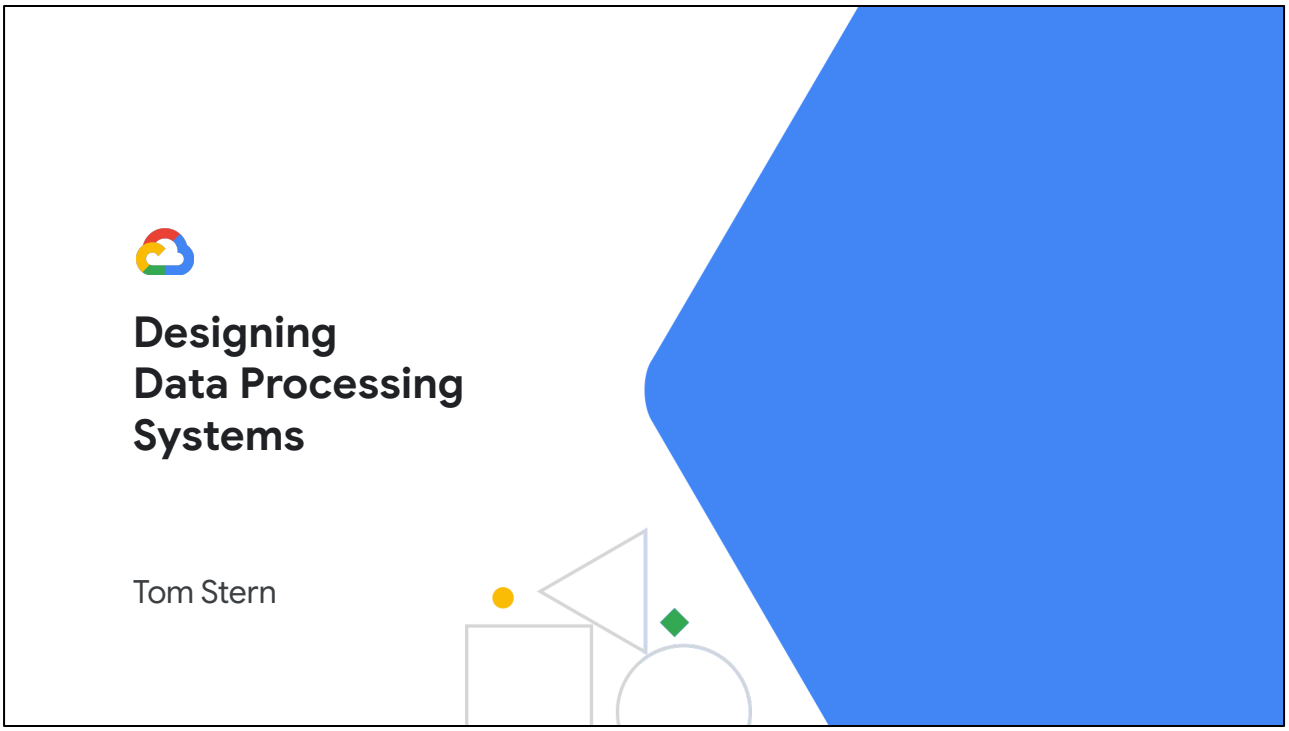
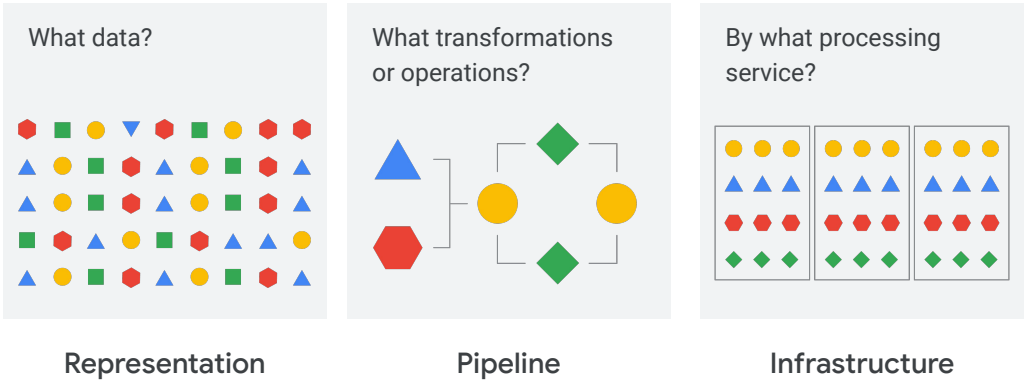


Designing Data Processing systems includes designing flexible data representations, designing data pipelines, and designing data processing infrastructure.

Data processing anatomy



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

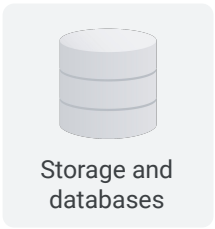
The same questions or interests show up in different contexts. Data representation, pipelines, processing infrastructure. For example, innovations in the technology could make the data representation of a chosen solution outdated. The data processing pipeline might have implemented a very involved transformation 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 not Data Representation. And "Capacity" is important to processing, but not the abstract pipeline or the representation.

Think about Data Engineering in Google Cloud as a platform consisting of components that can be assembled into solutions.

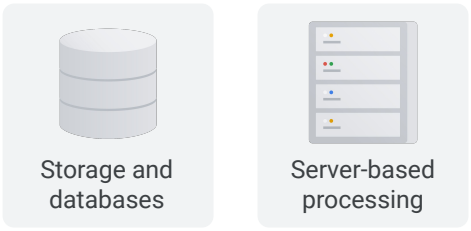
Let's review the elements of Google Cloud that form the Data Engineering platform.

A view of data engineering on Google Cloud



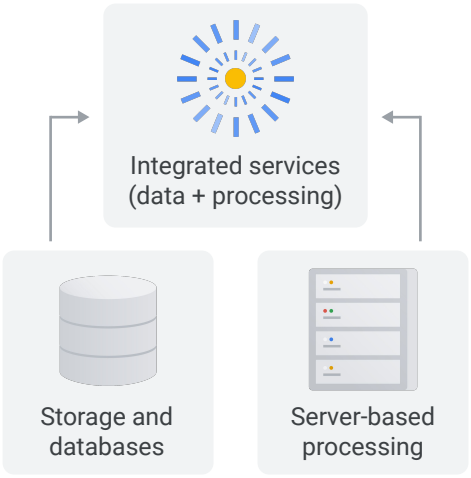
Storage and databases - Services that enable storing and retrieving data; differ in storage and retrieval methods that make them more efficient for specific use cases.

A view of data engineering on Google Cloud



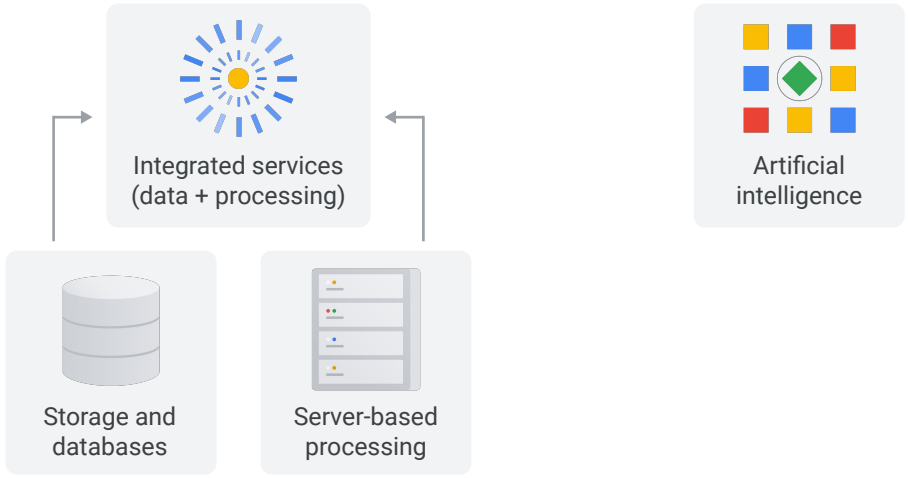
Server-based processing - Services that enable application code and software to run that can make use of stored data to perform operations -- actions and transformations -- producing results.

A view of data engineering on Google Cloud

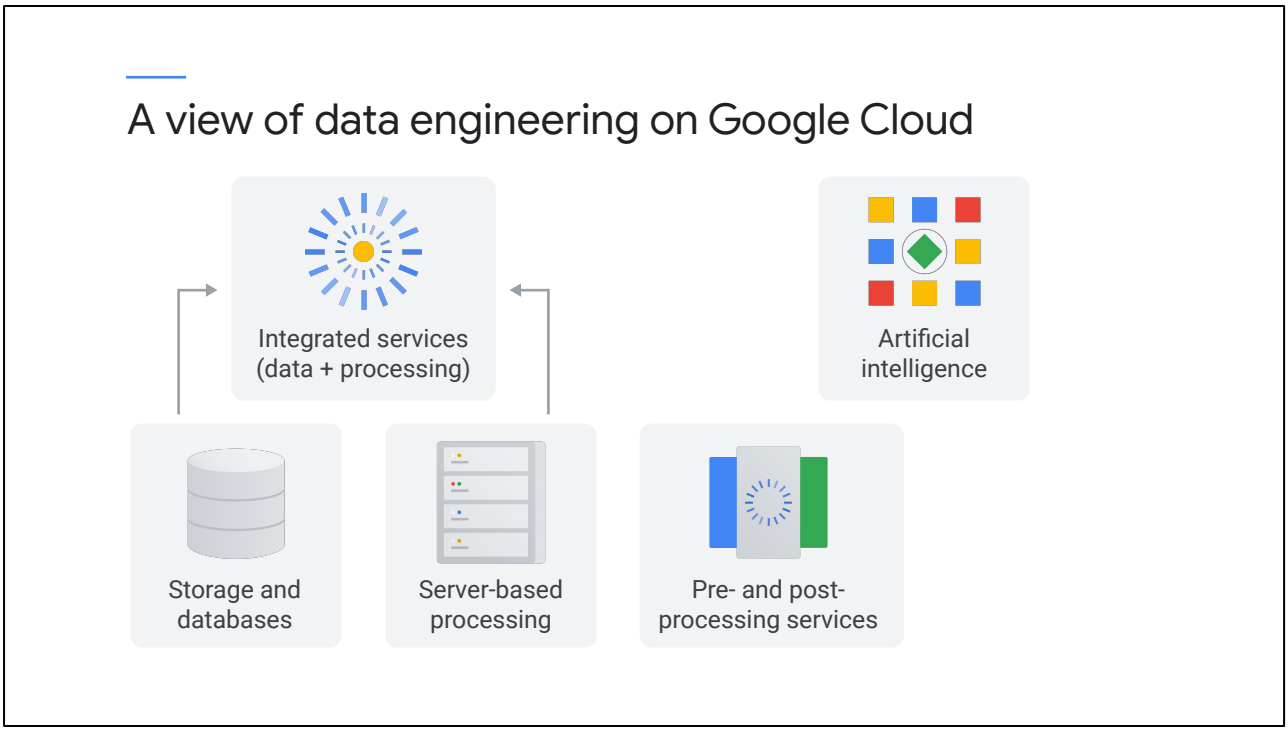


Integrated services -- combines storage and scalable processing in a framework design to process data (rather than general applications). More efficient and flexible than isolated server/database solutions.

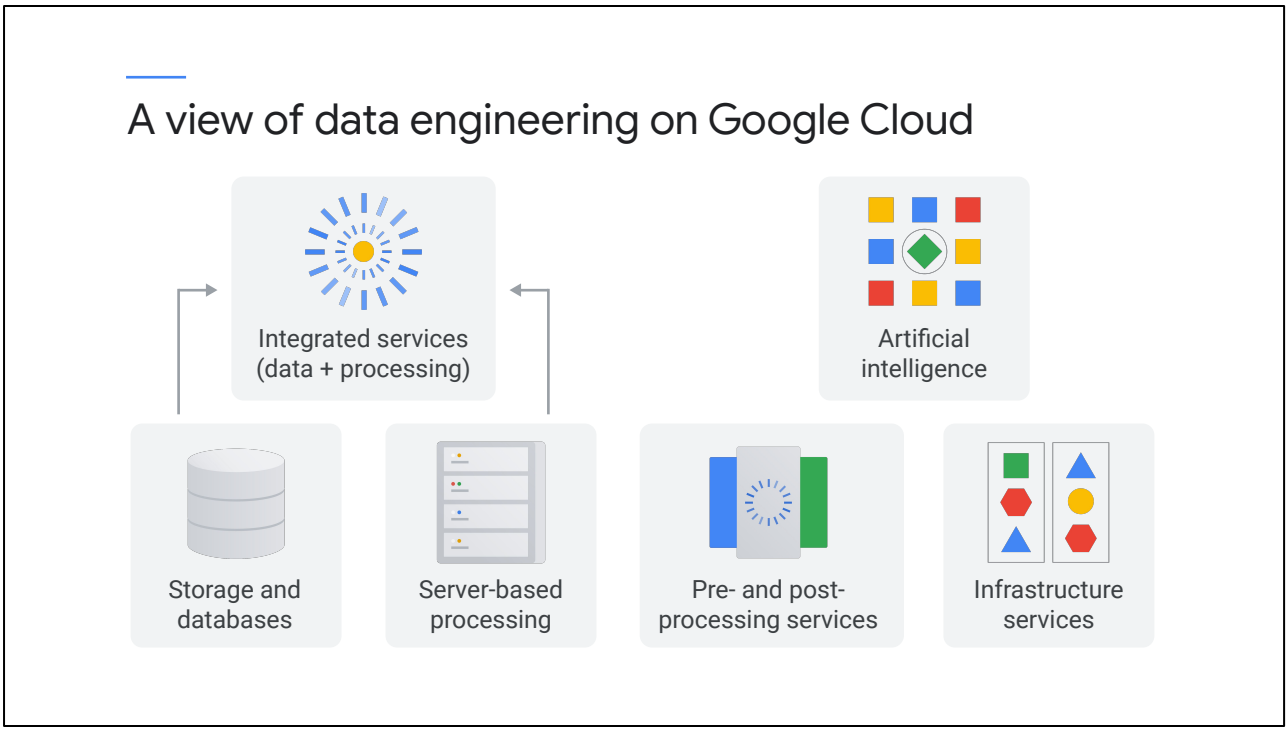
A view of data engineering on Google Cloud



