): Java for Map and FlatMap

GroupBy: Shuffle

GroupByKey: Explicitly shuffle

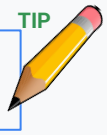
Combine: Aggregate values

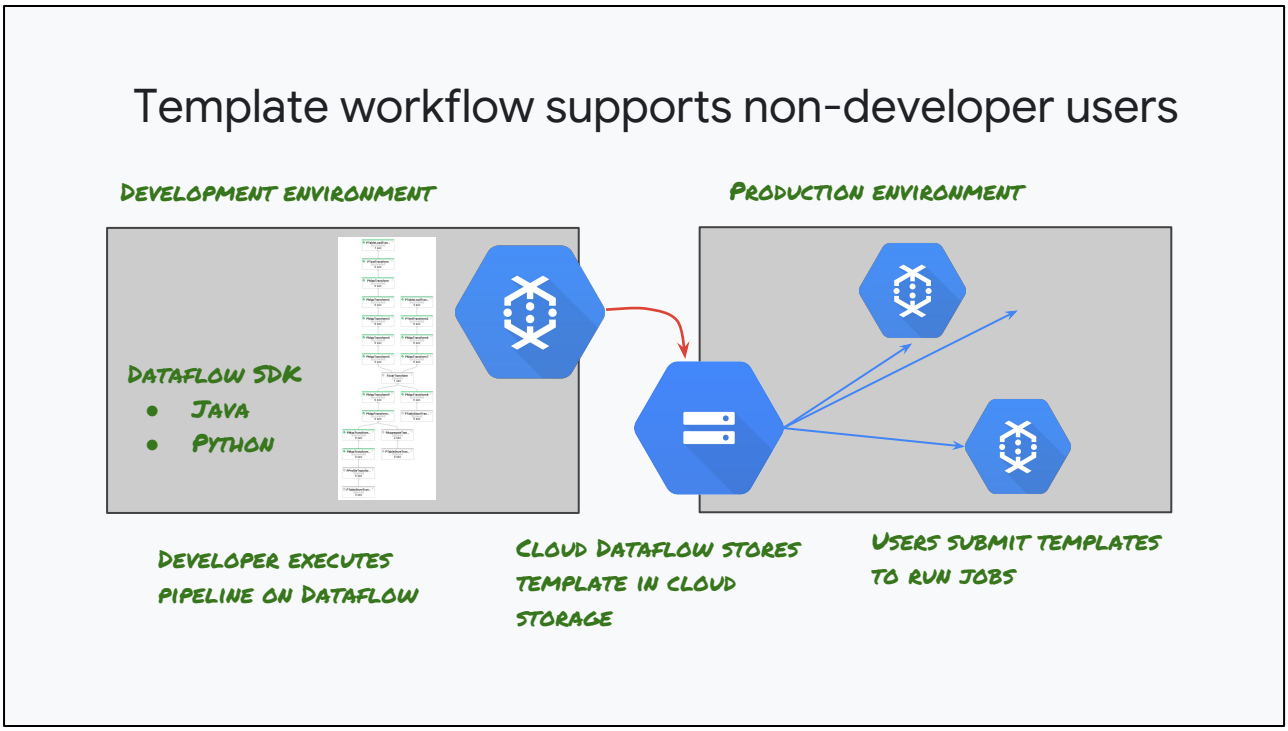
Pipelines are often organized into Map and Reduce sequences.

Side Inputs: Parallel paths of execution for making different transformations on the same source data.

TIP

A pipeline is a more maintainable and less error-prone way to organize data processing code.





TIP: Do you need to separate Cloud Dataflow developers of pipelines from Cloud Dataflow consumers/users of the pipelines? Templates create the single step of indirection that allows the two classes of users to have different access.

Cloud Dataflow templates enable a new development and execution workflow. The templates help separate the development activities and the developers from the execution activities and the users. The user environment no longer has dependencies back to the development environment. The need for recompilation to run a job is limited. The new approach facilitates the scheduling of batch jobs and opens up more ways for users to submit jobs, and more opportunities for automation.

https://cloud.google.com/dataflow/docs/templates/overview

BigQuery

Near real–time analysis of massive datasets

NoOps

Pay for use

Durable (replicated), inexpensive storage

Immutable audit logs

Mashing up different datasets to derive insights

Interactive analysis of petabyte-scale databases

Familiar, SQL 2011 query language and functions

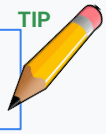
Many ways to ingest, transform, load, export data to/from BigQuery

Nested and repeated fields, user-defined functions in JavaScript

Data storage is inexpensive; queries charged on amount of data processed

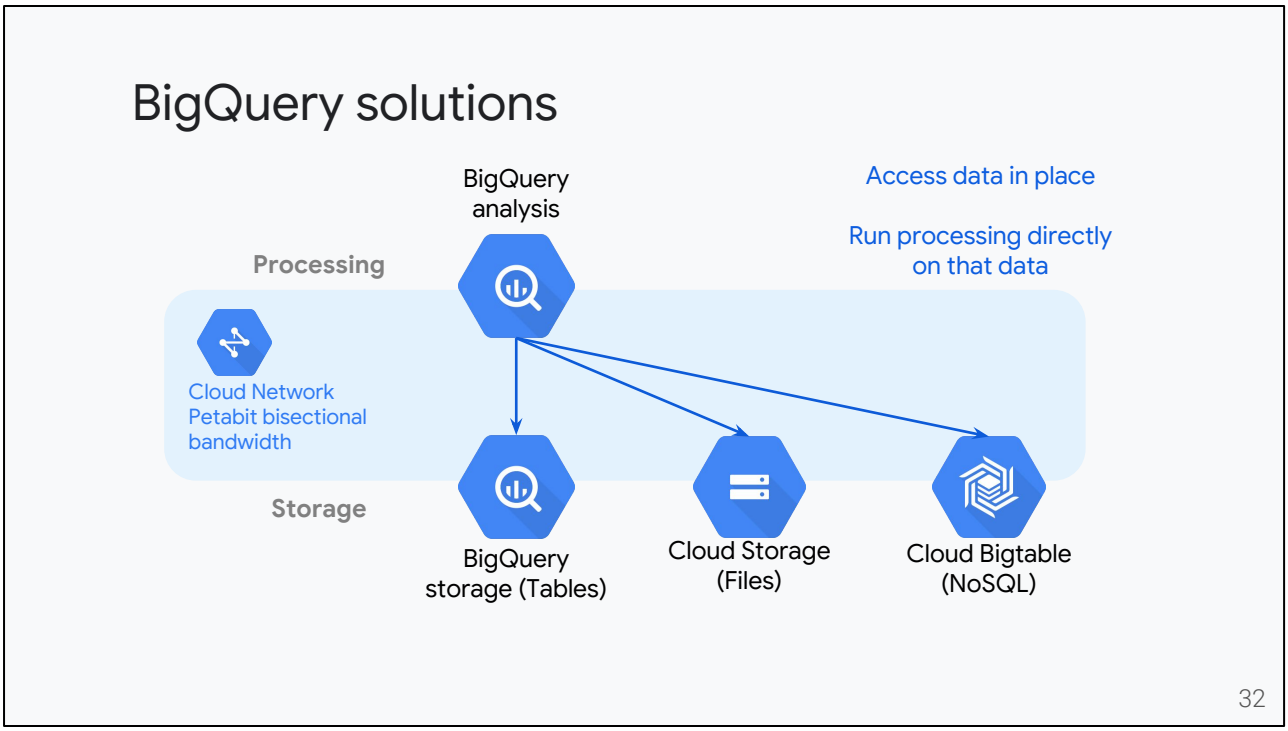
Structured data, primarily for analytics, latency of seconds is okay

BigQuery frontend does analysis.
BigQuery backend does storage.
Together: a Data Warehouse solution



