PREV
Chapter 2 Building and Operationalizing Storage Syster

NEXT
pter 4 Designing a Data Processing Solution

Chapter 3
Designing Data Pipelines

Google Cloud Professional Data Engineer Exam objectives covered in this chapter include the following:

1. **Designing data processing systems**
1. **✓ 1.2 Designing data pipelines. Considerations include:**
1. Data publishing and visualization (e.g., BigQuery)
2. Batch and streaming data (e.g., Cloud Dataflow, Cloud Dataproc, Apache Beam, Apache Spark and Hadoop ecosystem, Cloud Pub/Sub, Apache Kafka)
3. Online (interactive) vs. batch predictions
4. Job automation and orchestration (e.g., Cloud Composer)
2. **Building and operationalizing data processing systems**
1. **✓ 2.2 Building and operationalizing pipelines. Considerations include:**
1. Data cleansing
2. Batch and streaming
3. Transformation
4. Data acquisition and import
5. Integrating with new data sources

Data pipelines are sequences of operations that copy, transform, load, and analyze data. There are common high-level design patterns that you see repeatedly in batch, streaming, and machine learning pipelines. In this chapter, you will review those high-level design patterns, along with some variations on those patterns. You will also review how GCP services like Cloud Dataflow, Cloud Dataproc, Cloud Pub/Sub, and Cloud Composer are used to implement data pipelines. We'll also look at migrating data pipelines from an on-premises Hadoop cluster to GCP.

Overview of Data Pipelines

A data pipeline is an abstract concept that captures the idea that data flows from one stage of processing to another. Data pipelines are modeled as *directed acyclic graphs (DAGs)*. A *graph* is a set of nodes linked by edges. A *directed graph* has edges that flow from one node to another. Figure 3.1 shows a simple three-node graph with directed edges indicating that the flow in the graph moves from Node A to Node B and then to Node C.

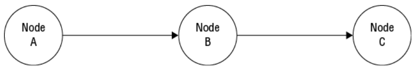


Figure 3.1 A simple directed graph

Sometimes, graphs have edges that loop back to a previous node or to the node that is the origin of the edge. Figure 3.2 shows a graph with an edge that loops from Node B back to Node A and an edge from Node C to itself. Graphs with these kinds of looping back edges are known as *cyclic graphs*, and the loops are cycles. Cycles are not allowed in data pipelines, and for that reason the graphs that model data pipelines are directed acyclic graphs.

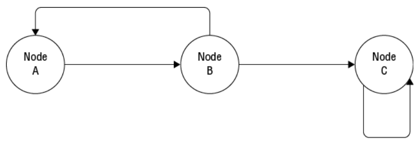


Figure 3.2 A simple cyclic graph

DATA PIPELINE STAGES

The nodes in a data pipeline DAG represent processing stages in the pipeline, and edges represent the flow of data. The four types of stages in a data pipeline are as follows:

- Ingestion
- Transformation
- Storage
- Analysis

Data pipelines may have multiple nodes in each stage. For example, a data warehouse that extracts data from three different sources would have three ingestion nodes. Not all pipelines have all stages. A pipeline may ingest audit log messages, transform them, and write them to a Cloud Storage file but not analyze them. It is possible that most of those log messages will never be viewed, but they must be stored in case they are needed. Log messages that are written to storage without any reformatting or other processing would not need a transformation stage. These examples are outliers, though. In most cases, data pipelines have one or more types of each of the four stages.

Ingestion

Ingestion (see Figure 3.3) is the process of bringing data into the GCP environment. This can occur in either batch or streaming mode.

In batch mode, data sets made up of one or more files are copied to GCP. Often these files will be copied to Cloud Storage first. There are several ways to get data into Cloud Storage, including `gsutil` copying, Transfer Service, and Transfer Appliance.

Streaming ingestion receives data in increments, typically a single record or small batches of records, that continuously flow into an ingestion endpoint, typically a Cloud Pub/Sub topic.