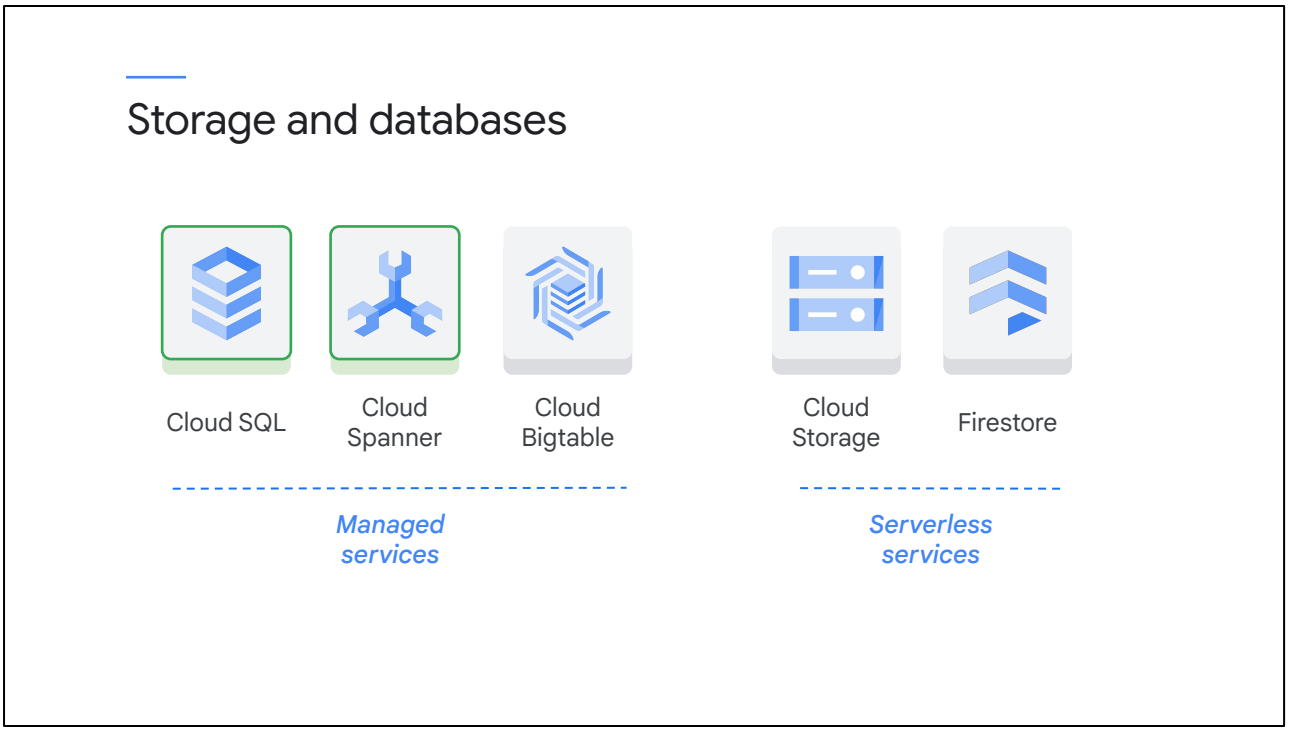
Artificial intelligence - MethodS to help identify (tag), categorize, and predict; three actions that are very hard or impossible to accomplish in data processing without Machine Learning.



Pre-and-post processing 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, and so forth.



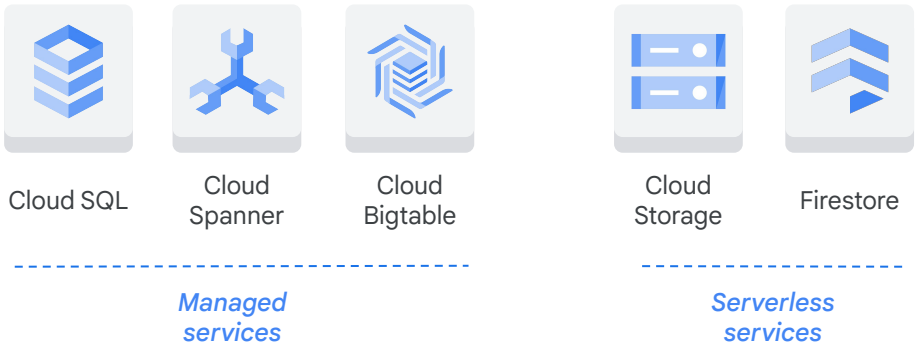
Storage and database systems are designed and optimized for storing and retrieving. They are not really built to do data transformation.
It is assumed in their design that the computing power necessary to perform transformations on the data is external to the storage or database

The organization method and access method of each of these services is efficient for specific cases. For example, a Cloud SQL database is very good at storing consistent individual transactions. But it is not really optimized for storing large amounts of unstructured data like video files.

Database services perform minimal operations on the data within the context of the access method. For example SQL queries can aggregate, accumulate, count, and summarize results of a search query.

Here is an exam tip: Know the differences between Cloud SQL and Cloud Spanner, and when to use each.

Storage and databases

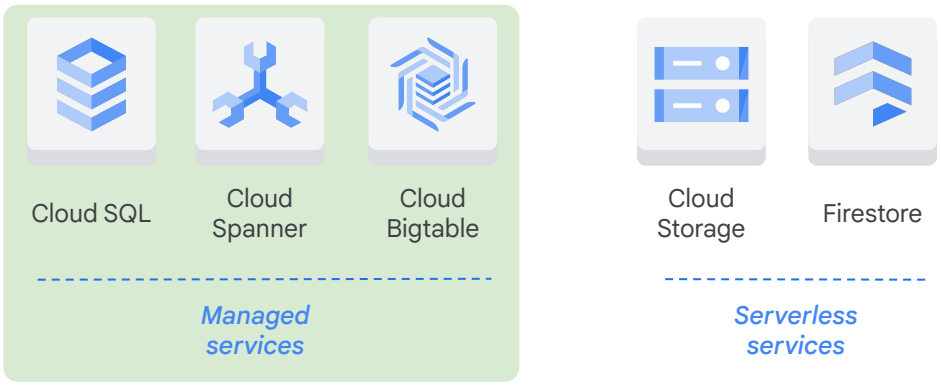


Service differentiators include access methods, the cost or speed of specific actions, size of data, and how data is organized and stored.

Details and differences between the data technologies are discussed later in this course.

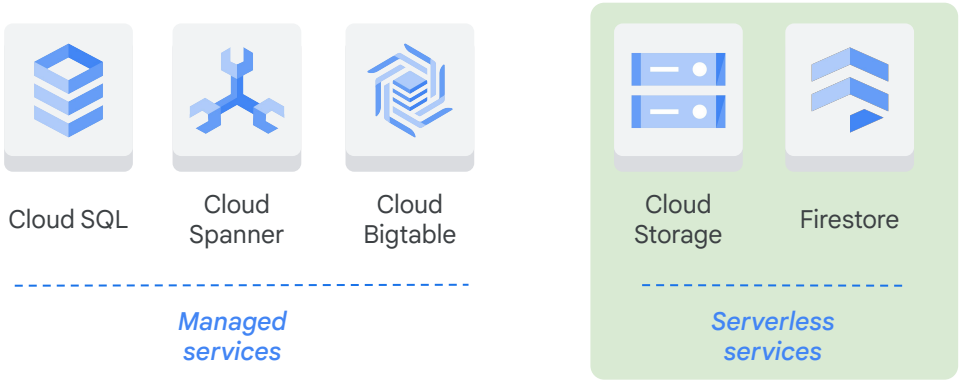
Exam tip: know how to identify technologies backwards from their properties. For example, which data technology offers the fastest ingest of data?
Which one might you use for ingest of streaming data?

Storage and databases

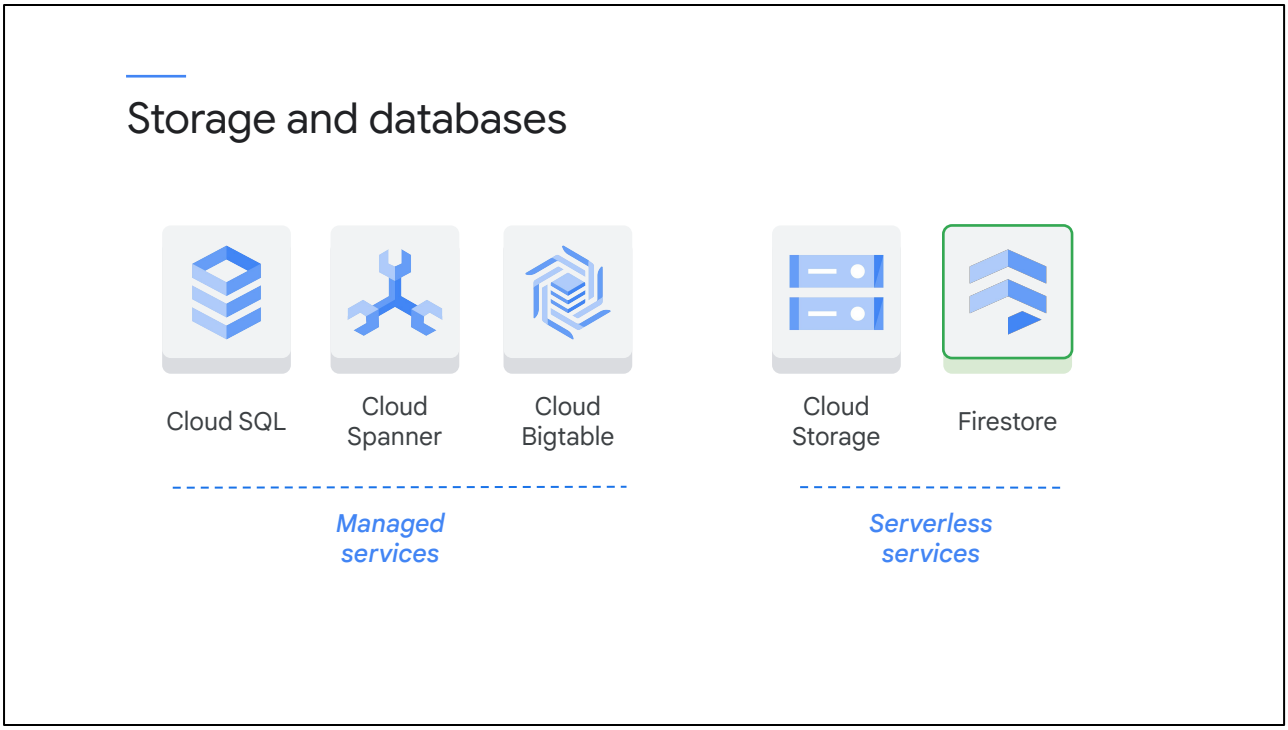


Managed Services are ones where you can see the individual instance or cluster. Exam tip -- managed services still have some IT overhead. It doesn't completely eliminate the overhead or manual procedures but it minimizes them compared with on prem solutions.

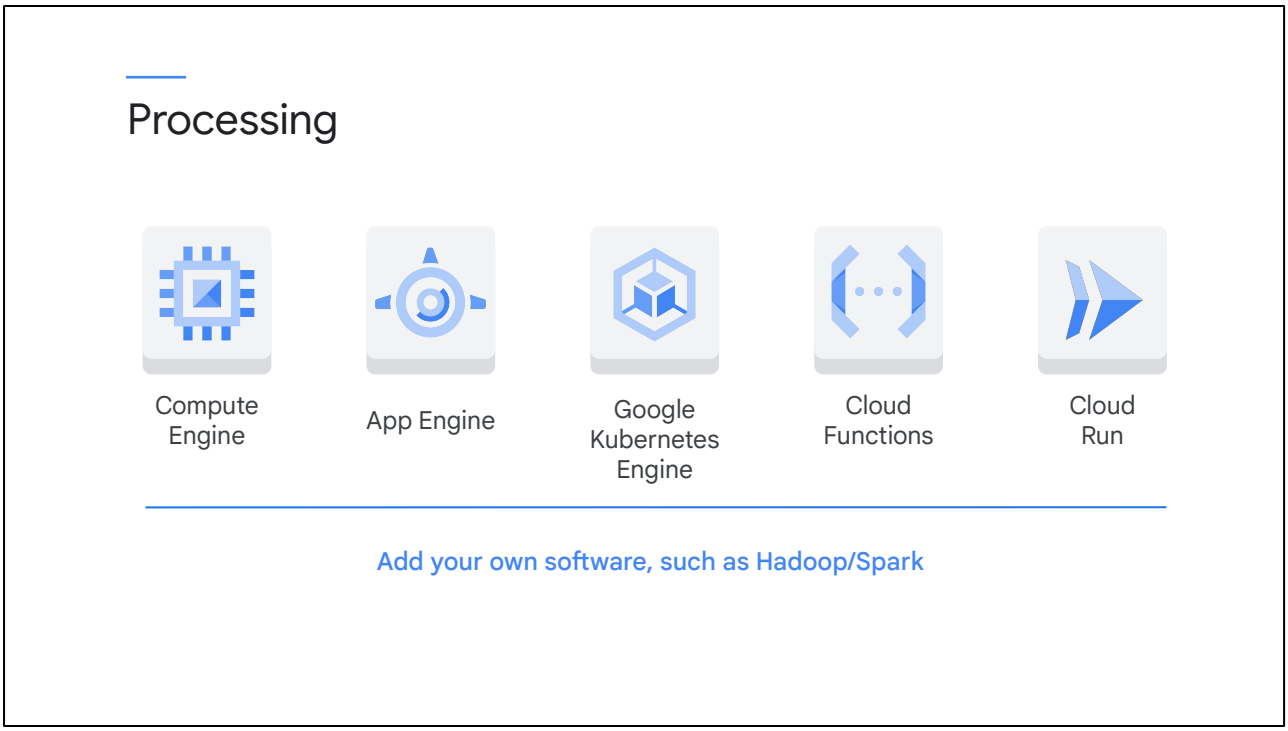
Storage and databases



Serverless services remove more of the IT responsibility, so managing the underlying servers is not part of your overhead and the individual instances are not visible.