Access control granularity:
Projects and Datasets

Access Control: https://cloud.google.com/bigquery/docs/access-control



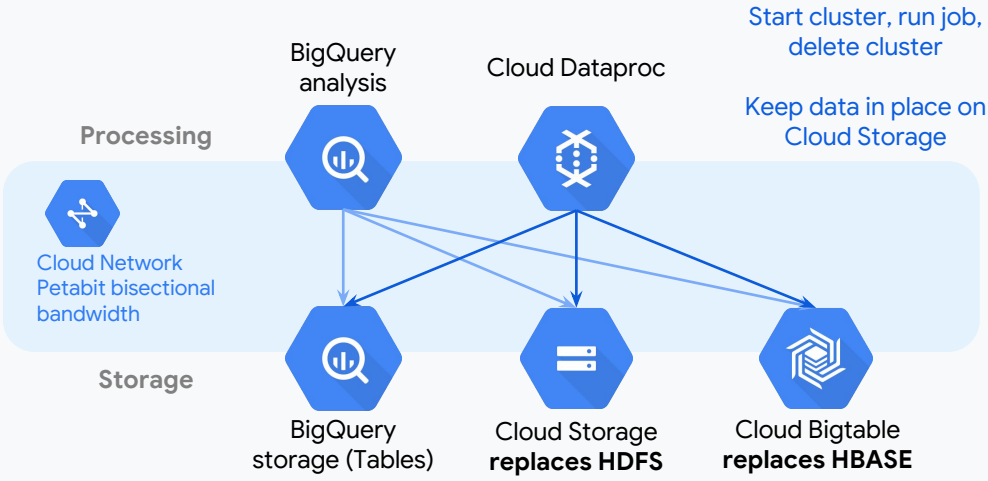
TIP: Separating Compute/Processing from Storage/Database enables serverless operations.
BigQuery has its own Analytic/SQL Query frontend that is available in the Console and from the command line with BigQuery. **It is "just" a query engine.**
The backend data warehouse part of BigQuery stores data in tables.

BigQuery has a connector to Cloud Storage. This is commonly used to work directly with CSV files.
BigQuery has a connector to Cloud Bigtable.

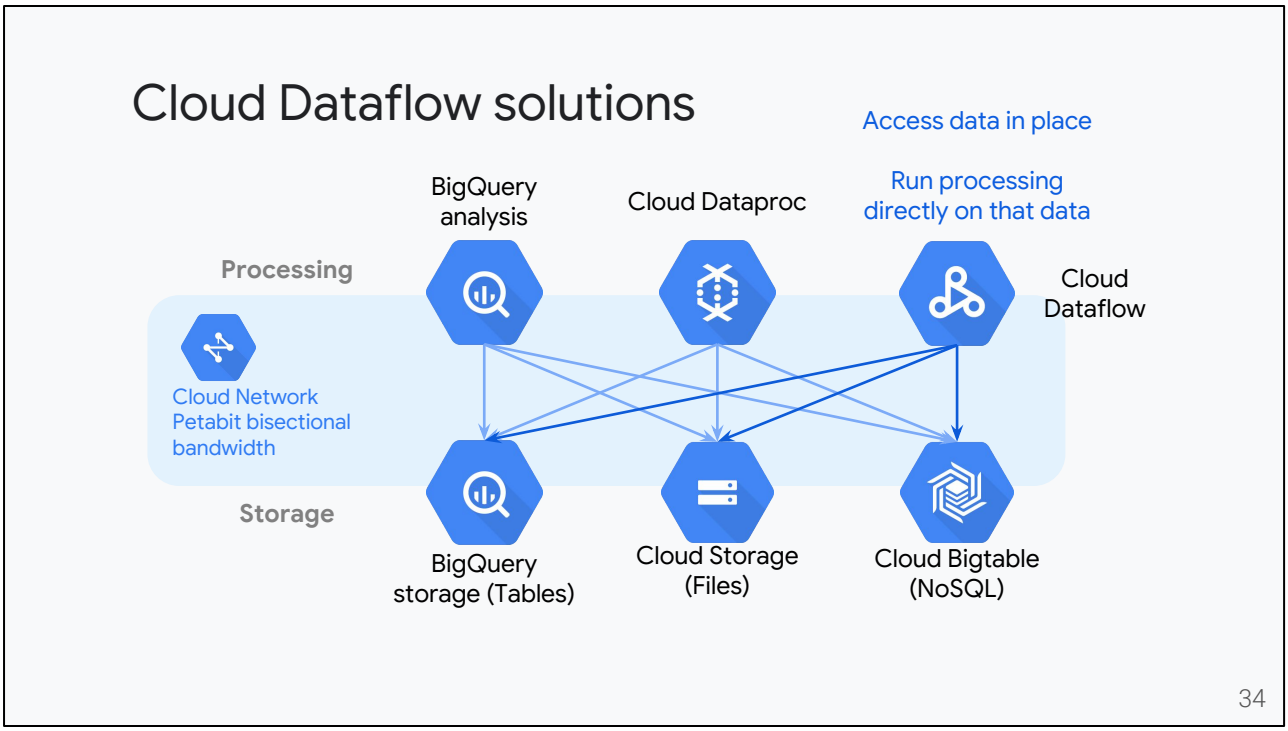
TIP: If you need more capabilities than a query engine, consider Cloud Dataproc or Cloud Dataflow.

Petabit bisectional bandwidth is what makes serverless services possible.

Cloud Dataproc solutions



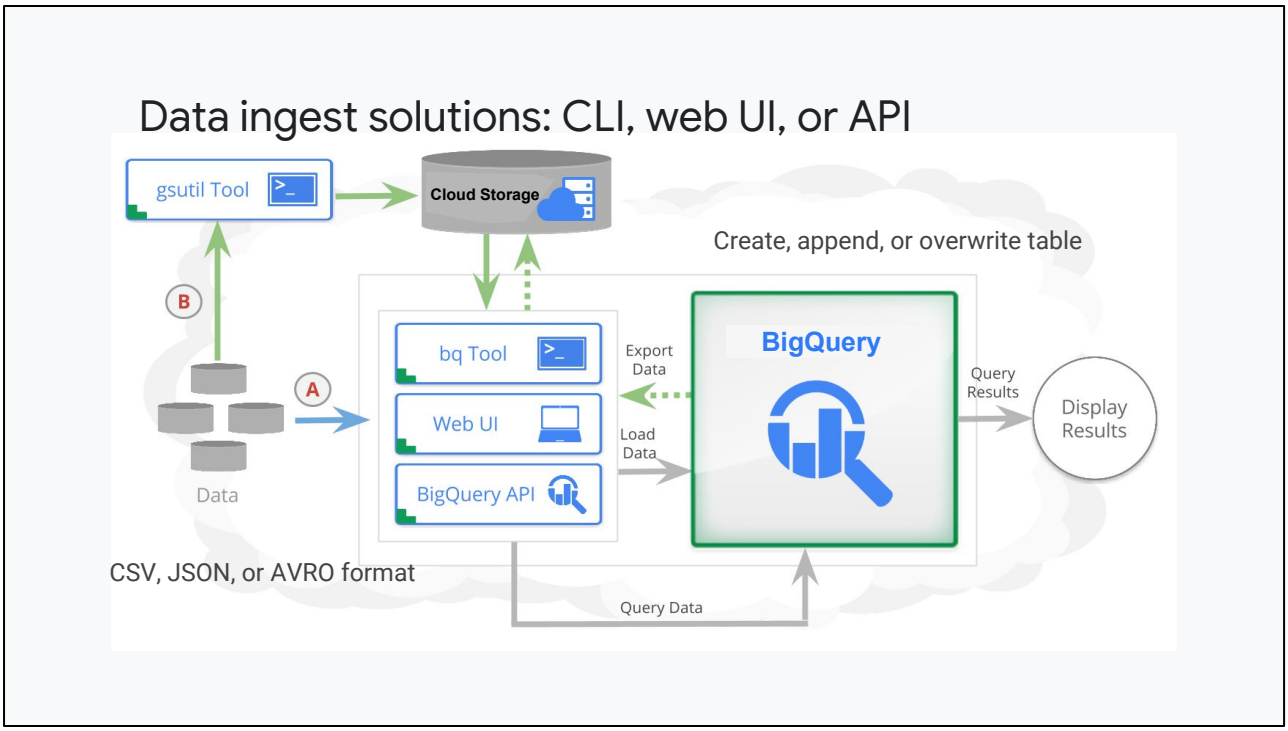
TIPS:
Cloud Dataproc can use Cloud Storage in place of HDFS for persistent data. If you use Cloud Storage, you can (A) shut down the cluster when it is not actually processing data, and (B) start up a cluster per job or per category or work, so you don't have to tune the cluster to encompass different kinds of jobs.
Cloud Bigtable is a drop-in replacement for HBASE, again, separating state from the cluster so the cluster can be shut down when not in use and started up to run a specific kind of job.



TIP: Cloud Dataproc and Cloud Dataflow can output separate files as CSV files in Cloud Storage. This is an easy way to accumulate distributed results for later collating.