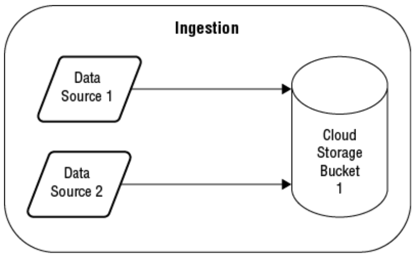


Figure 3.3 An example ingestion stage of a data pipeline

Transformation

Transformation is the process of mapping data from the structure used in the source system to the structure used in the storage and analysis stages of the data pipeline. There are many kinds of transformations, including the following:

- Converting data types, such as converting a text representation of a date to a datetime data type
- Substituting missing data with default or imputed values
- Aggregating data; for example, averaging all CPU utilization metrics for an instance over the course of one minute
- Filtering records that violate business logic rules, such as an audit log transaction with a date in the future
- Augmenting data by joining records from distinct sources, such as joining data from an employee table with data from a sales table that includes the employee identifier of the person who made the sale
- Dropping columns or attributes from a dataset when they will not be needed
- Adding columns or attributes derived from input data; for example, the average of the previous three reported sales prices of a stock might be added to a row of data about the latest price for that stock

In GCP, Cloud Dataflow and Cloud Dataproc are often used for transformation stages of both batch and streaming data. *Cloud Dataprep* is used for interactive review and preparation of data for analysis. Cloud Datafusion can be used for the same purpose, and it is more popular with enterprise customers (see Figure 3.4).

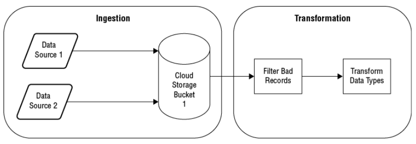


Figure 3.4 Data pipeline with transformations

Storage

After data is ingested and transformed, it is often stored. Chapter 2, "Building and Operationalizing Storage Systems," describes GCP storage systems in detail, but key points related to data pipelines will be reviewed here as well.

Cloud Storage can be used as both the staging area for storing data immediately after ingestion and also as a long-term store for transformed data. BigQuery can treat Cloud Storage data as external tables and query them. Cloud Dataproc can use Cloud Storage as HDFS-compatible storage.

BigQuery is an analytical database that uses a columnar storage model that is highly efficient for data warehousing and analytic use cases.

Bigtable is a low-latency, wide-column NoSQL database used for time-series, IoT, and other high-volume write applications. Bigtable also supports the HBase API, making it a good storage option when migrating an on-premises HBase database on Hadoop (see Figure 3.5).

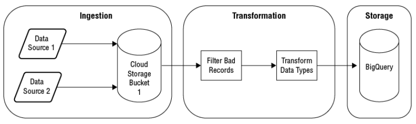


Figure 3.5 Example pipeline DAG with storage

Analysis

Analysis can take on several forms, from simple SQL querying and report generation to machine learning model training and data science analysis.

Data in BigQuery, for example, is analyzed using SQL. *BigQuery ML* is a feature of the product that allows SQL developers to build machine learning models in BigQuery using SQL.

Data Studio is a GCP service used for interactive reporting tool for building reports and exploring data that is structured as dimensional models. Cloud Datalab is an interactive workbook based on the open source Jupyter Notebooks. Datalab is used for data exploration, machine learning, data science, and visualization.

Large-scale machine learning models can be built using Spark machine learning libraries running on Cloud Dataproc while accessing data in Bigtable using the HBase interface (see Figure 3.6).

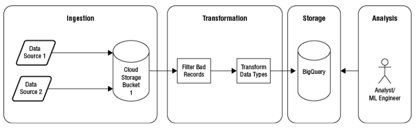


Figure 3.6 Complete data pipeline from ingestion to analysis

The basic four-stage data pipeline pattern can take on more specific characteristics for different kinds of pipelines.