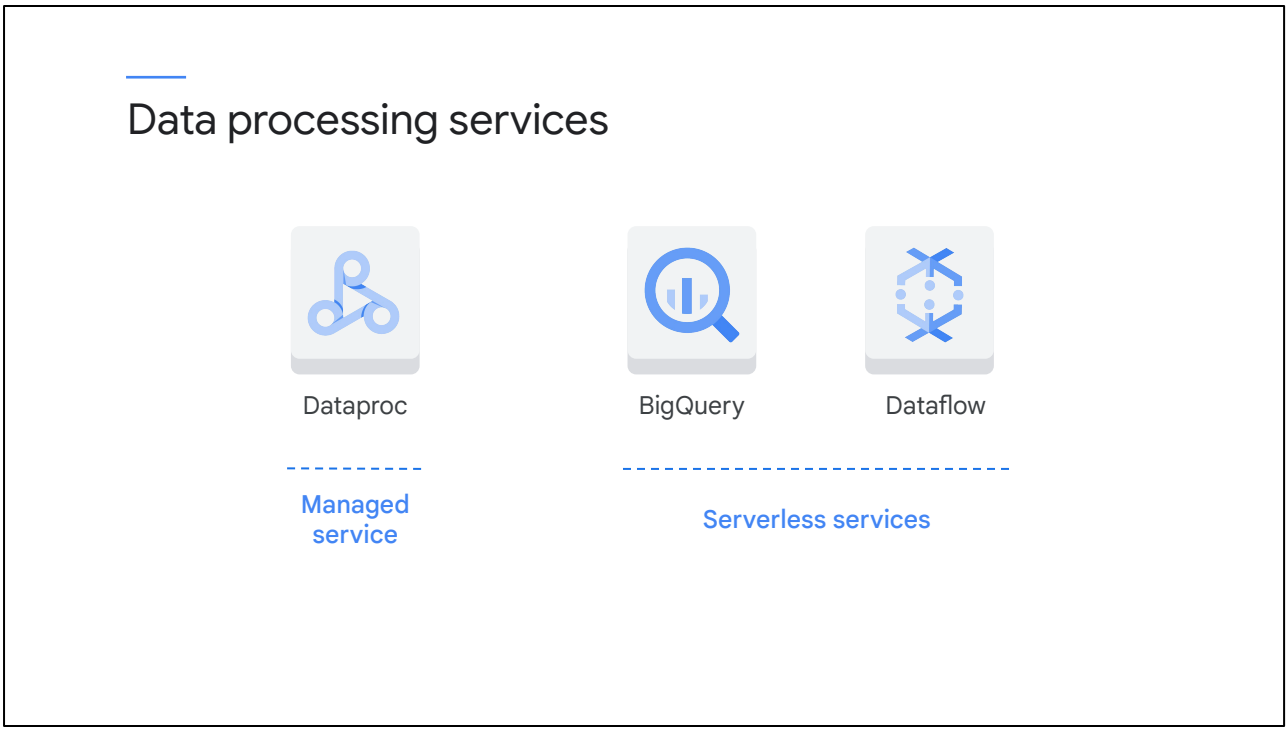
A more recent addition to this list is Firestore. Firestore is a NoSQL document database built for automatic scaling. It offers high performance, and ease of application development. And it includes a "Datastore compatibility mode".



As mentioned, storage and databases provide limited processing capabilities. And what they do offer is 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 and computing power. So where do you get these resources?

You could use any of these computing platforms to write your own application or parts of an application that use storage or database services.

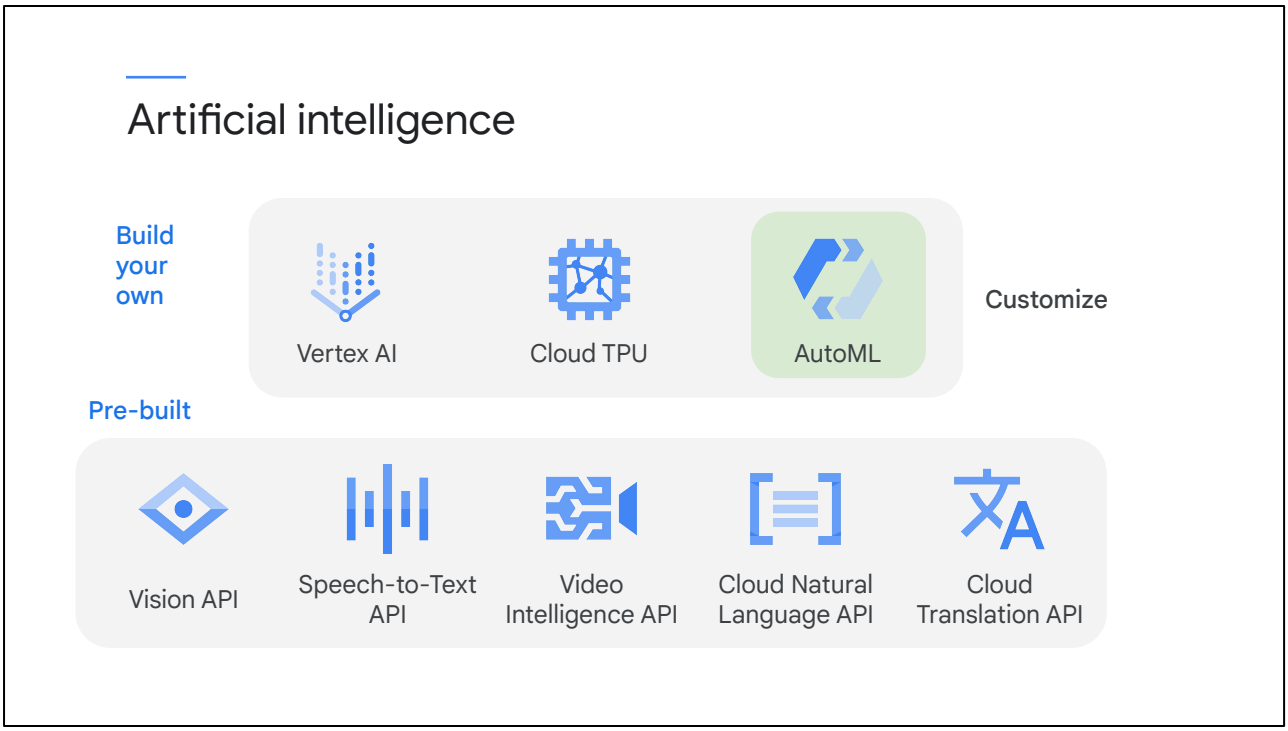
You could install open source software such as MySQL, an open source database, or Hadoop, an open source data processing platform, on compute engine.
 Build-your-own solutions are driven mostly by business requirements. They generally involve more IT overhead than using a cloud platform service.



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 solution 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 Dataproc, the data abstraction (with Spark) is a Resilient Distributed Dataset (RDD), and the processing abstraction is a Directed Acyclic Graph (DAG).

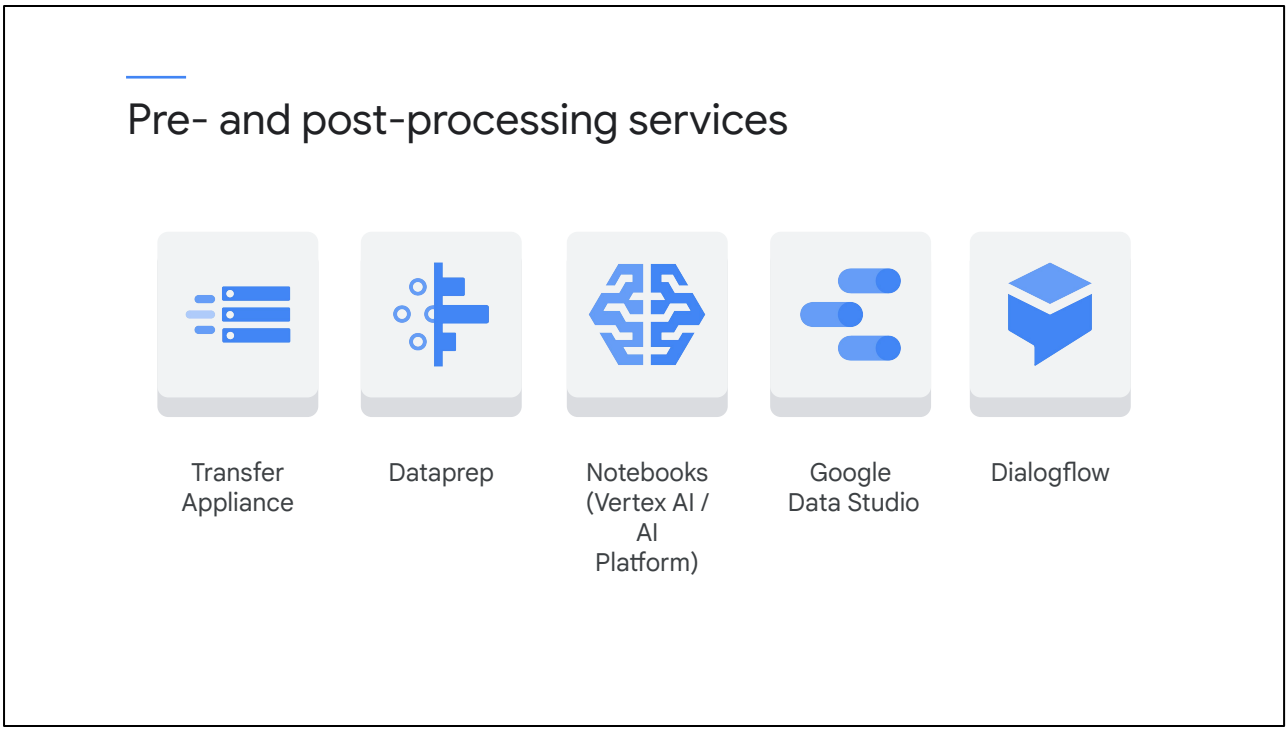
Implementing storage and processing as abstractions enable the underlying systems to adapt to the workload and the user / data engineer to focus on the data and business problems they are trying to solve.



There is great potential value in product or process innovation using Machine Learning. Machine Learning can make unstructured data, such as logs, useful by identifying or categorizing the data and thereby enabling business intelligence.

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.

Your exam tip is to understand the array of machine learning technologies offered on Google Cloud and when you might want to use each.



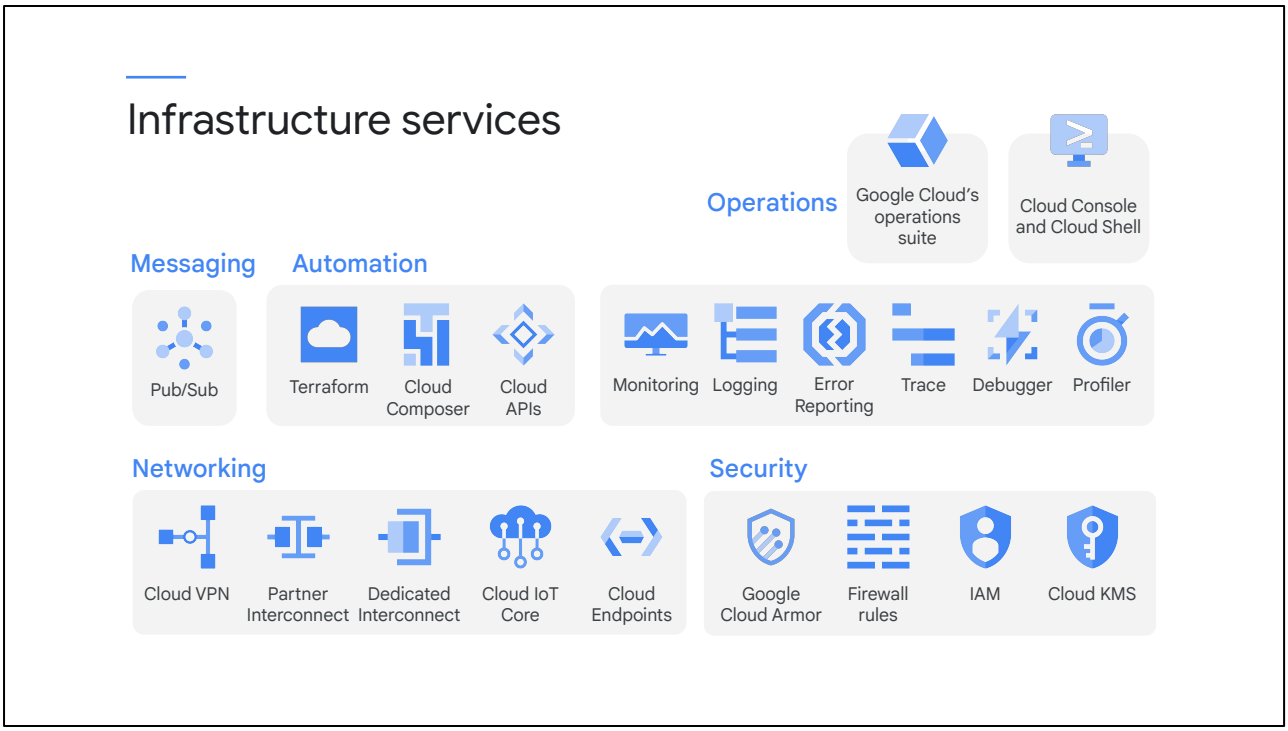
A Data Engineering solution involves data ingest, management during processing, analysis, and visualization. These elements can be critical to the business requirements.

Here are a few services that you should generally be familiar with:

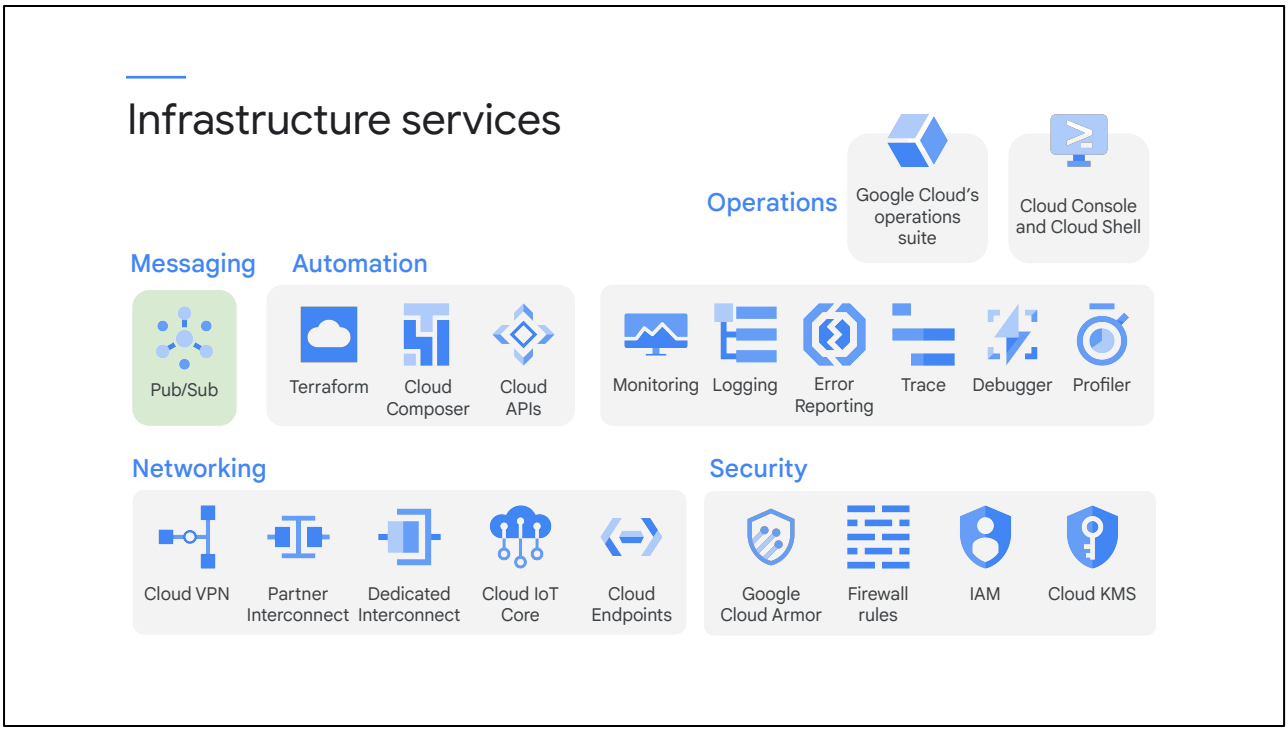
- Data Transfer Services operate online and a Data Transfer Appliance is a shippable device that are used for synchronizing data in the cloud with an external source
- Cloud Data Studio is used for visualization of data after it has been processed.
- Dataprep is used to prepare or condition data and to prepare pipelines before processing data.