Access any storage service from any data processing service.
Cloud Dataflow is an excellent ETL solution for BigQuery.
Use Cloud Dataflow to aggregate data in support of common queries.

Design data processing infrastructure



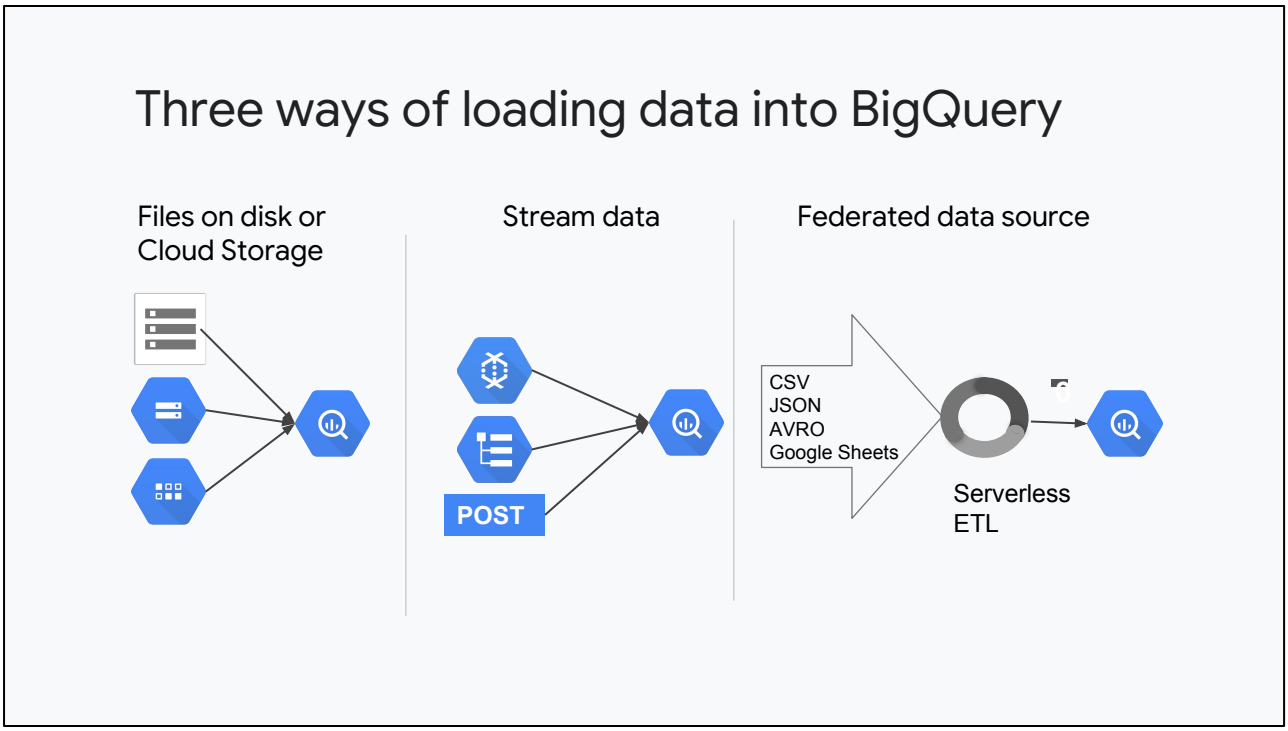
TIP: Manual ingest solutions.

BigQuery CLI:

- Good for uploading large data files, scheduling data file uploads
- Create table, define schema, and load data with one command
- Can also run queries from the command line
    - Interactively or batch mode queries
    - Automating scripts using scripting language
- Run the bq command-line tool from:
    - A Compute Engine instance
    - Cloud Shell
    - Client machine (requires installing the Google Cloud SDK)

Syntax for loading data via CLI:
bq load [--source_format=NEWLINE_DELIMITED_JSON|CSV] destination_table data_source_uri table_schema

More details: https://developers.google.com/bigquery/bq-command-line-tool



1. From the web GCP Console and files on disk, Cloud Storage or Cloud Datastore
2. Stream data with Cloud Dataflow, from Cloud Audit Logging, or with POST calls
3. Set up federated data source (serverless ETL!)
   for CSV, JSON, Avro on Cloud Storage or Google Sheets

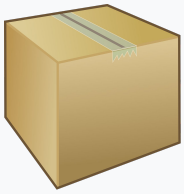
#1 is easy: just do it from the Google Cloud Platform Console. It can also be done programmatically of course.

#3 is super-convenient. BigQuery will auto-detect CSV and JSON format! See https://cloud.google.com/bigquery/federated-data-sources.

Options for transferring storage

gsutil

Storage Transfer Service



Transfer Appliance

TIP

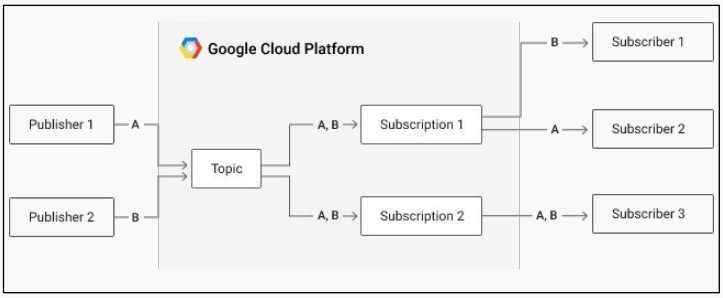
When transferring data from an on-premises location, use gsutil.
When transferring data from another cloud storage provider, use Storage Transfer Service.
Otherwise, evaluate both tools with respect to your specific scenario.

TIP: The three "Vs": Volume, Velocity, Variety. How much, how often, and how consistent?

Storage Transfer Service: https://cloud.google.com/storage-transfer/docs/overview

https://pixabay.com/en/flat-design-symbol-icon-www-2126883/
https://pixabay.com/en/box-cardboard-closed-carton-moving-40302/

Cloud Pub/Sub



Cloud Pub/Sub provides a NoOps, serverless global message queue.

Asynchronous; publisher never waits, a subscriber can get the message now or any time within 7 days.

At-least-once delivery guarantee.

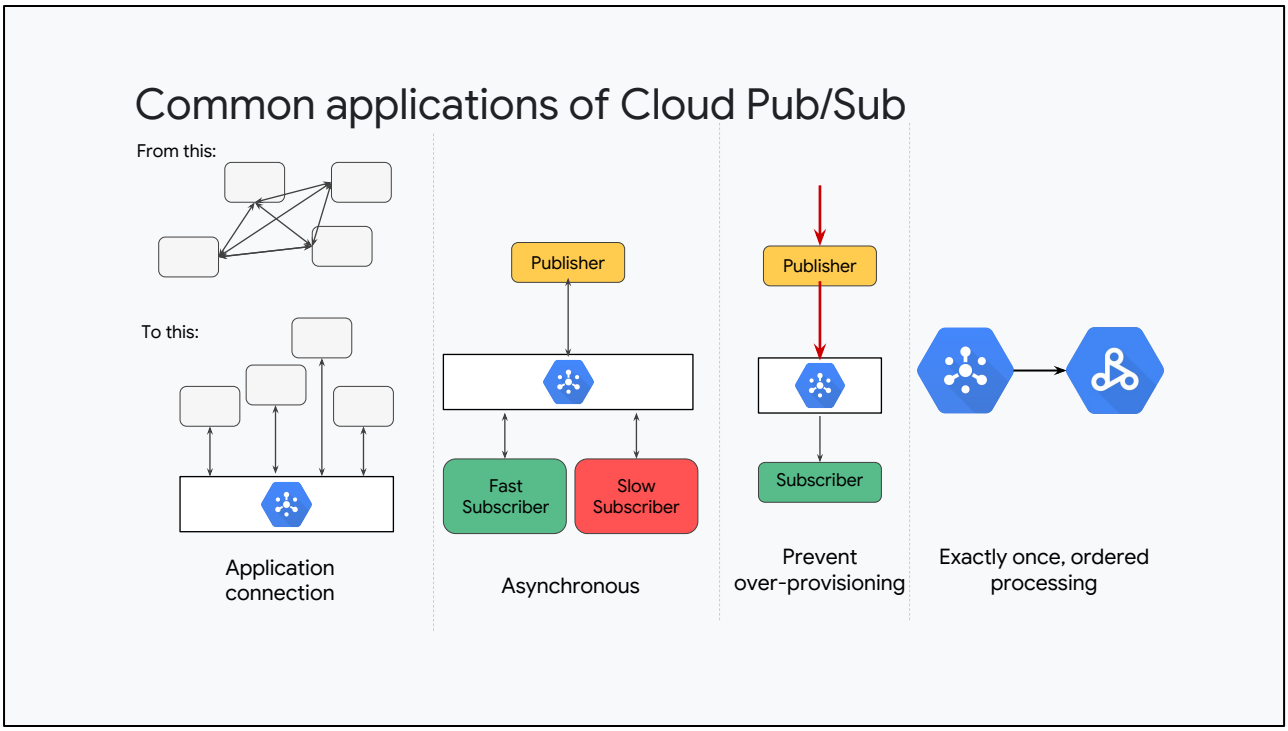
Push subscription and pull subscription delivery flows.