TYPES OF DATA PIPELINES

The structure and function of data pipelines will vary according to the use case to which they are applied, but three common types of pipelines are as follows:

- Data warehousing pipelines
- Stream processing pipelines
- Machine learning pipeline

Let's take a look at each in more detail.

Data Warehousing Pipelines

Data warehouses are databases for storing data from multiple data sources, typically organized in a dimensional data model. Dimensional data models are denormalized; that is, they do not adhere to the rules of normalization used in transaction processing systems. This is done intentionally because the purpose of a data warehouse is to answer analytic queries efficiently, and highly normalized data models can require complex joins and significant amounts of I/O operations. Denormalized dimensional models keep related data together in a minimal number of tables so that few joins are required.

Collecting and restructuring data from online transaction processing systems is often a multistep process. Some common patterns in data warehousing pipelines are as follows:

- Extraction, transformation, and load (ETL)
- Extraction, load, and transformation (ELT)

- Extraction and load
- Change data capture

These are often batch processing pipelines, but they can have some characteristics of streaming pipelines, especially in the case of change data capture.

Extract, Transformation, and Load

Extract, transformation, and load (ETL) pipelines begin with extracting data from one or more data sources. When multiple data sources are used, the extraction processes need to be coordinated. This is because extractions are often time based, so it is important that the extracted data cover the same time period. For example, an extraction process may run once an hour and extract data inserted or modified in the previous hour.

Consider an inventory data warehouse that extracts data once an hour from a database that tracks the number of products in each of the company's storage facilities. Products are coded using stock keeping unit (SKU) codes. A product database maintains the details on each product, such as description, suppliers, and unit costs. A data warehouse would need to extract data from the inventory database for the level of inventory information and from the products database for description information. If a new product is added to the product database and stocked in the warehouse, the data warehouse would need up-to-date data from both source databases; otherwise, there could be inventory data with no corresponding description data about the product.

In an ETL pipeline, data is transformed in the pipeline before being stored in a database. In the past, data warehouse developers may have used custom scripts to transform the data or a specialized ETL tool that allowed developers to specify transformation steps using a graphical user interface (GUI). This can work well in cases where the transformation code is already captured in scripts or when data analysts with limited programming experience are building transformations. It does, however, sometimes require developers to learn a tool in addition to SQL for manipulating data.

In GCP, transformations can be done using Cloud Dataproc or Cloud Dataflow. With Cloud Dataproc, transformations can be written in a Spark- or Hadoop-supported language. Spark uses an in-memory distributed data model for data manipulation and analysis. Spark programs can be written in Java, Scala, Python, R, and SQL. When using Cloud Dataproc, your transformations are written according to Hadoop's map reduce model or Spark's distributed tabular data structure. In addition to supporting Java for transformations, Hadoop provides *Pig*, a high-level language for data manipulation. Pig programs compile into map reduce programs that run on Hadoop.

When using Cloud Dataflow, you write transformations using the Apache Beam model, which provides a unified batch and stream processing model. Apache Beam is modeled as a pipeline and has explicit support for pipeline constructs, including the following:

- **Pipelines:** An encapsulation of the end-to-end data processing task that executes the data operations
- **PCollection:** A distributed dataset
- **PTransform:** An operation on data, such as grouping by, flattening, and partitioning of data
- Apache Beam programs are written in Java and Python.

For writing data to a database, Cloud Dataflow uses connectors, including Bigtable, Cloud Spanner, and BigQuery.

Cloud Dataproc is a good choice for implementing ETL processes if you are migrating existing Hadoop or Spark programs. Cloud Dataflow is the recommended tool for developing new ETL processes. Cloud Dataflow is serverless, so there is no cluster to manage and the processing model is based on data pipelines. Cloud Dataproc's Hadoop and Spark platforms are designed on big data analytics processing, so they can be used for transformations, but Cloud Dataflow model is based on data pipelines.

Extract, Load, and Transformation

Extract, load, and transformation (ELT) processes are slightly different from ETL processes. In an ELT process, data is loaded into a database before transforming the data. This process has some advantages over ETL.