Notebooks is a self-contained workspace that holds code, executes the code, and displays results.

Dialogflow is a service for creating Chatbots. It uses AI to provides a method for direct human interaction with data.

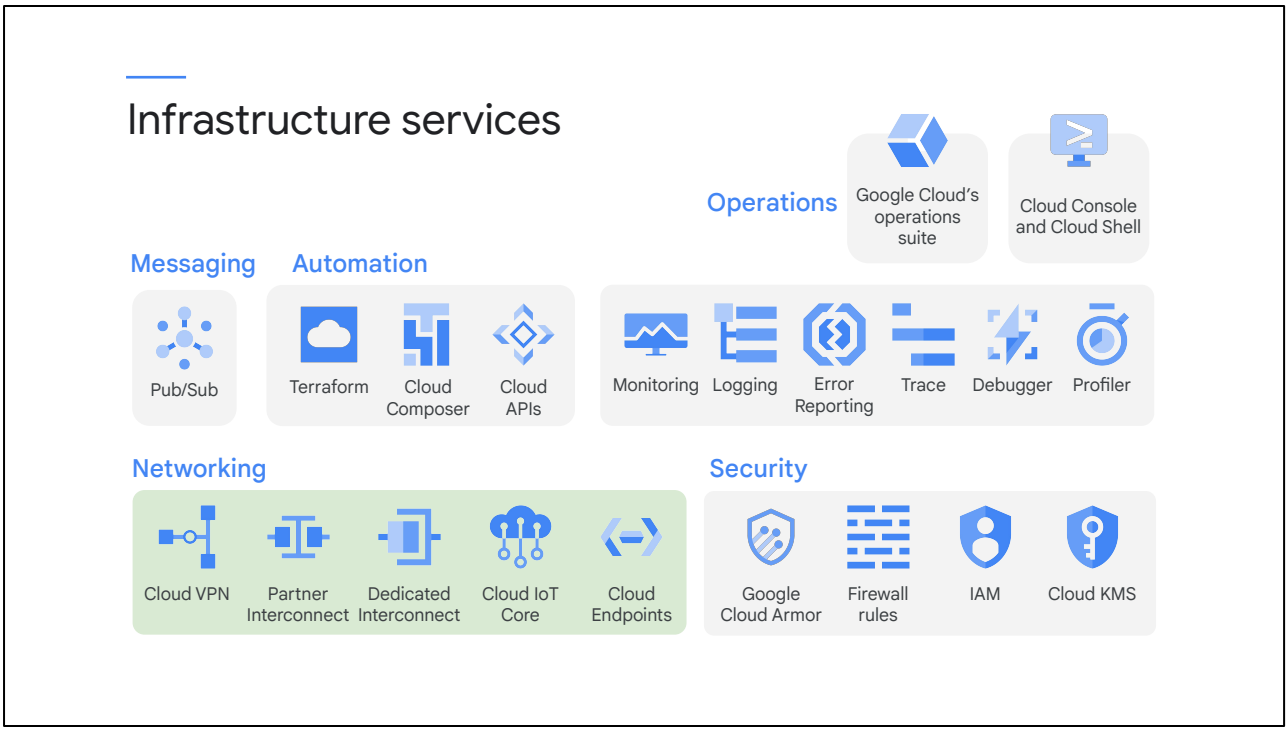


Your exam tip here is to familiarize yourself with infrastructure services that show up commonly in Data Engineering solutions. Often they are employed because of key features they provide.

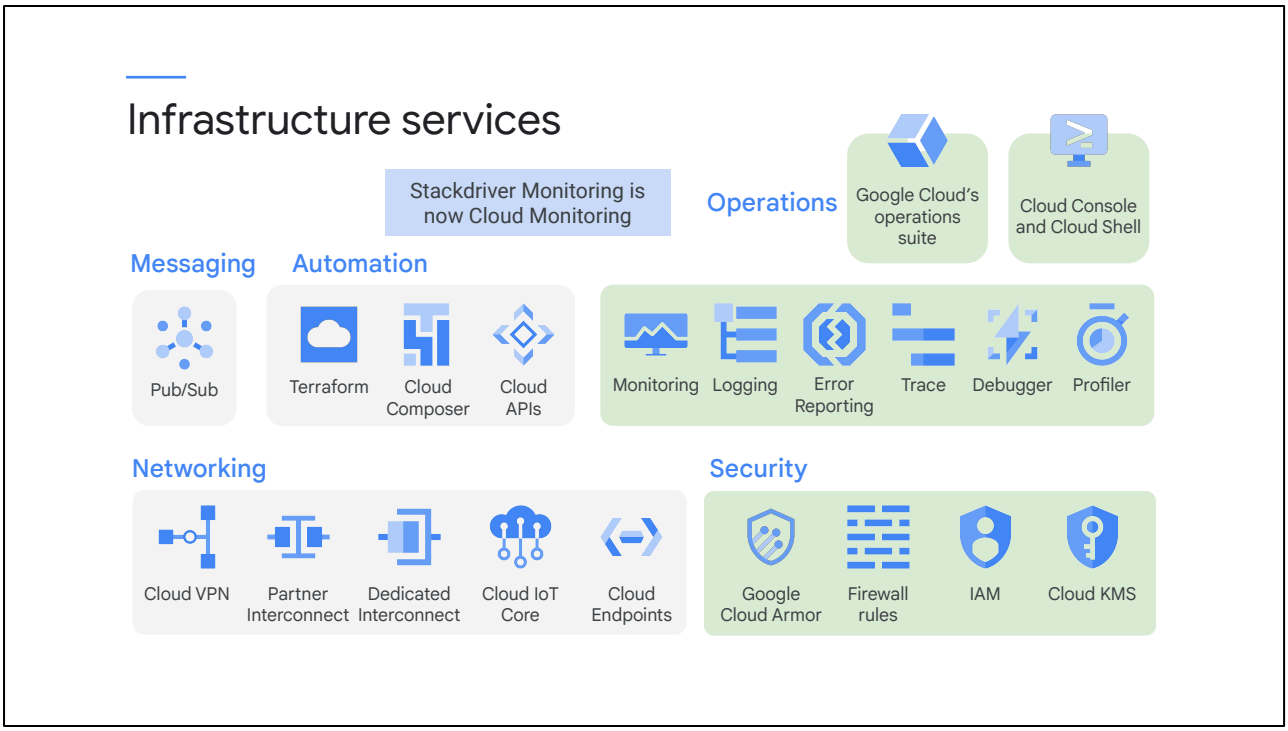


For example, Pub/Sub can hold a message for up to seven days, providing resiliency to Data Engineering solutions that otherwise would be very difficult to implement.

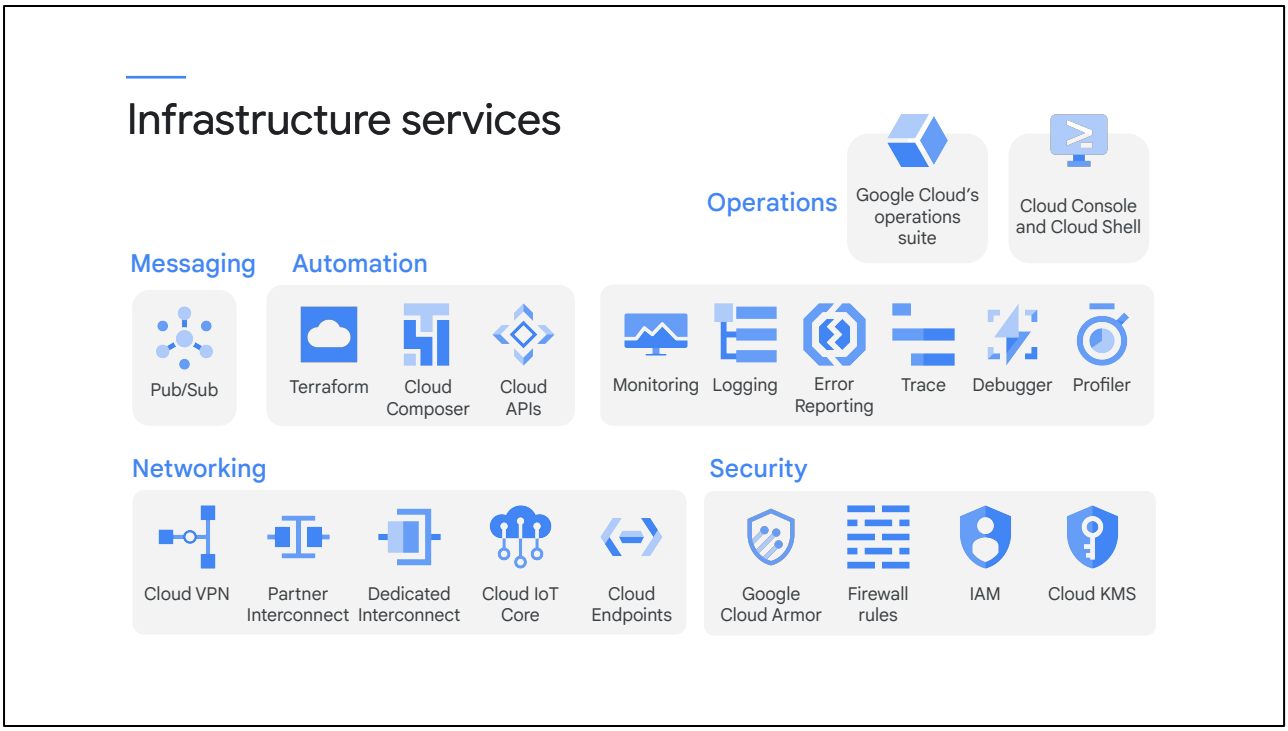
Every service in Google Cloud could be used in a Data Engineering solution. However, some of the most common and important services are shown here. Pub/Sub, a messaging service, features in virtually all live or streaming data solutions because it decouples data arrival from data ingest.



Cloud VPN, Partner Interconnect, or Dedicated Interconnect play a role whenever there is data on premise that must be transmitted to services in the 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 the Cloud Console and data sent to Cloud Monitoring.



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 contribute to the data solutions, and to be familiar with key features and options.

There are a lot of details that I wouldn't memorize. For example, the exact number of IOPS supported by a specific instance, is something I would expect to look up, not know. Also, the cost of a particular instance type compared with another instance type -- the actual values -- is not something I would expect I needed to know as a Data Engineer. I would look those details up if I needed them. However, the fact that an n4-standard instance has higher IOPS than an n1-standard instance, or that the n4-standard costs more than an n1-standard are concepts that I WOULD need to know as a Data Engineer.

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 **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**.

TIP

It is good to know HOW data is stored.

What purpose or use case is the storage/database optimized for?

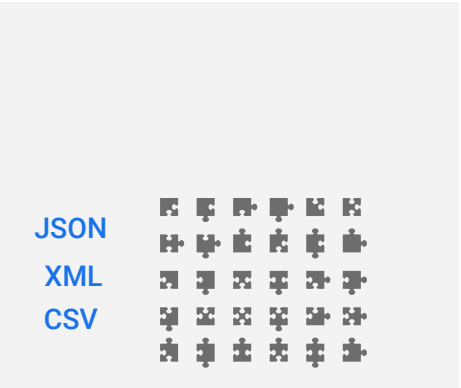
The Data Engineer needs to be familiar with basic concepts and terminology of data representation.

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 an SQL database such as Cloud SQL or Cloud Spanner.

If an exam question describes an Entity and a Kind—which are concepts used in 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.