100s of milliseconds -- fast.

Message broker enables complete ingest solutions.

Loose coupling between systems

Long-lived connections between systems



Cloud Pub/Sub connects applications and services through a messaging infrastructure.

Pub/Sub simplifies event distribution by replacing synchronous point-to-point connections with a single, high-availability asynchronous bus.

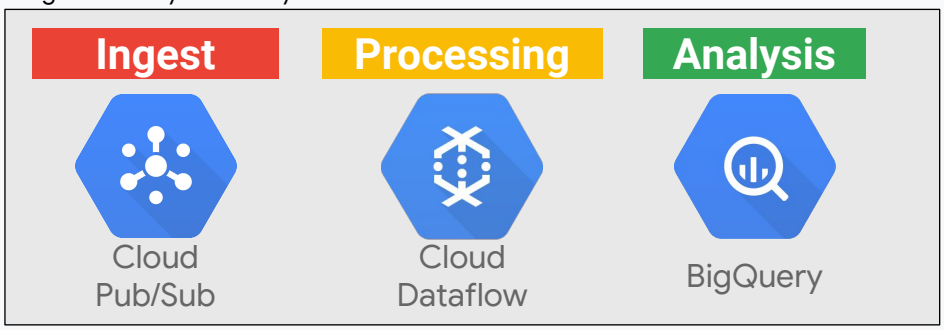
Can avoid over-provisioning for traffic spikes with Cloud Pub/Sub.

With Cloud Dataflow: exactly once, ordered processing

Cloud Pub/Sub with Cloud Dataflow: exactly once, ordered processing. Cloud Pub/Sub handles exactly once delivery and Cloud Dataflow handles deduplication, ordering, and windowing. Separation of duties enables a scalable solution that surpasses bottlenecks in competing messaging systems.

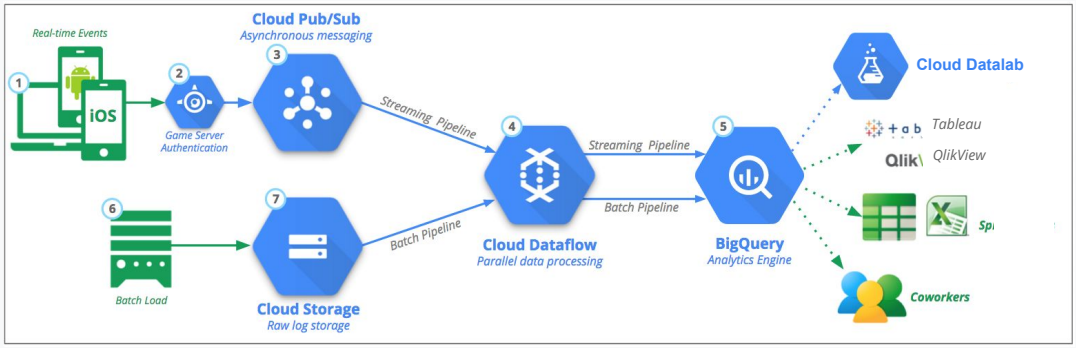
Serverless analytics solution

Stages of Analytics Lifecycle



Why Serverless?
No need to guess capacity.
No need to worry about idle resources.
Nothing to maintain.

Data analytics solution



Build a mobile gaming analytics platform: a reference architecture

Build a mobile gaming analytics platform: a reference architecture
https://cloud.google.com/solutions/mobile/mobile-gaming-analysis-telemetry

Popular mobile games can attract millions of players and generate terabytes of game-related data in a short burst of time. This places extraordinary pressure on the infrastructure powering these games and requires scalable data analytics services to provide timely, actionable insights in a cost-effective way.

To address these needs, a growing number of successful gaming companies use Google's web-scale analytics services to create personalized experiences for their players. They use telemetry and smart instrumentation to gain insight into how players engage with the game and to answer questions like: At what game level are players stuck? What virtual goods did they buy? And what's the best way to tailor the game to appeal to both casual and hardcore players?

Storage for CSV files. Analysts will run ANSI SQL queries. Support complex aggregate queries and reuse existing I/O-intensive custom Apache Spark transform. How should you transform the input data?

A. Use BigQuery for storage. Use Cloud Dataflow to run the transformations.
B. Use BigQuery for storage. Use Cloud Dataproc to run the transformations.
C. Use Cloud Storage for storage. Use Cloud Dataflow to run the transformations.
D. Use Cloud Storage for storage. Use Cloud Dataproc to run the transformations.

DEPE

Storage for CSV files. Analysts will run ANSI SQL queries. Support complex aggregate queries and reuse existing I/O-intensive custom Apache Spark transform. How should you transform the input data?

A. Use BigQuery for storage. Use Cloud Dataflow to run the transformations.
B. Use BigQuery for storage. Use Cloud Dataproc to run the transformations. ✓
C. Use Cloud Storage for storage. Use Cloud Dataflow to run the transformations.
D. Use Cloud Storage for storage. Use Cloud Dataproc to run the transformations.

DEPE

Solution

Use BigQuery for storage (for the analysts to run SQL queries) and use Cloud Dataproc to run the existing Spark transformations.

A is not correct because you should not use Cloud Dataflow for this scenario.

C and D are not correct because you should not use Cloud Storage for this scenario, and you should also not use Cloud Dataflow. Instead, you should just add secondary indexes.

https://stackoverflow.com/questions/46436794/what-is-the-difference-between-google-cloud-dataflow-and-google-cloud-dataproc

DEPE

Streaming log messages will be stored in a searchable repository. How should you set up the input messages? Requirements are final result message ordering, stream input for 5 days, and querying of the most recent message value.

A. Use Cloud Pub/Sub for input. Attach a timestamp to every message in the publisher.
B. Use Cloud Pub/Sub for input. Attach a unique identifier to every message in the publisher.
C. Use Apache Kafka on Compute Engine for input. Attach a timestamp to every message in the publisher.
D. Use Apache Kafka on Compute Engine for input. Attach a unique identifier to every message in the publisher.

DEPE

You are selecting a streaming service for log messages that must include final result message ordering as part of building a data pipeline on Google Cloud. You want to stream input for 5 days and be able to query the most recent message value. You will be storing the data in a searchable repository. How should you set up the input messages?

Streaming log messages will be stored in a searchable repository. How should you set up the input messages? Requirements are final result message ordering, stream input for 5 days, and querying of the most recent message value.

A. Use Cloud Pub/Sub for input. Attach a timestamp to every message in the publisher. ✓
B. Use Cloud Pub/Sub for input. Attach a unique identifier to every message in the publisher.
C. Use Apache Kafka on Compute Engine for input. Attach a timestamp to every message in the publisher.
D. Use Apache Kafka on Compute Engine for input. Attach a unique identifier to every message in the publisher.

DEPE

Solution

A is correct because of recommended Google practices

B is not correct because you should not attach a GUID to each message to support the scenario.

C and D are not correct because you should not use Apache Kafka for this scenario (it is overly complex compared to using Cloud Pub/Sub, which can support all of the requirements).

https://cloud.google.com/pubsub/docs/ordering

DEPE

Designing Data Processing Systems

Exam Guide Review

Storage

Selecting the appropriate storage technologies.	Mapping storage systems to business requirements Data modeling Tradeoffs involving latency, throughput, and transactions Distributed systems Schema design

Tip: Be familiar with the common use cases and qualities of the different storage options. Each storage system or database is optimized for different things -- some are best at atomically updating the data for transactions. Some are optimized for speed of data retrieval but not for updates or changes. Some are very fast and inexpensive for simple retrieval but slow for complex queries.