When data is loaded before transformation, the database will contain the original data as extracted. This enables data warehouse developers to query the data using SQL, which can be useful for performing basic data quality checks and collecting statistics on characteristics such as the number of rows with missing data.

A second advantage is that developers can use SQL for transformation operations. This is especially helpful if developers are well versed in SQL but do not have programming experience. Developers would also be able to use SQL tools in addition to writing SQL from scratch.

Extraction and Load

Extraction and load procedures do not transform data. This type of process is appropriate when data does not require changes from the source format. Log data, for example, may be extracted and loaded without transformation. Dimensional data extracted from a data warehouse for loading into a smaller data mart also may not need transformation.

Change Data Capture

In a *change data capture* approach, each change in a source system is captured and recorded in a data store. This is helpful in cases where it is important to know all changes over time and not just the state of the database at the time of data extraction.

For example, an inventory database tracking the number of units of products available in a warehouse may have the following changes:

- Product A's inventory is set to 500 in Warehouse 1.
- Product B's inventory is set to 300 in Warehouse 1.
- Product A's inventory is reduced by 250 in Warehouse 1.
- Product A's inventory is reduced by 100 in Warehouse 1.

After these changes, Product A's inventory level is 150 and Product B's is 300. If you need to know only the final inventory level, then an ETL or ELT process is sufficient; however, if you need to know all the changes in inventory levels of products, then a change data capture approach is better.

Data warehousing pipelines are often batch oriented and run on regular schedules. When data needs to be processed continuously, a stream processing pipeline is required.

Stream Processing Pipelines

Streams are unending, continuous sources of data. Streams can be generated from many sources. Here are some examples:

- IoT devices collecting weather condition data may stream temperature, humidity, and pressure data from various locations every five minutes.
- Heart monitors in a hospital may stream cardiovascular metrics every second the monitor is in use.
- An agent collecting application performance metrics might send data every 15 seconds to a stream processing system for anomaly detection and alerting.

In the case of weather data, there may not be a need to process and analyze the data as soon as possible. This would be the case if the data is collected for long-term studies of climate change. The heart monitor example, however, may need to be analyzed as soon as possible in case the data indicates a medical condition that needs attention. Similarly, if an application has stopped reporting, then a DevOps engineer may want to be alerted immediately. In these cases, streamed data should be analyzed as soon as possible.

The kinds of analysis that you perform on streaming data is not all that different from batch processing analysis. It includes aggregating data, looking for anomalous patterns, and visualizing data in charts and graphs. The difference is in the way that you group the data. In a batch processing environment, all the data you need is available at once. This is not the case with streaming data.

Streaming data is continuous, so you need to pick subsets of the stream to work on at any one time. For example, you might average all the temperature readings from a sensor for the last hour to get an hourly average. You might calculate that average once an hour at the top of the hour. This would allow you to track how temperatures varied from hour to hour. You could also calculate the moving average of the last hours' worth of data every five minutes. This would give you a finer-grained view of how the temperatures vary. This is just one example of the different things that you should take into account when working with streaming data.

When building data pipelines for streaming data, consider several factors, including

- Event time and processing time
- Sliding and tumbling windows
- Late-arriving data and watermarks
- Missing data

There are various ways to process streaming data, and the configuration of your processing pipeline depends on these various factors.

Event Time and Processing Time

Data in time-series streams is ordered by time. If a set of data A arrives before data B, then presumably the event described by A occurred before the event described by B. There is a subtle but important issue implied in the previous sentence, which is that you are actually dealing with two points in time in stream processing:

- *Event time* is the time that something occurred at the place where the data is generated.
- *Processing time* is the time that data arrives at the endpoint where data is ingested. Processing time could be defined as some other point in the data pipeline, such as the time that transformation starts.

When working with streaming data, it is important to use one of these times consistently for ordering the streams. Event time is a good choice when the processing logic assumes that the data in the stream is in the same order as it was generated.