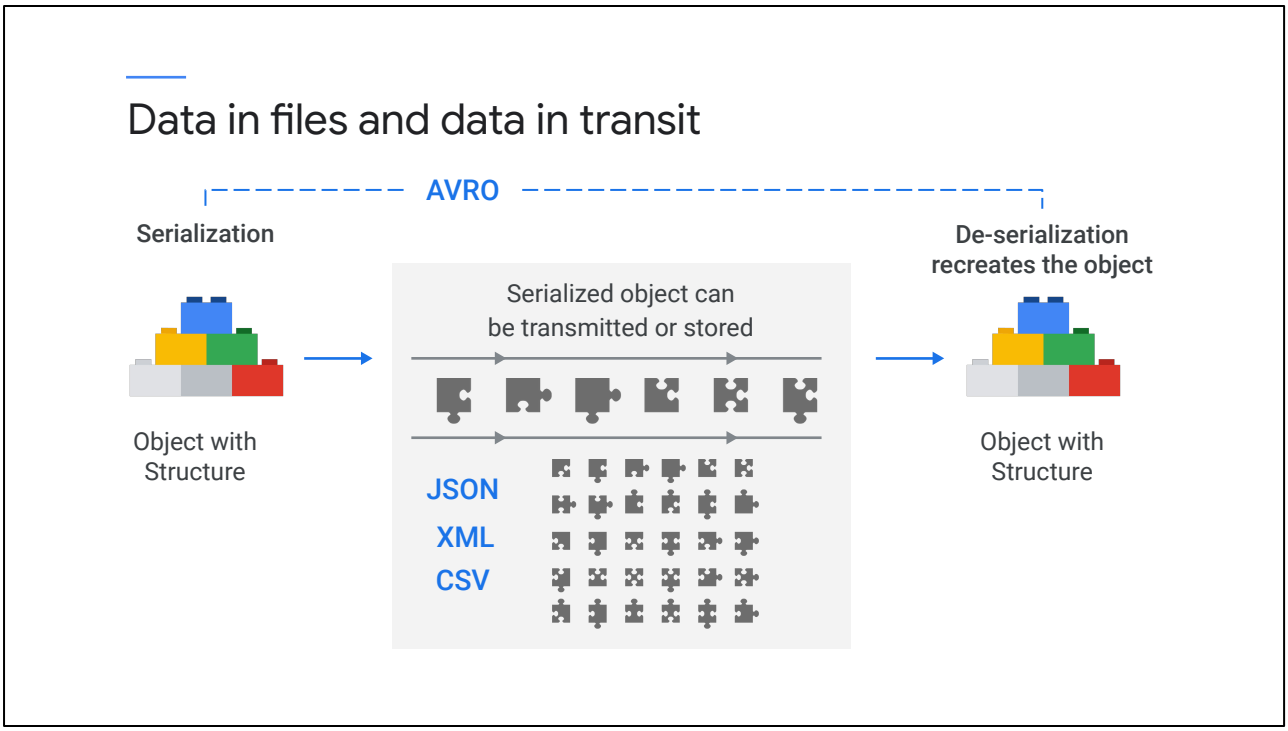
Exam tip is that it is good to know HOW data is stored and what purpose or use case is the storage/database optimized for.

Data in files and data in transit



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 not only represents the data but also the relationships.

- CSV, which stands for Comma Separated Values is a simple file format used to store tabular data.
- XML, which stands for Extensible Markup Language, was designed to store and transport data, and was designed to be self-descriptive.
- JSON, which stands for 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.



Networking transmits serial data as a stream of bits -- zeroes and ones. And data is stored as bits. That means if you have a data object with a meaningful structure to it. you need some method to flatten and serialize the data first, so that it is just zeroes and ones. Then it can be transmitted and stored. And when it is retrieved, the data needs to be de-serialized to restore the structure into a meaningful data object. One example of software that does this is Avro.

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

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.

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.

Now why did I mention the NUMERIC data type? Because to understand NUMERIC, you have to already know the difference between integer and floating point numbers, and you already have to know about rounding errors that can occur when performing math on some kinds of floating point data representations. So if you understand this you understand a lot of the other items you ought to know for SQL and data engineering.

You should also make sure you are familiar with these basic data types.

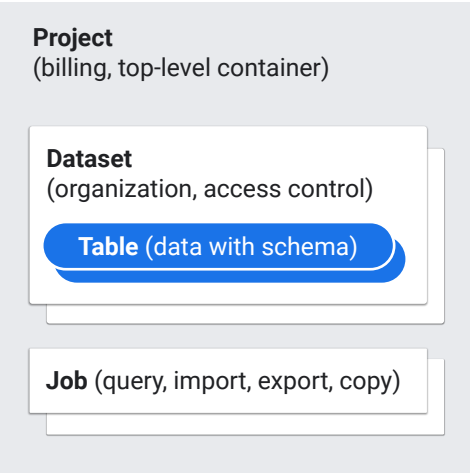
BigQuery datasets, tables, and jobs

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.



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 Google Cloud exists inside a Project. And besides security and access control, a Project is what 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 a SQL query. Although there are some update and maintenance activities supported using Data Manipulation Language, or DML.

Exam tip -- know the hierarchy of objects within a data technology and how they relate to one another.

BigQuery storage is columnar

- Record-oriented storage.
- Supports transactional updates.

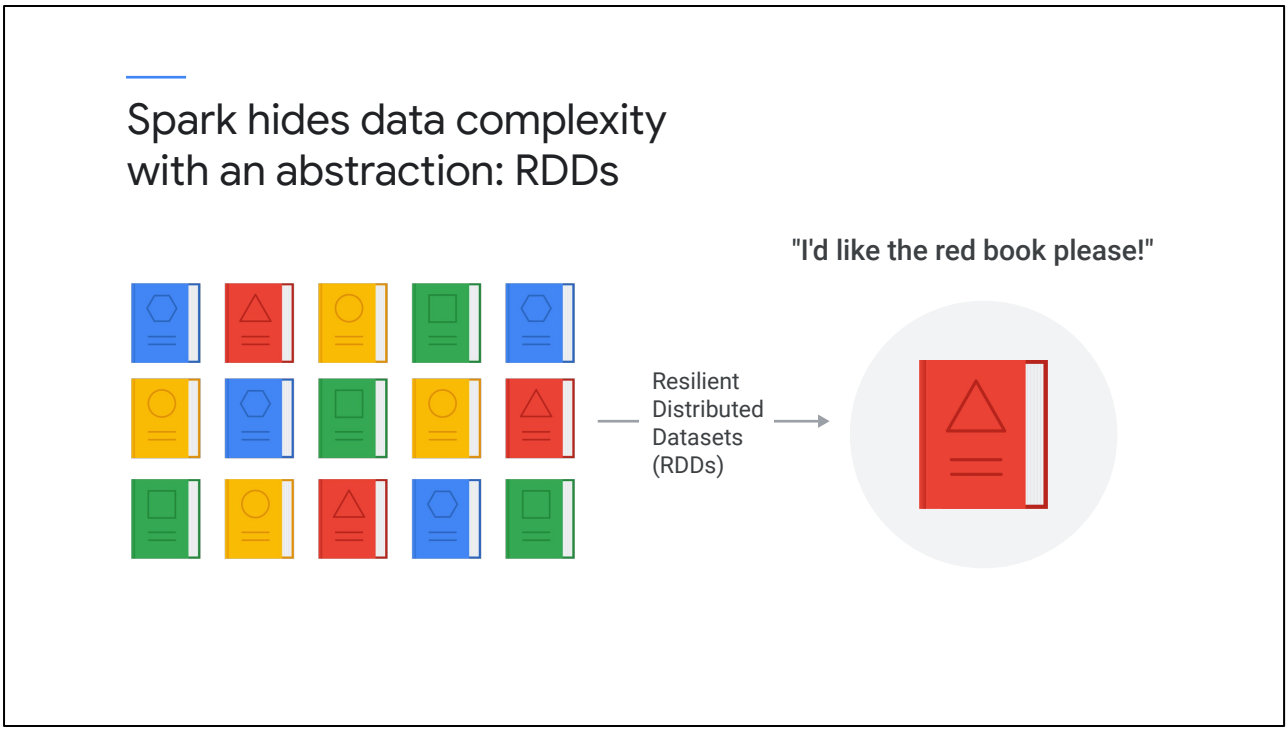
- Each column in a separate, compressed, encrypted file that is replicated 3+ times.
- No indexes, keys, or partitions required.
- For immutable, massive datasets.

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.

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

You can stream append data easily to BigQuery tables, but you can't easily change existing values.

Replicating the data 3 times also helps the system determine optimal compute nodes to do filtering, mixing, and so forth.



You treat your data in Dataproc in Spark as a single entity. But Spark knows the truth.

Your data is stored in Resilient Distributed Datasets or RDDs.

RDDs are an abstraction that hides the complicated details of how data is located and replicated in a cluster.

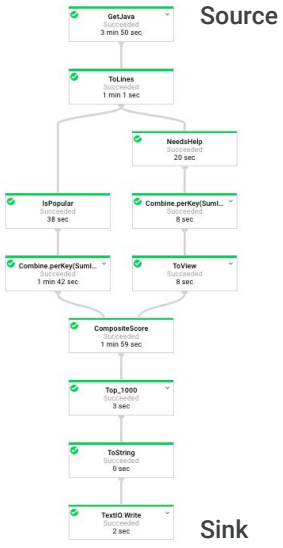
Spark partitions data in memory across the cluster and knows how to recover the data through an RDD's lineage, should anything go wrong.

Spark has the ability to direct processing to occur where there are processing resources available.

Data partitioning, Data replication, Data recovery, Pipelining of processing -- all are automated by Spark so you don't have to worry about them.

Here is an exam tip: You should know how different services store data, and how each method is optimized for specific use cases, as previously mentioned. But also understand the key value of the approach. In this case, RDDs hide complexity and allow Spark to make decisions on your behalf.

Dataflow terms and concepts: PCollection



There are a number of concepts that you should know about Dataflow.

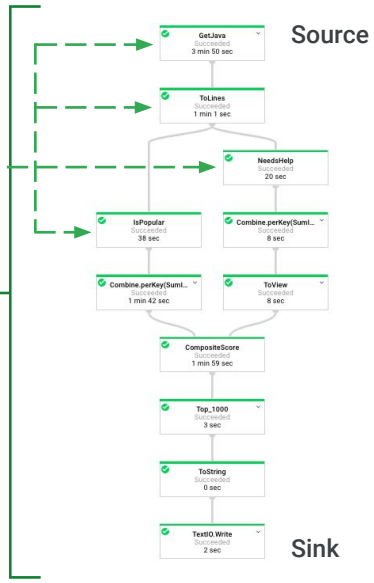
Your data in Dataflow is represented in PCollections.

The pipeline shown in this example reads data from BigQuery, does a bunch of processing and writes its output to Cloud Storage.

Dataflow terms and concepts: PCollection

Each step is a Transform

Together, they form a Pipeline



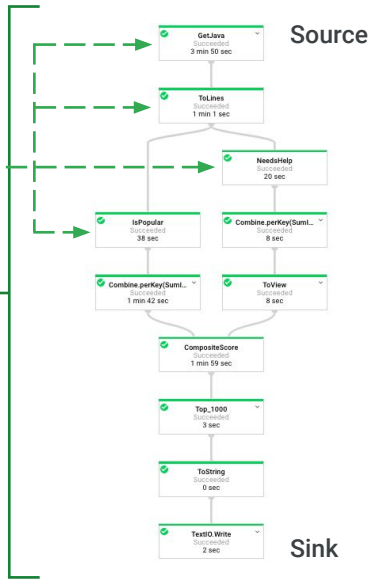
In Dataflow, each step is a transformation, and the collection of transforms makes a pipeline.

Dataflow terms and concepts: PCollection

Each step is a Transform

Together, they form a Pipeline

The Pipeline is executed on the cloud by a Runner



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.

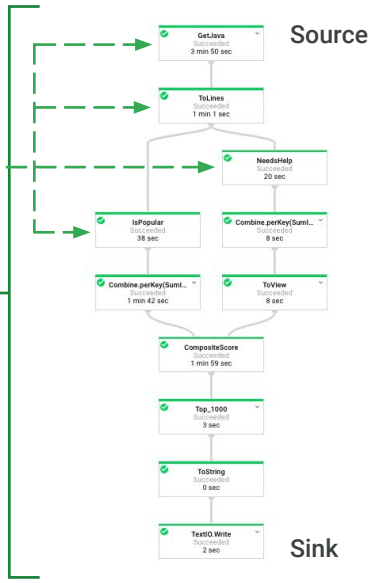
Dataflow terms and concepts: PCollection

1. Each step is elastically scaled.
2. Each Transform is applied on a PCollection.
3. The result of an `apply()` is another PCollection.

Each step is a Transform

Together, they form a Pipeline

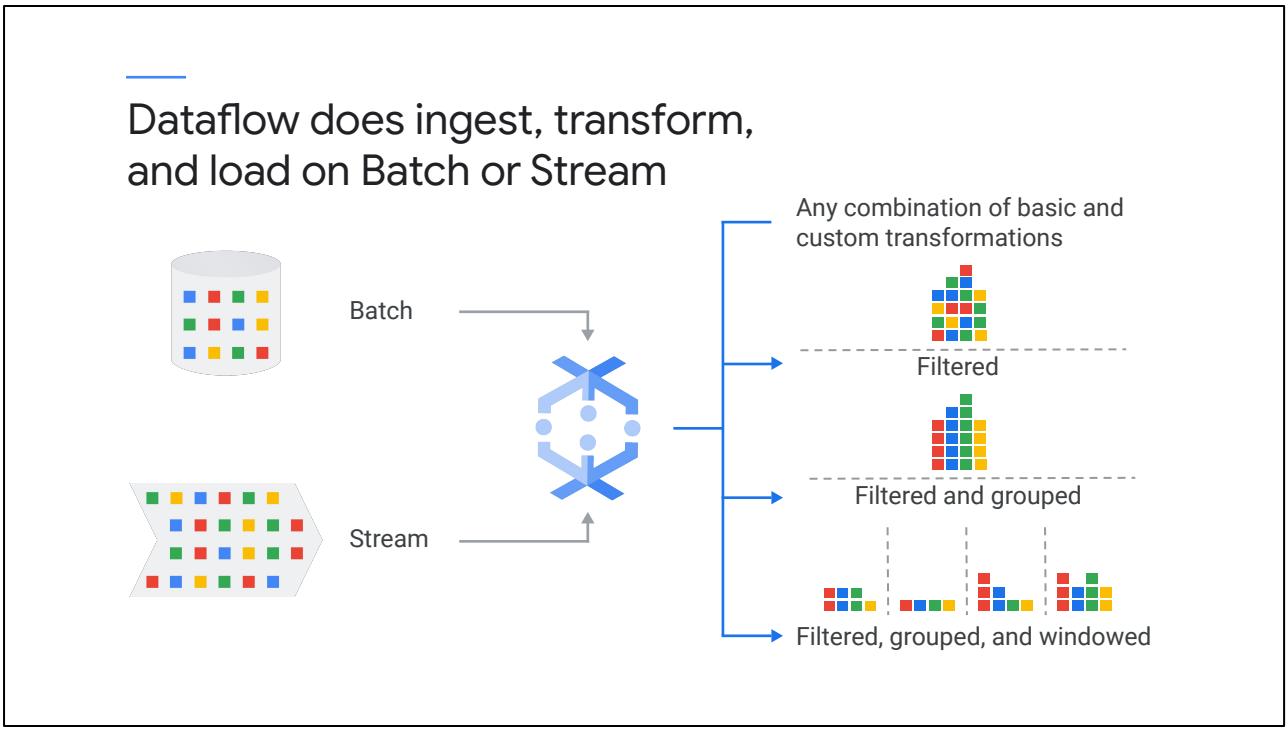
The Pipeline is executed on the cloud by a Runner



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."

The idea is to write Python or Java code and deploy it to Dataflow which then executes the pipeline in a scalable serverless context.