Pipelines

Designing data pipelines.	Data publishing and visualization Batch and streaming Online (interactive) vs. batch predictions Job automation and orchestration

Tip: An important element in designing the data processing pipeline starts with selecting the appropriate service or collection of services.

Tip: Cloud Datalab, Cloud Data Studio, BigQuery all have interactive interfaces. Do you know when to use each?

Processing Infrastructure

Designing data Designing a data processing solution.	Choice of infrastructure System availability and fault tolerance Use of distributed systems Capacity planning Hybrid cloud and edge computing Architecture options At least once, in-order, and exactly once event planning

Tip: Cloud Pub/Sub and Cloud Dataflow together provide once, in-order, processing of possibly delayed or repeated streaming data.

Tip: Be familiar with the common assemblies of services and how they are often used together., Cloud Dataflow, Cloud Dataproc, BigQuery, Cloud Storage, and Cloud Pub/Sub.

Migration

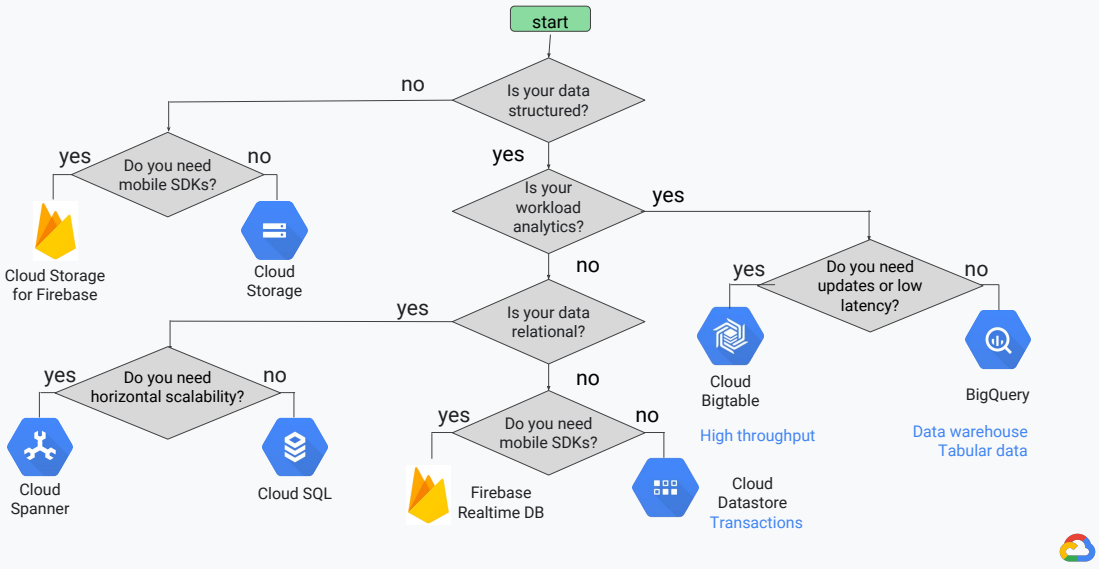
Migrating data warehousing and data processing.	Awareness of current state and how to migrate design to a future state. Migrating from on-premise to cloud Validating a migration

Tip: Technologically, Cloud Dataproc is superior to Open Source Hadoop, and Cloud Dataflow is superior to Cloud Dataproc. However, this does not mean that the most advanced technology is always the best solution. You need to consider the business requirements. The client might want to first migrate from the data center to the cloud. Make sure everything is working (validate it). And only after they are confident with that solution, to consider improving or modernizing.

Building data processing systems

Building and maintaining data structures and databases

Selecting storage options



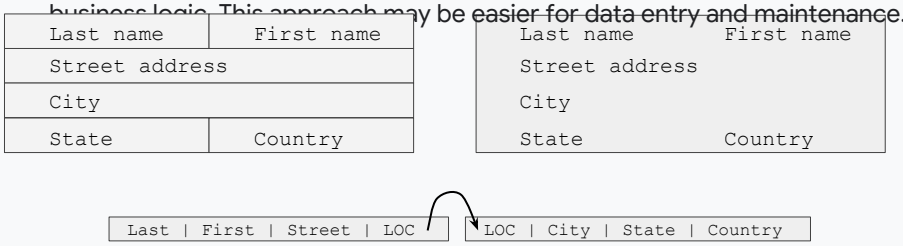
TIP:
BigQuery is recommended as a data warehouse.
BigQuery is the default storage for tabular data.
Use Cloud Bigtable if you need transactions.
Use Cloud Bigtable if you want low latency/high throughput.

Everything we will cover next is summed up in the following slides.

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.



In the example on the left, each data item is stored separately, making it easy to filter on a specific field 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?

CAP Theorem

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

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

64

TIP: 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.

TIP: Did you know that in some cases an eventually consistent solution can be made strongly consistent for a specific limited use case?
https://cloud.google.com/datastore/docs/articles/balancing-strong-and-eventual-consistency-with-google-cloud-datastore/
In Cloud 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.

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

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

https://en.wikipedia.org/wiki/Consistency_(database_systems)
https://en.wikipedia.org/wiki/ACID
Atomicity: A transaction is either "all or nothing"
Consistency: Any transaction brings the database from one valid state to another.
Isolation: Transactions executed concurrently produce the same result as if executed sequentially.

Durability: When a transaction is committed, the results are stable. Even in the event of a power loss, the result is non-volatile.

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

https://pixabay.com/en/menu-yemekservisi-akula-1197654/
https://en.wikipedia.org/wiki/CAP_theorem

In database systems, ACID (most SQL systems) optimize for consistency and BASE (most NoSQL systems) optimize for availability.

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

Access control: https://cloud.google.com/storage/docs/access-control/making-data-public
Streaming: https://cloud.google.com/storage/docs/streaming
https://cloud.google.com/storage/docs/changing-storage-classes

https://cloud.google.com/storage/docs/best-practices -- Traffic Estimation

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 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

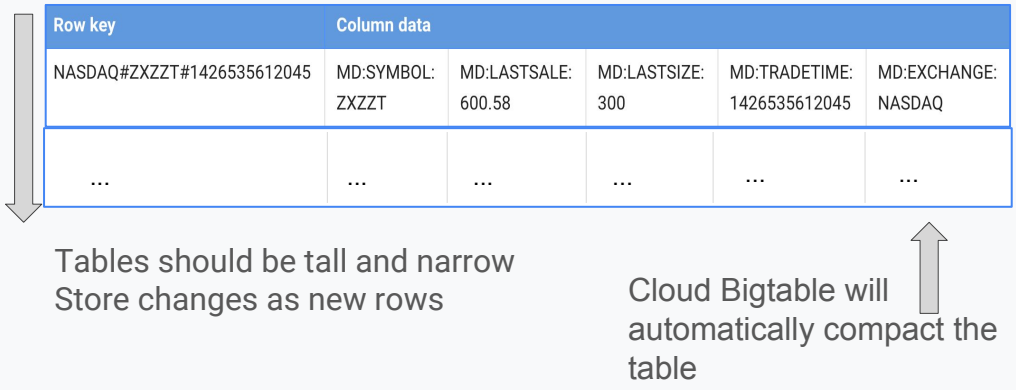
Access control

Performance design

Choosing between SSD and HDD

Schema design for Cloud Bigtable:
https://cloud.google.com/bigtable/docs/schema-design
Cloud Bigtable Schema Design for Time Series Data:
https://cloud.google.com/bigtable/docs/schema-design-time-series
Access Control: https://cloud.google.com/bigtable/docs/access-control
Understanding Cloud Bigtable Performance:
https://cloud.google.com/bigtable/docs/performance
Choosing Between SSD and HDD Storage:
https://cloud.google.com/bigtable/docs/choosing-ssd-hdd

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



https://cloud.google.com/bigtable/docs/schema-design-time-series

- Tall and narrow: each trade is its own row. This will result in 100s of millions of rows per day. This is fine.
- Autobalanced: tables are broken up into “tablets” based on row-key. Data is stored in a durable way with redundancy. As entries aredeleted, Cloud Bigtable automatically compacts the table to reduce overall storage.