If data arrives at the ingestion endpoint in the same order that it was sent, then the ordering of the two times would be the same. If network congestion or some other issue causes delays, then data that was generated earlier could arrive later.

Sliding and Tumbling Windows

A *window* is a set of consecutive data points in a stream. Windows have a fixed width and a way of advancing. Windows that advance by a number of data points less than the width of the window are called *sliding windows*; windows that advance by the length of the window are *tumbling windows*.

Let's look at the example stream at the top of Figure 3.7. The stream has nine data points: 7, 9, 10, 12, 8, 9, 13, 8, and 4. The first three data points are shown in a window that is three data points wide (see Figure 3.7).

If you move the window forward one position in the stream, the window would include 9, 10, and 12. Moving the window down another position leaves the window containing 10, 12, and 8. This is how sliding windows advance through a stream of data.

If you move the window forward along the stream by the width of the window—in this case, three positions—you would have the first window include 7, 9, and 10. Advancing the window again by three positions would leave the window including 12, 8, and 9. This is how a tumbling window advances through a stream of data.

Sliding windows are used when you want to show how an aggregate—such as the average of the last three values—change over time, and you want to update that stream of averages each time a new value arrives in the stream.

Tumbling windows are used when you want to aggregate data over a fixed period of time, for example, for the last one minute. In that case, at the top of the minute you would calculate the average for the data that arrived in the last minute. The next time that you would calculate an average is at the top of the next minute. At that time, you would average the data that had arrived since the last average was calculated. There is no overlap in this case.

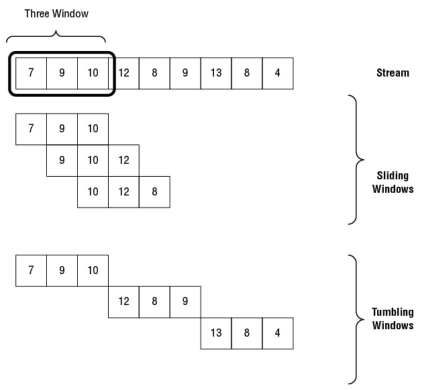


Figure 3.7 A stream with sliding and tumbling three window

Late Arriving and Watermarks

When working with streaming data, especially time-series data, you have to decide how long you will wait for data to arrive. If you expect data to arrive about once a minute, you might be willing to wait up to three or four minutes for a late-arriving piece of data. For example, if your stream of data is from a medical device and you want to have a record of all data points whenever possible, you may be willing to wait longer than if you used the data to update a streaming chart. In that case, you may decide that after two minutes you would rather show a moving average of the last three data points instead of waiting longer.

When you wait for late-arriving data, you will have to maintain a buffer to accumulate data before performing stream processing operations. Consider a use case in which you take a stream as input and output a stream of averages for the last three minutes. If you have received data for one minute ago and three minutes ago but not two minutes ago, you will have to keep the two data points that have arrived in a buffer until you receive the two-minute data point, or until you wait as long as possible and then treat the two-minute data point as a piece of missing data.

When working with streams, you need to be able to assume at some point that no more data generated earlier than some specified time will arrive. For example, you may decide that any data that arrives 10 minutes late will not be ingested into the stream. To help stream processing applications, you can use the concept of a *watermark*, which is basically a timestamp indicating that no data older than that timestamp will ever appear in the stream.

Up until know, you could think about streams as windows of data that are finite and complete—like a dataset that you process in batch mode. In that case, windows are just small batches of data, but the reality is more complicated. Watermarks indicate a boundary on the lateness of data. If a data point arrives so late that its event time occurred before the watermark's timestamp, it is ignored by the stream. That does not mean that you should ignore it completely, though. A more accurate reflection of the state of the system would include that late-arriving data.

You can accommodate late-arriving data and improve accuracy by modifying the way that you ingest, transform, and store data.