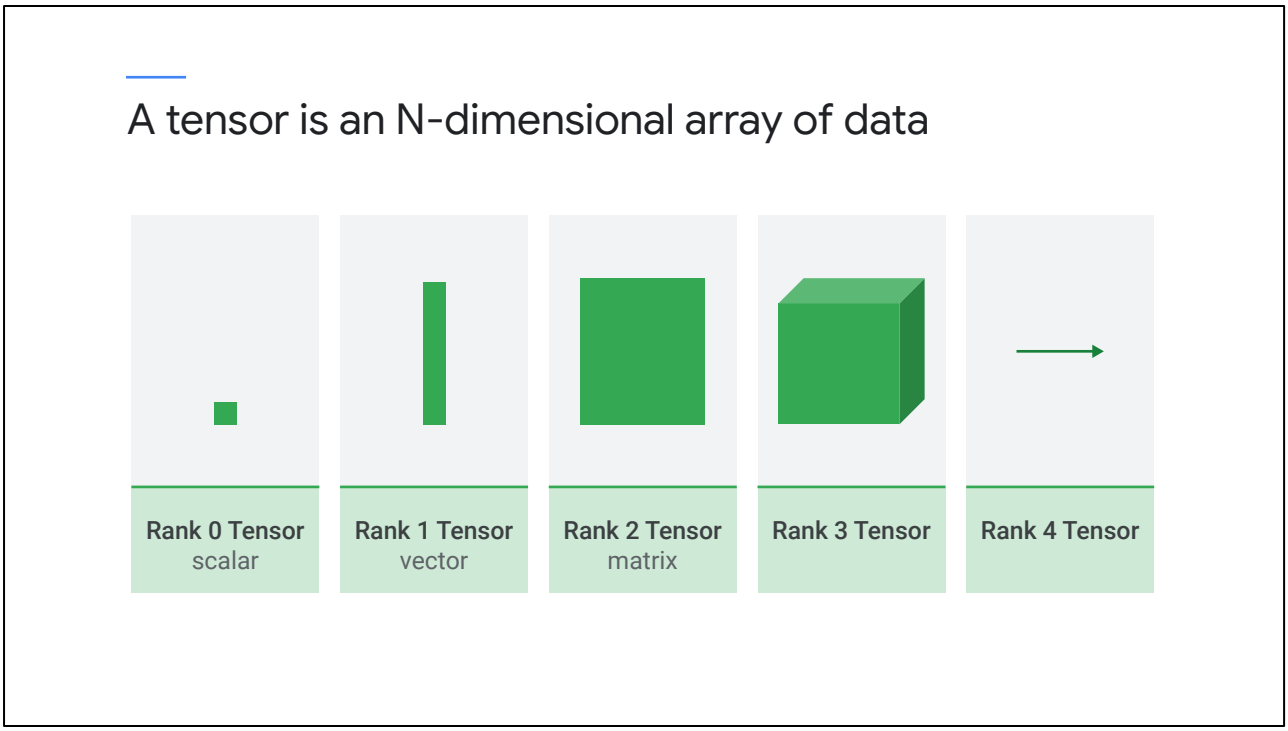
Unlike Dataproc, there is no need to launch a cluster or scale the cluster. That is handled automatically.



Dataflow is designed to use the same pipeline, the same operations, the same code for both Batch and Stream processing. 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. One example is a simple average. You add up all the values and divide by the total 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 you never know when to divide or what number to use. So what Dataflow does is it allows you to define a period or window and to calculate the average within that window. That's an example of how both kinds of data can be processed with the same single block of code. Filtering and Grouping are also supported.

Many Hadoop workloads can be run more easily and are easier to maintain with Dataflow. But PCollections and RDDs are not identical. So existing code has to be redesigned and adapted to run in the Dataflow pipeline. This can be a consideration because it can add time and expense to a project.



Your data in TensorFlow is represented in tensors.

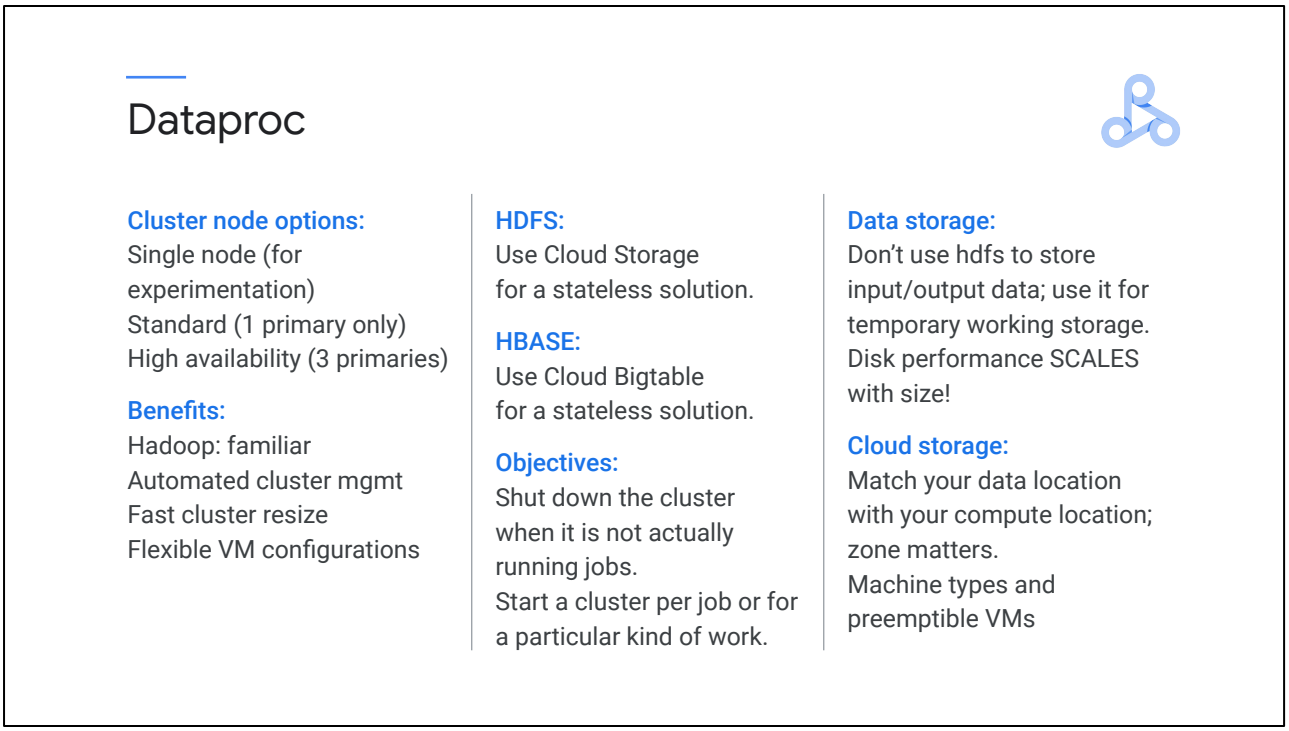
Where does the name Tensorflow come from? Well, the "flow" is a Pipeline, just like we discussed in Dataflow. But the data object in Tensorflow is not a PCollection, but something called a Tensor. A Tensor is a special mathematical object that unifies scalars, vectors, and matrixes. Tensor-zero is just 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.

Tensorflow is the open source code that you use to create Machine Learning models. 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 or rank of tensor. So it makes solving some problems much easier.

The next section of the exam guide is designing data pipelines.

You already know how the data is represented. In Dataproc in Spark it is RDDs and in Dataflow it is a PCollection. In BigQuery the data is in a Dataset in Tables.

And you know that a pipeline is some kind of sequence of actions or operations to be performed on the data representation. But each service handles a pipeline differently.



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 things you should know about Dataproc is how to use it differently from standard Hadoop. If you store your data external 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 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.

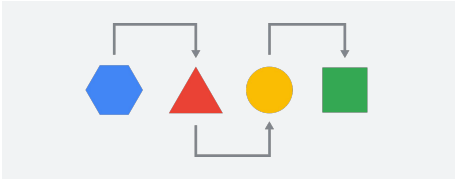
When you have a stateless Dataproc cluster, it typically takes only about 90 seconds for the cluster to start up and become active.

Dataproc supports Hadoop, Pig, Hive, and Spark.

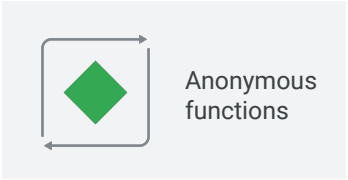
One exam tip: Spark is important because it does part of its pipeline processing in memory, rather than copying from disk. For some applications, this makes Spark extremely fast.

Dataproc Spark RDD pipeline operations use lazy execution

Transformations are "lazy"



Actions: "Do it now"



TIP

Spark can wait till all the requests are in before applying resources.

With a SPARK pipeline you have two different kinds of operations; transforms and actions. Spark builds it's pipeline using an abstraction called a Directed Graph. Each transform builds additional nodes into the graph. But Spark doesn't execute the pipeline until it sees an Action. Very simply, Spark waits until it has the whole story, all the information. This allows Spark to choose the best way to distribute the work and run the pipeline. The process of waiting on transforms and executing on actions is called Lazy Execution.

For a transformation the input is an RDD and the output is an RDD. When Spark sees a Transformation, it registers it in the directed graph. Then it waits.

An Action triggers Spark to process the pipeline. The output is usually a "result format", such as a text file, rather than an RDD.

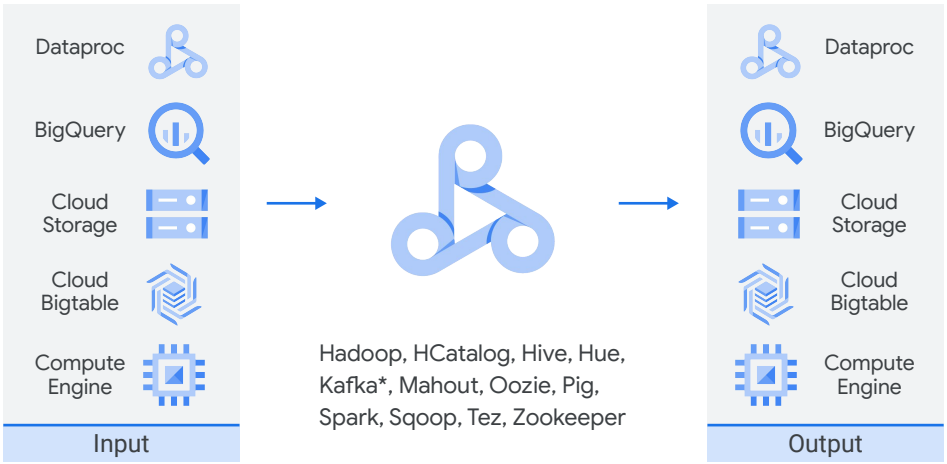
Transformations and Actions are API calls that reference the functions you want them to perform.

Anonymous functions in Python, Lambda functions are commonly used to make the API calls.

They are a self-contained way to make a request to Spark. Each one is limited to a single specific purpose. They are defined inline, making the sequence of the code easier to read and understand. And because the code is used in only one place, the function doesn't need a name and doesn't clutter the namespace.

An interesting and opposite approach, where the system tries to process the data as soon as it is received is called Eager Execution. TensorFlow, for example, can use both lazy and eager approaches.

Use Dataproc to run open source software



Dataproc has connectors to all kinds of Google Cloud resources. You can read from Google Cloud sources and write to Google Cloud sources and use Dataproc as the interconnecting glue.

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 Google Cloud would be Pub/Sub. Do you know what the alternative on Google Cloud is to the open source HBASE? It is Cloud Bigtable. And the alternative to HDFS? Cloud Storage.

Installing and running Hadoop open source software on the Dataproc cluster is also available.

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.

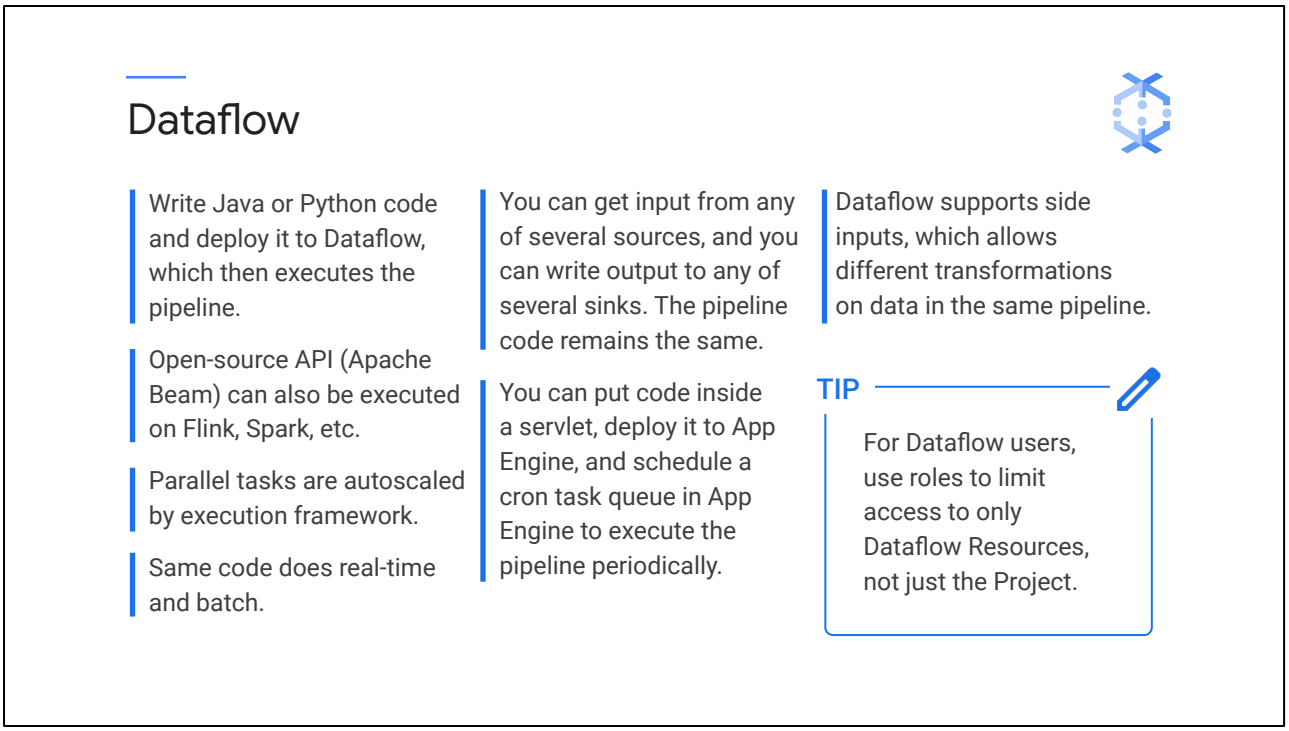
Specified by `file_prefix:property=value` in gcloud SDK.