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

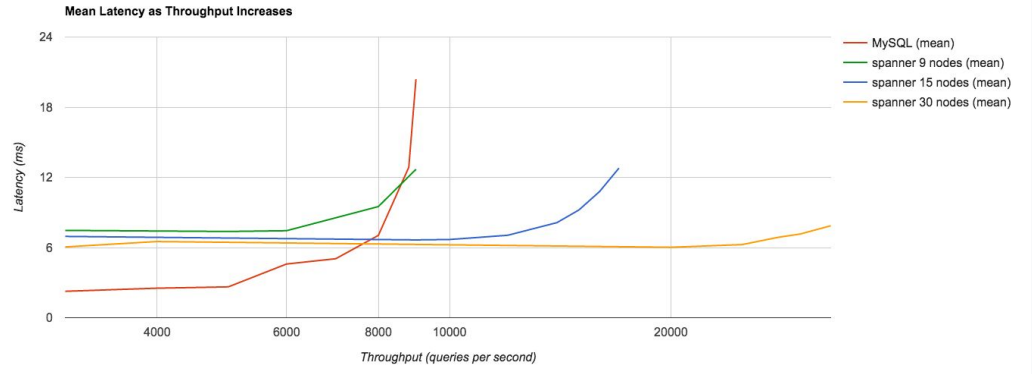
Data types

Transactions

Timestamp bounds and Commit timestamps

Schema Design: https://cloud.google.com/spanner/docs/schema-design
Data Model: https://cloud.google.com/spanner/docs/schema-and-data-model
Schema updates: https://cloud.google.com/spanner/docs/schema-updates
Secondary indexes: https://cloud.google.com/spanner/docs/secondary-indexes
Data types: https://cloud.google.com/spanner/docs/data-types
Transactions: https://cloud.google.com/spanner/docs/transactions
Timestamp bounds: https://cloud.google.com/spanner/docs/timestamp-bounds
Timestamp commits: https://cloud.google.com/spanner/docs/commit-timestamp

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



Source:
https://quizlet.com/blog/quizlet-cloud-spanner

Cloud SQL is fine if you can only need a single database. But if you need multiple databases, Cloud Spanner is a great choice.
The graphs above (by Quizlet) illustrate this. MySQL hits a wall at around 8000 queries/second. If you look at the 99th percentile of latency, it is clear that performance degrades beyond 5000. Distributing MySQL is hard. However, Cloud Spanner distributes easily (even globally) and provides consistent performance. To support more throughput, just add more nodes.

Cloud 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

Identity and Access Management:
https://cloud.google.com/datastore/docs/access/iam
Storage size calculations:
https://cloud.google.com/datastore/docs/concepts/storage-size
Multitenancy: https://cloud.google.com/datastore/docs/concepts/multitenancy
Encryption: https://cloud.google.com/datastore/docs/concepts/encryption-at-rest

Comparing storage options: Use cases

	Cloud 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

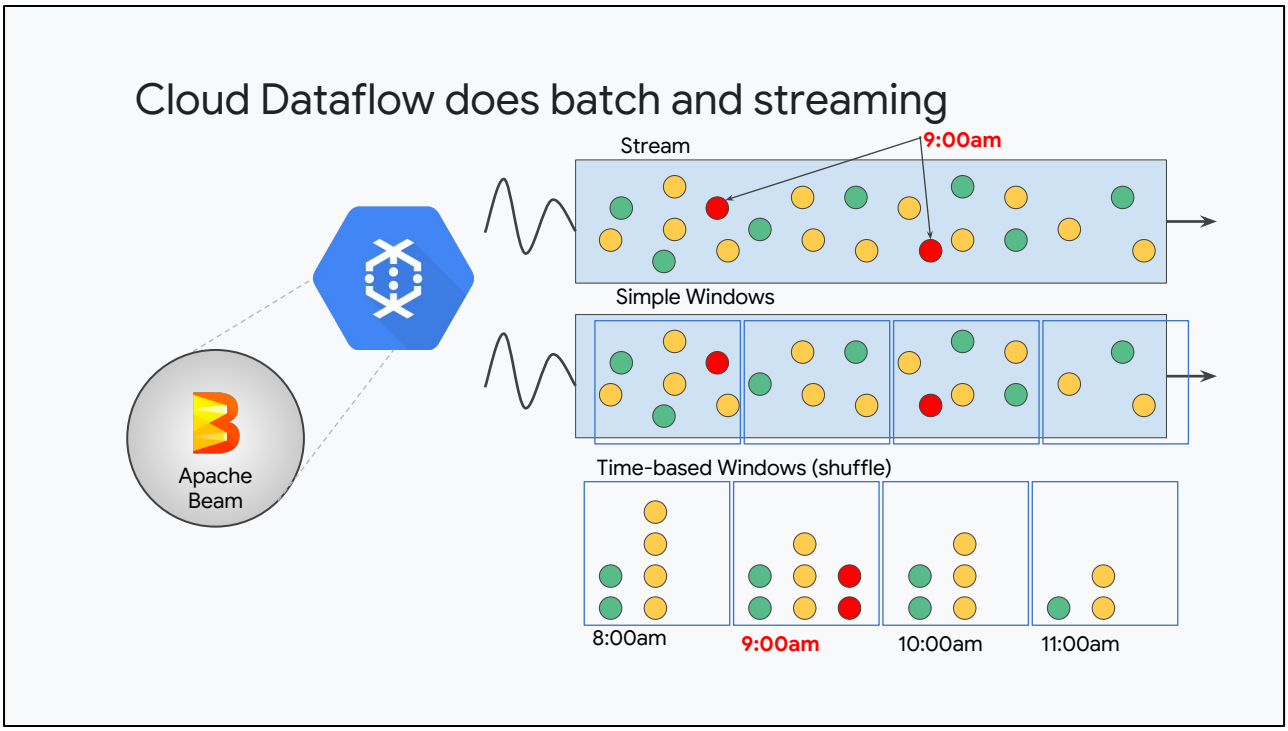
TIP: Commit this table to memory, and be able to use it backwards. Example: If the exam question contains "Data Warehouse," you should be thinking "BigQuery is a candidate."

Google Cloud Platform delivers various storage service offerings which remove much of the burden of building and managing storage and infrastructure. Like Google's other cloud services, storage services free you to focus on doing what you do best and differentiating at the application or service layer.

Google's storage offerings range across the spectrum. You can use different types of storage in the same project.

- Cloud SQL gives you fully managed MySQL so you have relational DB and a more traditional approach to queries.
- Cloud Datastore provides a nearly infinitely scalable, schemaless solution.
- If you want a disk, you can mount Persistent Disk as a block store that can be used by Compute Engine.
- For just pure data and blobs, Cloud Storage can deliver what you need.
- Cloud Bigtable offers companies a fast, fully managed, infinitely scalable NoSQL database service ideal for web, mobile, and IoT applications.

Building and maintaining pipelines

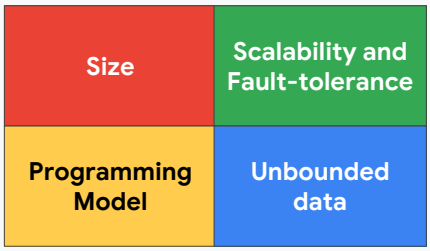


Apache Beam is an open programming platform for unifying batch and streaming. Before Apache Beam, you needed two pipelines to balance latency, throughput, and fault tolerance.
Cloud Dataflow is Apache Beam as a service; a fully managed autoscaling service that runs Beam pipelines.

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

Cloud Dataflow solves many stream processing issues

Autoscaling and rebalancing handles variable volumes of data and growth



On-demand and distribution of processing scales with fault tolerance

Efficient pipelines (Apache Beam) and 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

Size: Data can change (spike) and grow over time.
Scalability: Service must adapt and grow, yet remain fault-tolerant.
Programming model: Compare traffic today with traffic last Friday; is this stream or batch?
Unbounded data: Unbounded data could arrive late or out of order.

Cloud 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?

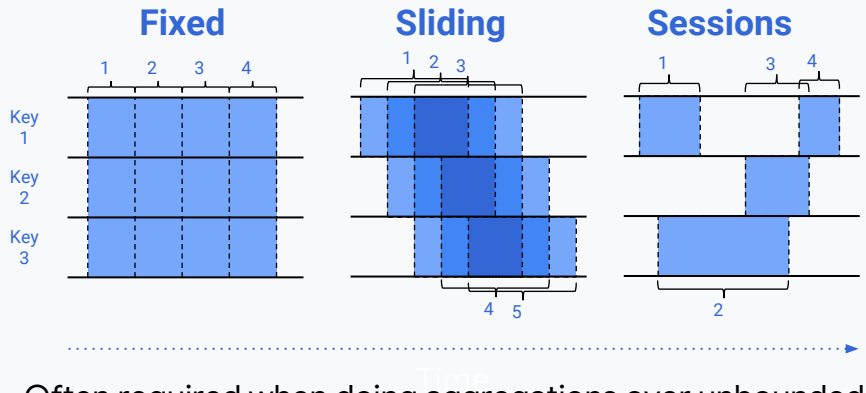
Fixed, **sliding**, and **session-based** windows.