Hot Path and Cold Path Ingestion

We have been considering a streaming-only ingestion process. This is sometimes called a *hot path ingestion*. It reflects the latest data available and makes it available as soon as possible. You improve the timeliness of reporting data at the potential risk of a loss of accuracy.

There are many use cases where this tradeoff is acceptable. For example, an online retailer having a flash sale would want to know sales figures in real time, even if they might be slightly off. Sales professionals running the flash sale need that data to adjust the parameters of the sale, and approximate, but not necessarily accurate, data meets their needs.

Accountants in the Finance department of that same online retailer have a different set of requirements. They do not need data immediately, but they do need complete data. In this case, even if the data was too late to be used in a stream processing pipeline, it could still be written to a database where it could be included in reports along with data that was included in the streaming dataset. This path from ingestion to persistent storage is called *cold path ingestion*. See Figure 3.8 for an example.

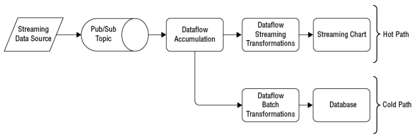


Figure 3.8 Data pipeline with both a hot path and a cold path

Stream and batch processing pipelines, like those described here, can meet the requirements of many use cases. The combination of the two can also meet the needs of machine learning pipelines, but there are some machine-learning specific steps in those pipelines, so it's worth looking into those details.

Machine Learning Pipelines

Machine learning pipelines typically include some of the same steps as data warehousing pipelines but have some specific stages as well. A typical machine learning pipeline includes

- Data ingestion
- Data preprocessing, which is called *transformation* in data warehousing pipelines
- Feature engineering, which is another form of transformation
- Model training and evaluation
- Deployment

Data ingestion uses the same tools and services as data warehousing and streaming data pipelines. Cloud Storage is used for batch storage of datasets, whereas Cloud Pub/Sub can be used for the ingestion of streaming data.

Cloud Dataproc and Cloud Dataflow can be used for programmatic data preprocessing. In addition, Cloud Dataprep may be used for more ad hoc and interactive preparation of data. Cloud Dataprep is especially helpful when working with new datasets to understand the data in terms of the distribution of values of various attributes, the frequency of missing data, and for spotting other data quality problems. Ideally, once you have a good understanding of the kinds of preprocessing required by data from a particular source, you will encode that logic into a Cloud Dataflow process.

Feature engineering is a machine learning practice in which new attributes are introduced into a dataset. The new attributes are derived from one or more existing attributes. Sometimes, the new attributes are relatively simple to calculate. In the case of an IoT data stream sending weather data, you might want to calculate the ratio of temperature to pressure and of humidity to pressure and include those two ratios as new attributes or features. In other cases, the engineered features may be more complicated, like performing a fast Fourier transformation to map a stream of data into the frequency domain.

GCP Pipeline Components

GCP has several services that are commonly used components of pipelines, including the following:

- Cloud Pub/Sub
- Cloud Dataflow
- Cloud Dataproc
- Cloud Composer

Among these services, developers have a number of different processing model options.

CLOUD PUB/SUB

Cloud Pub/Sub is a real-time messaging service that supports both push and pull subscription models. It is a managed service, and it requires no provisioning of servers or clusters. Cloud Pub/Sub will automatically scale and partition load as needed.

Working with Messaging Queues

Messaging queues are used in distributed systems to decouple services in a pipeline. This allows one service to produce more output than the consuming service can process without adversely affecting the consuming service. This is especially helpful when one process is subject to spikes in workload.

When working with Cloud Pub/Sub, you create a topic, which is a logical structure for organizing your messages. Once a topic is created, you create a subscription to the topic and then publish messages to the topic. Subscriptions are a logical structure for organizing the reception of messages by consuming processes.

When messaging queues receive data in a a message, it is considered a publication event. Upon publication, *push subscriptions* deliver the message to an endpoint. Some common types of endpoints are Cloud Functions, App Engine, and Cloud Run services. *Pull subscriptions* are used when you want the consuming application to control when messages are retrieved from a topic. Specifically, with pull subscriptions you send a request asking for *N* messages, and Cloud Pub/Sub responds with the next *N* or fewer messages.