Use initialization actions, which are init scripts, to load, install, and customize software. The cluster itself has limited properties that you can modify. But if you use Dataproc as suggested, starting a cluster for each kind of work, you won't need to tweak the properties the way you would with data center Hadoop.

Here is a tip about modifying the Dataproc cluster. If you need to modify the cluster, consider whether you have the right Data Processing solution. There are so many services available on Google Cloud, you might be able to use a service rather than hosting your own on the cluster.

If you are migrating data center Hadoop to Dataproc, you may already have customized Hadoop settings that you would like to apply to the cluster. You may want to customize some cluster configurations so that it works similarly. This is supported in a limited way by Cluster Properties.

Security in Dataproc is controlled by access to the cluster as a resource.



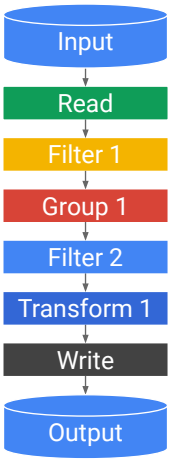
Here are some things you should know about Dataflow. You can write pipeline code in Java or Python. You can use the open source Apache Beam API to define the pipeline and submit it to Dataflow. Then Dataflow provides the execution framework.

Parallel tasks are automatically scaled by the framework. And the same code does real-time streaming and batch processing. One great thing about Dataflow is that you can get input from many sources and write output to many sinks, but the pipeline code in between remains the same. Dataflow supports side inputs. That is where you can take data and transform it in one way, and transform it in a different way in parallel, so that the two can be used together in the same pipeline.

Security in Dataflow is based on assigning roles that limit access to the Dataflow resources.

So your exam tip is "For Dataflow users, use roles to limit access to only Dataflow resources, not just the Project."