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.

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

Windowing divides data into event time–based finite chunks.



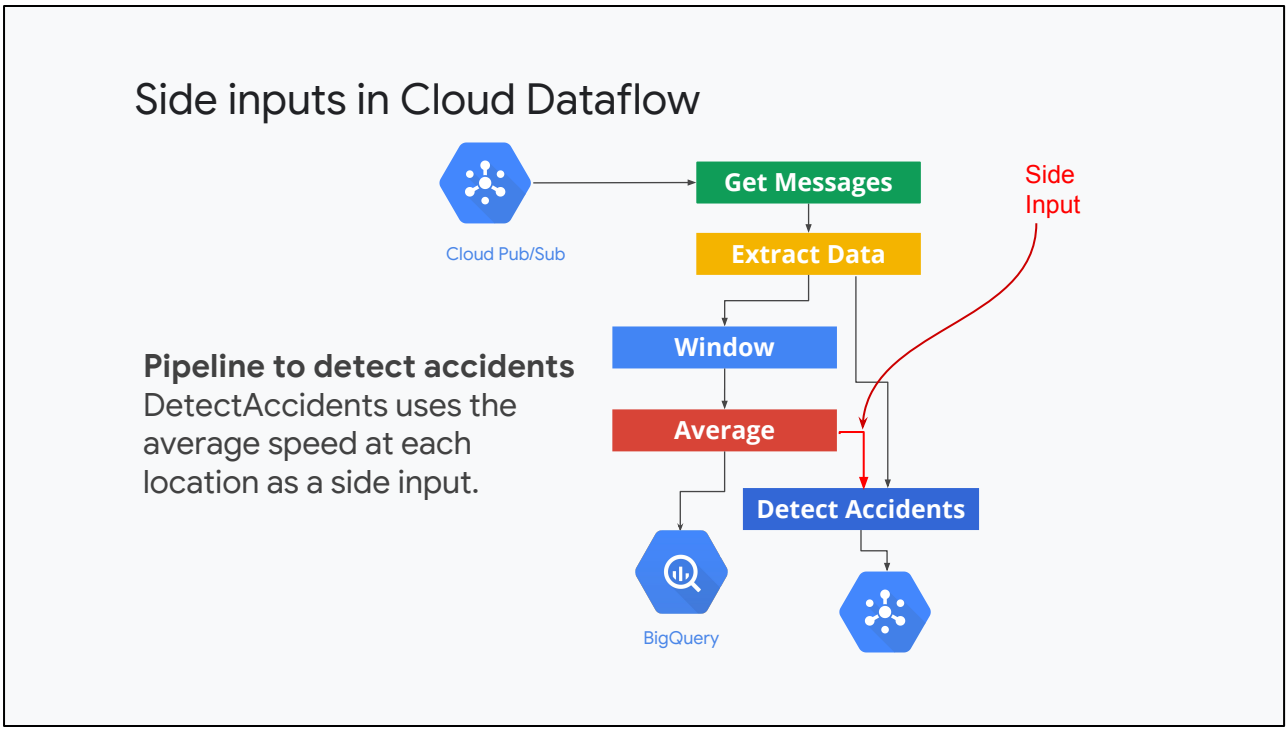
Often required when doing aggregations over unbounded data.

Windowing answers the question by creating individual results for different slices of event time.

Windowing divides a PCollection into finite chunks based on the event time of each element. It can be useful for computations over both bounded and unbounded PCollections, but it's required when you are trying to create aggregations on infinite data.

There are many different ways to implement windows on data, but some of the most common methods include fixed time (for example; hourly, daily, monthly), overlapping sliding windows (for example; the last 24 hours worth of data, every hour), and session-based windows that capture bursts of user activity.

Based on original slides by Tyler Akidau & Frances Perry, April 2016



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

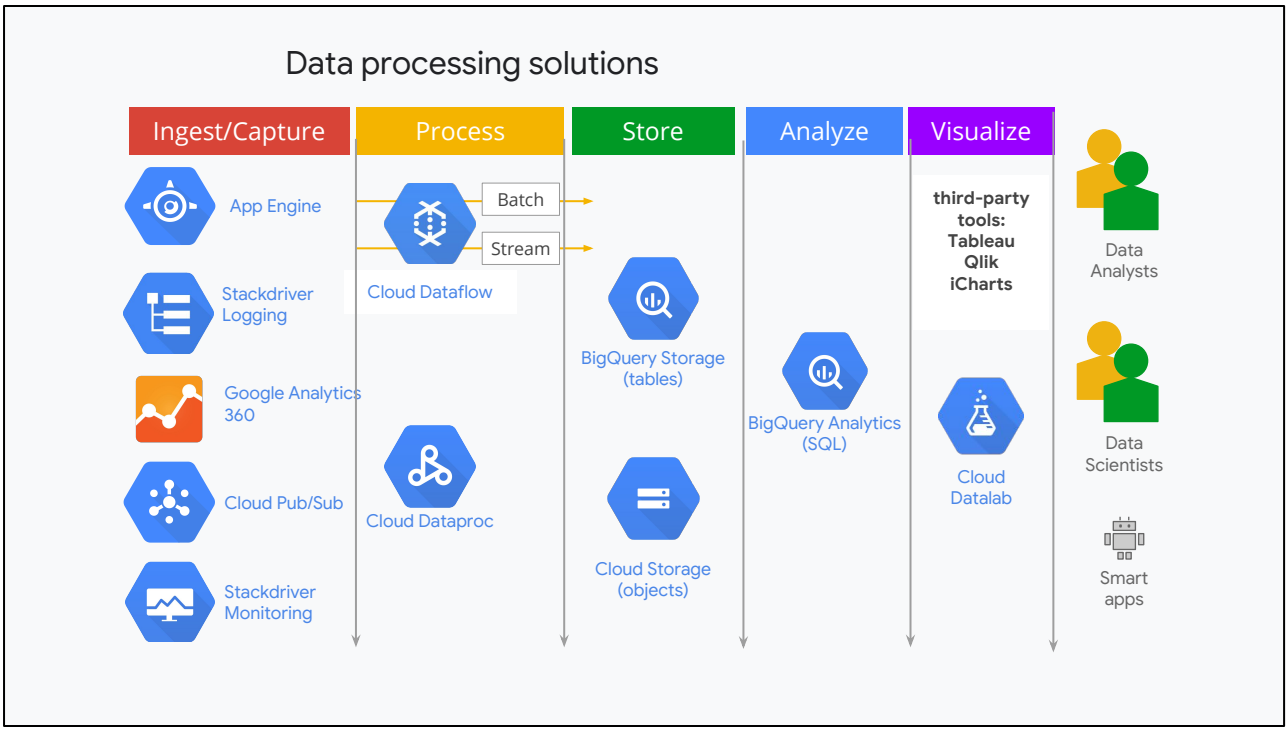
Building and maintaining processing infrastructure

Building a streaming pipeline

Stream from Cloud 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.