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();
```

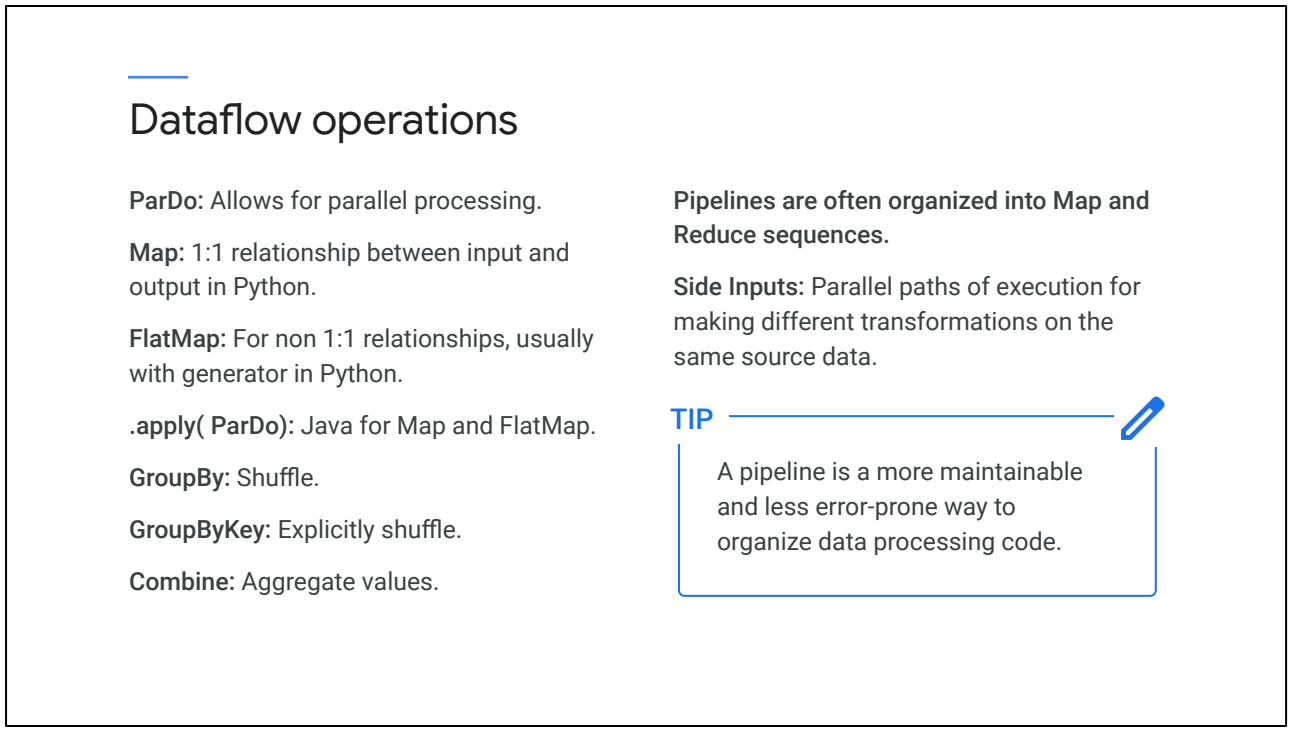
Open-source API (Apache Beam) can be executed on Flink, Spark, etc. also

Parallel task (autoscaled by execution framework)

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

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

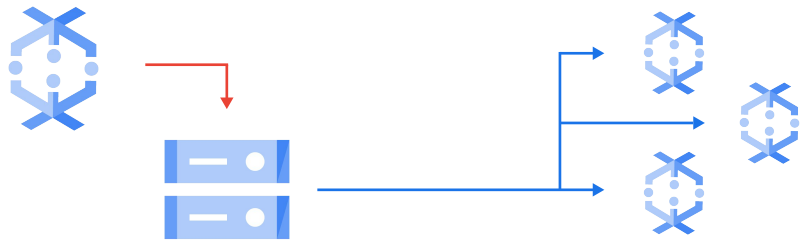
Each step of the pipeline does a filter, group, transform, compare, join, and so on. Transforms can be done in parallel.



Here are some of the most commonly used Dataflow operations. Do you know which operations are potentially computationally expensive? GroupByKey, for one, could consume resources on Big Data. This is one reason you might want to test your pipeline a few times on sample data to make sure you know how it scales before executing at production scale.

Exam tip: A pipeline is a more maintainable way to organize data processing code than, for example, an application running on an instance.

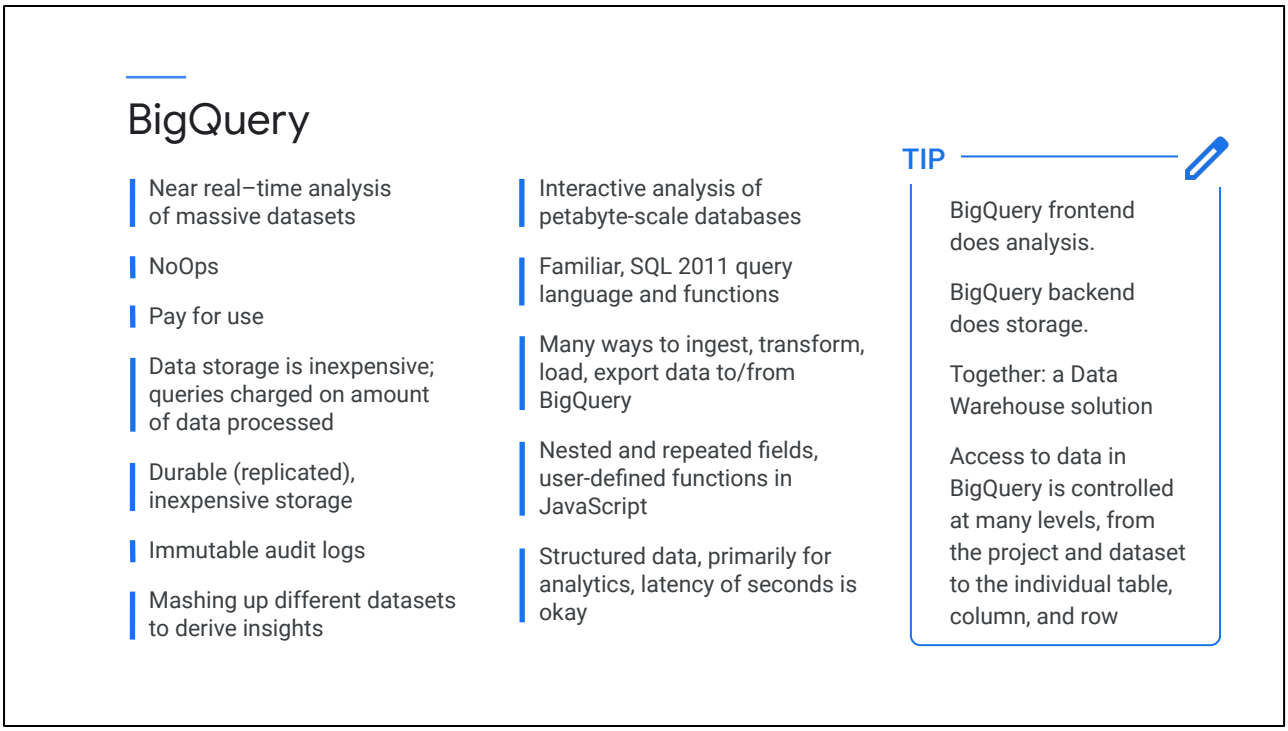
Template workflow supports non-developer users



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

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.

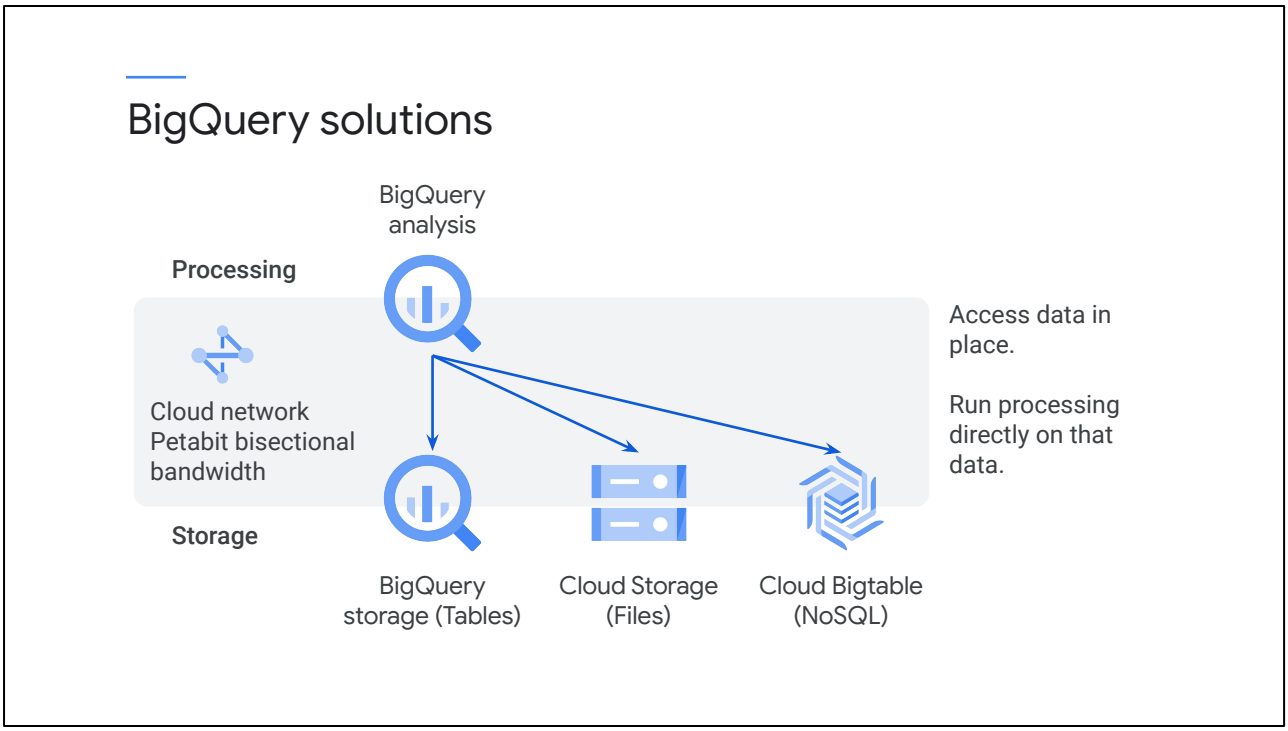
Your exam tip here is that Dataflow Templates open up new options for separation of work and that means better security and resources accountability.



BigQuery is two services. A front end service that does analysis and a back end service that does storage. It offers near real-time analysis of massive datasets. The data storage is durable and inexpensive. And you can connect and work with different datasets to derive new insights and business value. BigQuery uses SQL for queries, so it is immediately usable by many data analysts.

BigQuery is fast. But how fast is fast? Well, if you are using it with structured data for analytics, it could take a few seconds.

BigQuery connects to many services for flexible ingest and output. And it supports nested and repeated fields (for efficiency) and user-defined functions for extensibility.



Here is a major design tip. Separating Compute and Processing from Storage and Database enables serverless operations.

BigQuery has its own Analytic / SQL Query front-end, available in Console and from the command line with bq. It is "just" a query engine.

The back end Data Warehouse part of BigQuery stores data in tables.

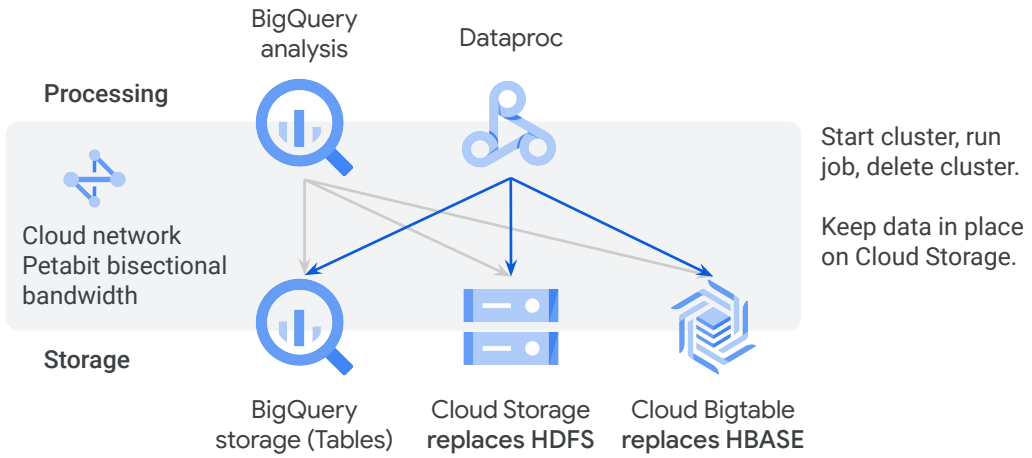
But BigQuery also has a connector to Cloud Storage. This is commonly used to work directly with CSV files.

BigQuery has a connector to Cloud Bigtable as well.

If you need more capabilities than a Query Engine, consider Dataproc or Dataflow.

What makes all this possible is the Cloud Network with Petabit speeds. It means that storing data in a service like Cloud Storage can be almost as fast and in some cases faster than storing the data locally where it will be processed. In other words, the network turns the concept of Hadoop and HDFS upside down. It is more efficient, once again, to store the data separate from the processing resources.

Dataproc solutions

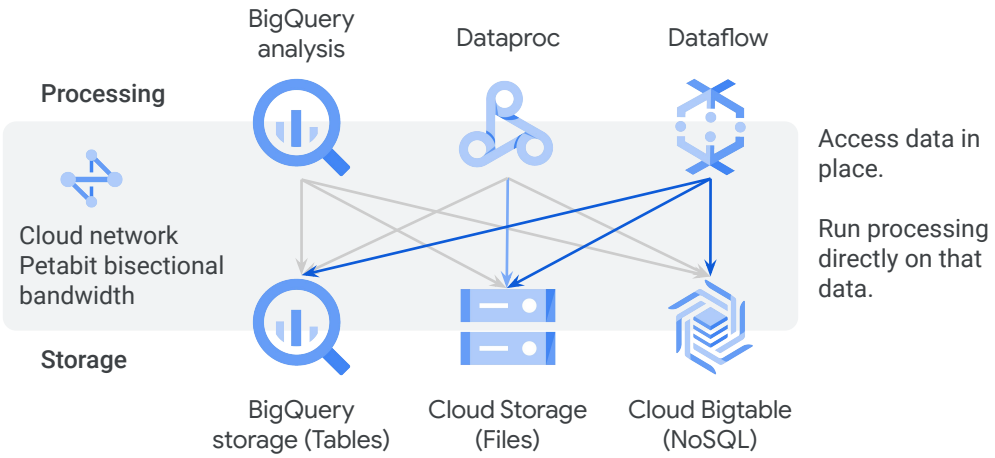


Now we are starting to explore how all these platform parts fit together to create really flexible and robust solutions.

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.

Dataflow solutions



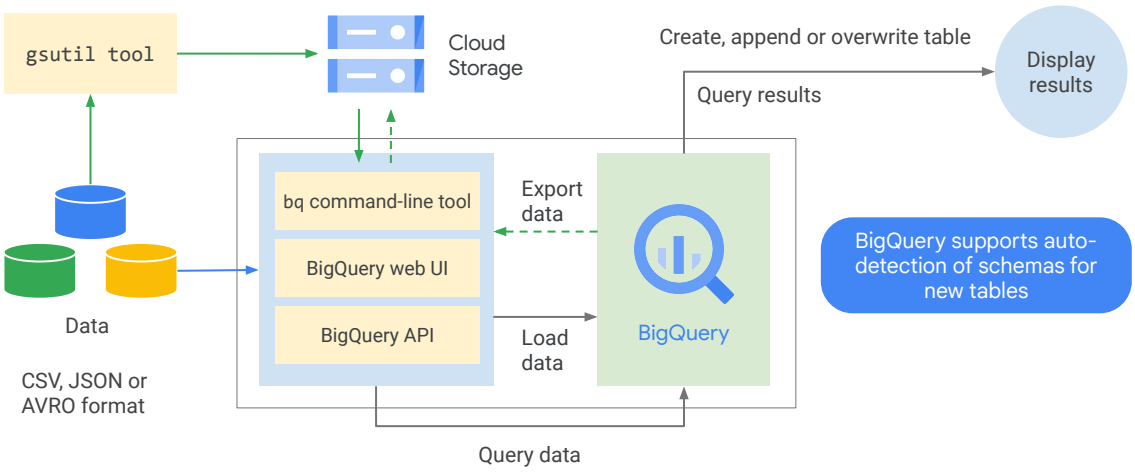
Dataproc and Dataflow can output separate files as CSV files in Cloud Storage. In other words, you can have a distributed set of nodes or servers processing the data in parallel and writing the results out in separate small files. This is an easy way to accumulate distributed results for later collating.

- Access any storage service from any Data Processing service.
- Dataflow is an excellent ETL solution for BigQuery.
- Use Dataflow to aggregate data in support of common Queries.

Now that you have all the pieces, let's start looking at how to put them together into data processing infrastructure.

There are a few common assemblies of services and technologies. You will see them used repeatedly in different contexts.

Data ingest solutions: CLI, web UI, or API

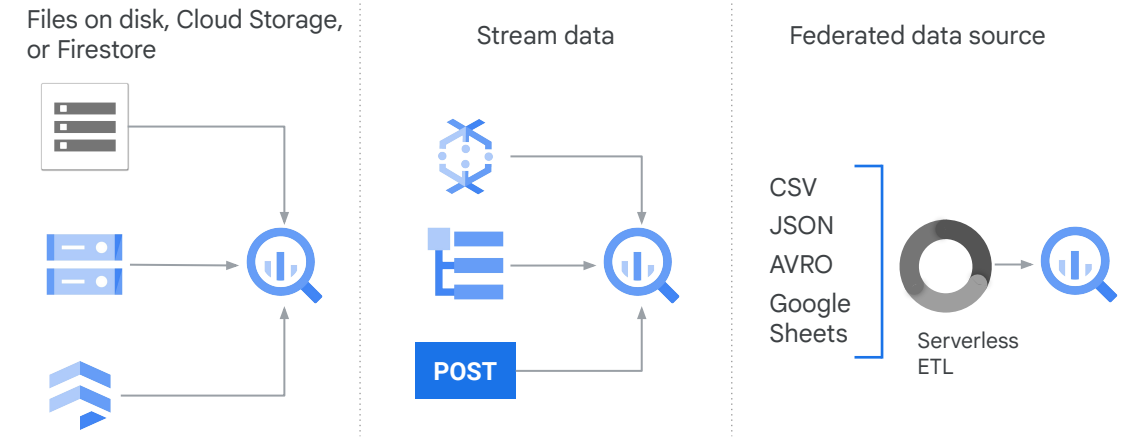


Here is an example showing the many manual ingest solutions available.

You can use the gsutil command line tool to load files into Cloud Storage. But if you want to load data into BigQuery, you need to be able to identify the structure.

The BigQuery command line tool, bq, is good for uploading large data files and for scheduling data file uploads. You can use the command to create tables, define schemas, load data, and run queries. The bq command is available on a Compute Engine instance, in Cloud Shell, or you can install it on any client machine as part of the Google Cloud Software Development Kit or Cloud SDK.

Three ways of loading data into BigQuery

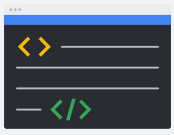


You can load data into BigQuery from the Cloud Console.

You can stream data using Dataflow, from Cloud Logging, or you can use POST calls from a program.

And it is very convenient that BigQuery can automatically detect CSV and JSON format files.

Options for transferring storage



gsutil

Storage Transfer Service

Transfer Appliance

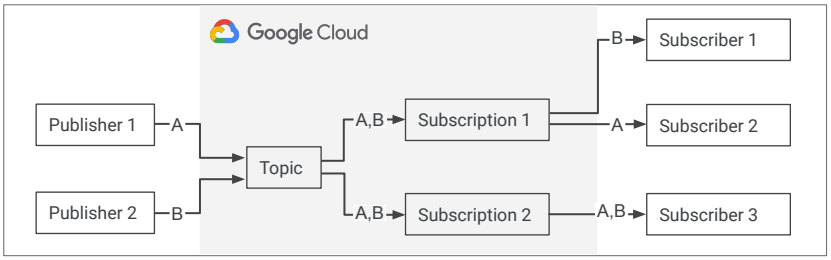
TIP

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.

Another tip; think about data in terms of the three "V's" -- Volume, Velocity, Variety. How much, how often, and how consistent?

This will guide you to the best approach for ingesting the data. In brief, use gsutil for uploading files. Use the Storage Transfer Service when the data is in another location such as another cloud. And use the Transfer Appliance when the data is too big to transfer electronically.

Pub/Sub



- 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.

The Pub/Sub message broker enables complete ingest solutions. It provides loose coupling between systems and long-lived connections between systems.

Exam tip: You need to know how long it Pub/Sub holds messages -- it is up to seven days. There are more details about Pub/Sub you should know. So if you are not familiar, you might want to review the documentation.

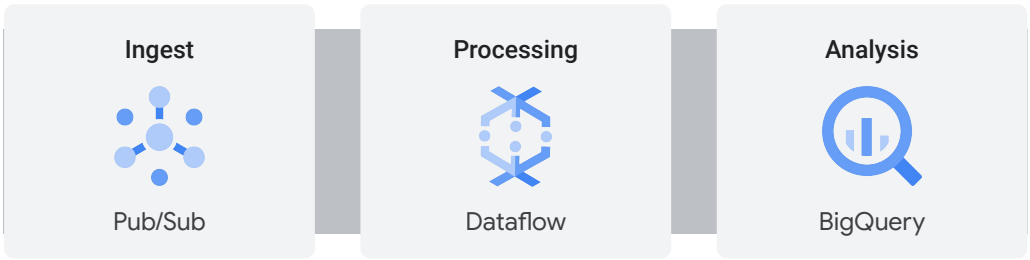
Common applications of Pub/Sub

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. You can avoid over-provisioning for traffic spikes with Pub/Sub.

If you use Pub/Sub with Dataflow you can get exactly once, ordered processing. With few exceptions, Pub/Sub delivers each published message at least once for every subscription and Dataflow handles de-duplication, ordering and windowing. Separation of duties enables a scalable solution that surpasses bottlenecks in competing messaging systems.

Serverless analytics solution

Stages of Analytics Lifecycle



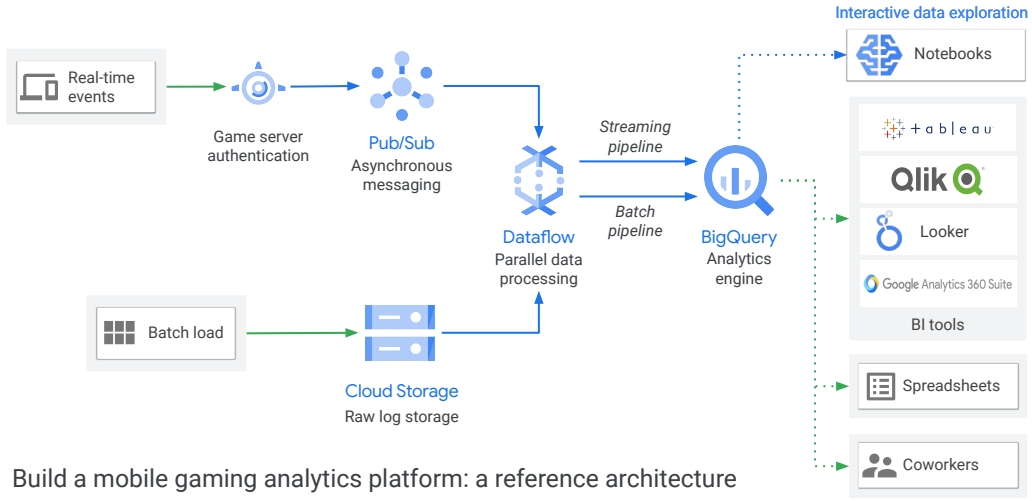
Why serverless?

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

This is the pattern you will often see. Pub/Sub for data ingest. Dataflow for data processing and ETL. And BigQuery for interactive analysis.

Exam tip -- be able to recognize this pattern in case scenarios.

Data analytics solution



Build a mobile gaming analytics platform: a reference architecture

This mobile gaming reference architecture illustrates the pattern at work.

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

Let's try some sample exam questions. Ready?

Scenario #1

Question

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 Dataflow to run the transformations.

B. Use BigQuery for storage. Use Dataproc to run the transformations.

C. Use Cloud Storage for storage. Use Dataflow to run the transformations.

D. Use Cloud Storage for storage. Use Dataproc to run the transformations.

The first question is: "You need a storage solution for CSV files. Analysts will run ANSI SQL queries. You need to support complex aggregate queries and reuse existing I/O-intensive custom Apache Spark transformations. How should you transform the input data? Looks like you've got a choice of BigQuery or Cloud Storage for the storage part of the solution, and Dataflow or Dataproc for the transformation and processing part of the solution.

Do you have an answer? How confident are you in the answer? This is a good time to consider whether you would bookmark this question and come back to it during an exam or whether you feel confident in your answer.

Let's see what the correct answer is.

Scenario #1

Answer

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 Dataflow to run the transformations.

B. Use BigQuery for storage. Use Dataproc to run the transformations.

C. Use Cloud Storage for storage. Use Dataflow to run the transformations.

D. Use Cloud Storage for storage. Use Dataproc to run the transformations.

The correct answer is "B". Use BigQuery for the storage solution and Dataproc for the processing solution.

Scenario #1

Rationale

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

A is not correct because you should not use 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 Dataflow.

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

Okay. Dataproc is correct because the question states you need to plan to reuse Apache Spark code. The CSV files could be in Cloud Storage or could be ingested into BigQuery.

In this case you need to support complex SQL Queries. So best to use BigQuery for storage. This is not a once-in-a-while straightforward case where you might consider just keeping the data in Cloud Storage.

Ready for another one?

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

Scenario #2

Question

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 Pub/Sub for input. Attach a timestamp to every message in the publisher.

B. Use 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.

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?

Ready to see the solution?

Scenario #2

Answer

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 Pub/Sub for input. Attach a timestamp to every message in the publisher.

B. Use 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.

The answer this time is A. Pub/Sub for input and attach a timestamp at the publisher.

Scenario #2

Rationale

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).

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

We can kind of figure that Apache Kafka is not the recommended solution in this scenario because you would have to set it up and maintain it. That could be a lot of work?

Why not just use the Pub/Sub service and eliminate the overhead? You need a timestamp to implement the rest of the solution. So applying it at ingest in the publisher is a good consistent way to get the timestamp required.

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


Google Cloud