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

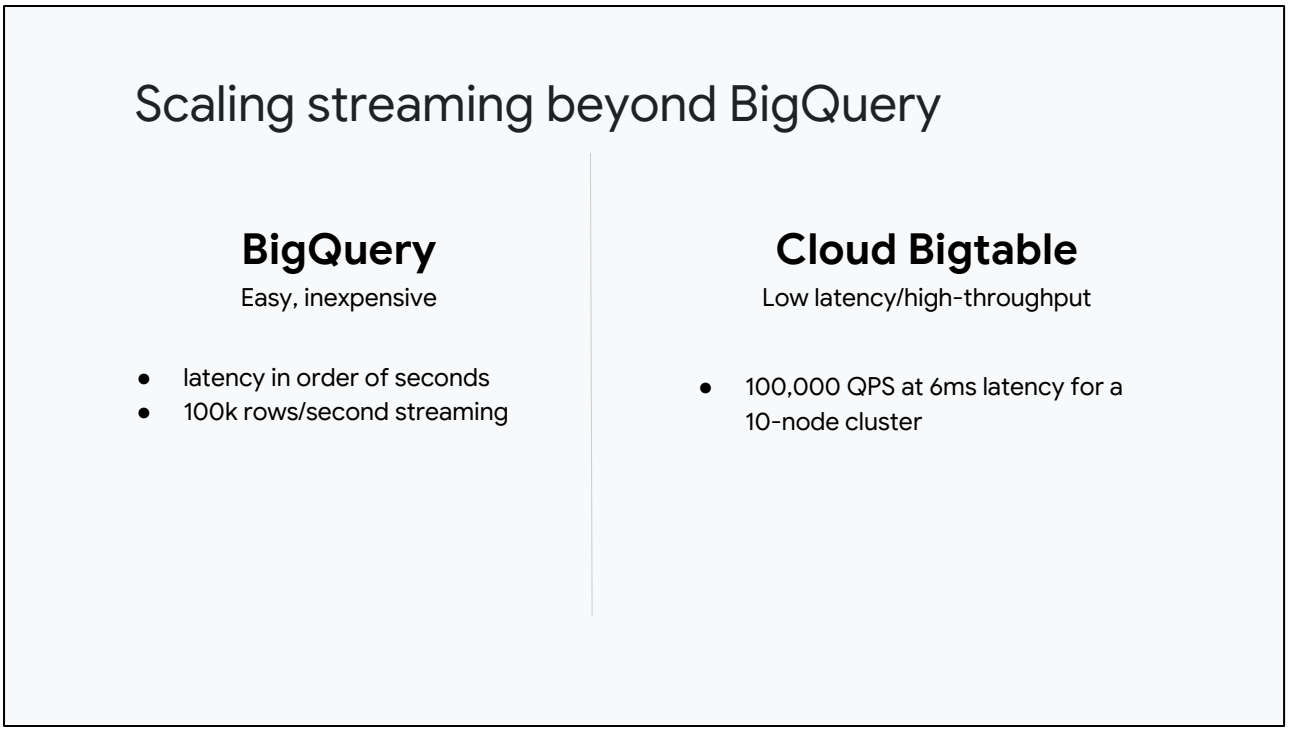


TIP: BigQuery is an inexpensive data store for tabular data. It is cost-comparable with Cloud Storage, so it makes sense to ingest into BigQuery and leave the data there.

Ingesting the data depends on where it is coming from. Cloud logs, GAP can directly ingested into BigQuery. From Cloud Pub/Sub, you have an API. In the most general case, you can use Cloud Dataflow and write code to ingest the data in batch/stream. You could also use OSS tools like Spark or Hadoop to do the processing, in which case you'd use Cloud Dataproc.

Analysis itself is done by BigQuery. The results can be visualized in a iPython notebook (Cloud Datalab) or in third-party tools.

So, BigQuery's role is in both storage and analysis. In other words, it is a data warehousing solution.



Why use Cloud Bigtable and not Cloud Spanner? Cost! Note that we can support 100,000 qps with 10 nodes in Cloud Bigtable, but would need ~150 nodes in Cloud Spanner.

Blog post to show read/write performance and write throughput: https://cloudplatform.googleblog.com/2015/05/introducing-Google-Cloud-Bigtable.html

In general, a cluster's performance increases linearly as you add nodes to the cluster. For example, if you create an SSD cluster with 10 nodes, the cluster can support up to 100,000 QPS for a typical workload, with 6 ms latency for each read and write operation.

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?

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. ✓

Solution

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.

https://cloud.google.com/sql/docs/mysql/replication/tips#read-replica

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.

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. ✓

Solution

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.

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

You 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.

DEPE

You 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.

DEPE

Solution

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.

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

DEPE

Building and Operationalizing Data Processing Systems

Exam Guide Review

Storage

Building and operationalizing storage systems.	Effective use of managed services. Storage costs and performance Data lifecycle management

Tip: Data management is often influenced by business requirements. After the data has been used for the "live" application, is it collected for reporting, for backup and recovery, for audits, or for legal compliance? What are the changing business purposes of the data in different time frames?

Tip: "Effective use of managed services" … choose the right service and the correct settings/features for specific use cases.

Consider costs, performance, and effective use cases (key features) for these:

- Cloud Bigtable
- Cloud Spanner
- Cloud SQL
- BigQuery
- Cloud Storage
- Cloud Datastore
- Cloud Memorystore

Pipelines

Building and operationalizing pipelines.	Data cleansing Batch and streaming Transformation Data acquisition and import Integrating with new data sources

Tip: What is Data cleansing? Data cleansing is improving the data quality through consistency. You could use Cloud Dataprep to Extract, Transform, or Load (ETL). You could run a data transformation job on Cloud Dataproc.

Tip: Batch and streaming together? You should already be thinking "Cloud Dataflow".

Tip: Integrating with new data sources. You should be familiar with the connectors available between services in the cloud, and common import/acquisition configurations.

Processing Infrastructure

Building and operationalizing processing infrastructure.	Provisioning resources Monitoring pipelines Adjusting pipelines Testing and quality control

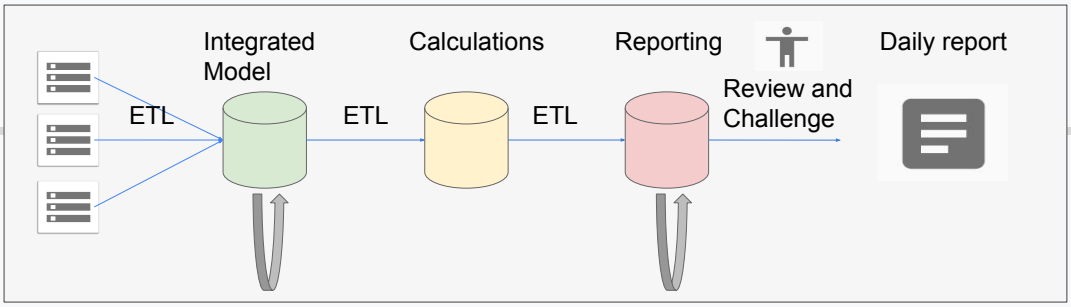
Tip: Testing and quality control and monitoring. You should be familiar with the common approaches to testing that are used in production environments, such as A/B testing, and other rollout scenarios. Likewise, there are operational and administrative monitoring elements for most services in the Cloud Console, and statistical and log monitoring in Stackdriver. Do you know how to enable and use Stackdriver with common services?

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



We mapped that to technical requirements like this...

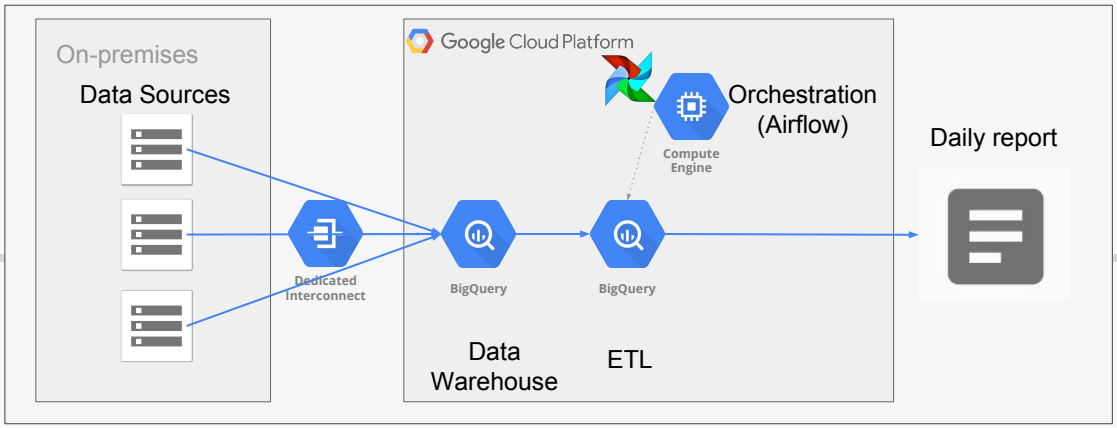
BigQuery and Cloud Composer (aka Apache Airflow)

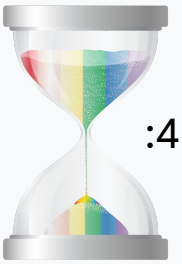
BigQuery: Reduce overall time to run with BQ as data warehouse and analytics engine

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

And this is how we implemented that technical requirement.

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





Challenge Lab 01

PDE Prep—BigQuery Essentials: Challenge Lab

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.

https://pixabay.com/en/hourglass-timer-rainbow-1895102/


Google Cloud Training and Google Cloud