Topics can be created in the console or the command line. The only required parameter is a topic ID, but you can also specify whether the topic should use a Google-managed key or a customer-managed key for encryption. The command to create a topic is `gcloud pubsub topics create`; for example:

```
gcloud pubsub topics create pde-topic-1
```

creates a topic called `pde-exam-topic-1`. Subscriptions can be created in the console, or with the `gcloud pubsub subscriptions create` command and specifying a topic ID and a subscription ID; for example:

```
gcloud pubsub subscriptions create --topic pde-topic-1 pde-subscripton-
```

Messages can be written to topics using APIs and client libraries as well as a `gcloud` command. Processes that write messages are called *publishers* or *producers*; services that read messages are called *subscribers* or *consumers*.

Client libraries are available for a number of languages, including C#, Go, Java, Node.js, PHP, Python, and Ruby. (Additional languages may have been added by the time you read this.) Cloud Pub/Sub also supports REST APIs and gRPC APIs. The command-line tool is useful for testing your ability to publish and consume messages from a topic. For example, to publish a message with a string, you could issue the following:

```
gcloud pubsub topics publish pde-topic-1 --message "data engineer exam"
```

This command inserts or publishes the message to the `pde-topic-1` topic, and the message is available to be read through a subscription. By default, when a topic is created, it is done so as a pull subscription. The `gcloud` command to read a message from a topic is `gcloud pubsub subscriptions`; for example:

```
gcloud pubsub subscriptions pull --auto-ack pde-subscripton-1
```

The `auto-ack` flag indicates that the message should be acknowledged automatically. Acknowledgments indicate to the subscription that the message has been read and processed so that it can be removed from the topic. When a message is sent but before it is acknowledged, the message is considered outstanding. While a message is outstanding to that subscriber, it will not be delivered to another subscriber on the same subscription. If the message is outstanding for a period of time greater than the time allowed for a subscriber to acknowledge the message, then it is no longer considered outstanding and will be delivered to another subscriber. The time allowed for a subscriber to acknowledge a message can be specified in the `subscription` command using the `ackDeadline` parameter. Messages can stay in a topic for up to seven days.

Pub/sub makes no guarantees that the order of message reception is the same as the publish order. In addition, messages can be delivered more than once. For these reasons, your processing logic should be idempotent; that is, the logic could be applied multiple times and still provide the same output. A trivial example is adding 0 to a number. No matter how many times you add 0 to a number, the result is always the same.

For a more interesting example, consider a process that receives a stream of messages that have identifiers and a count of the number of a particular type of event that occurred in a system in the past minute. The time is represented as the date followed by the number of minutes past midnight, so each minute of each day has a unique identifier. The stream processing system needs to keep a running cumulative total number of events for each day. If the process performing the aggregation simply added the count of each message it received to the running total, then duplicate messages would have their counts added in multiple times. This is not an idempotent operation. However, if instead the process keeps a list of all minutes for which data is received and it only adds in the counts of data points not received before, then the operation would be idempotent.

If you need guaranteed exactly once processing, use Cloud Dataflow PubsubIO, which de-duplicates based on a message ID. Cloud Dataflow can also be used to ensure that messages are processed in order.

Open Source Alternative: Kafka

One of the advantages of Cloud Pub/Sub is that it is a managed service. An open source alternative is *Apache Kafka*. Kafka is used to publish and subscribe to streams of messages and to reliably store messages in a fault-tolerant way.

Kafka runs in a cluster of one or more servers. You interact with Kafka programmatically by using one of the four APIs:

- Producer API
- Consumer API
- Streams API
- Connector API

The Producer API is used to publish messages, which are called records in Kafka parlance. The Consumer API supports subscriptions on topics. The Streams API supports stream processing operations that transform a stream of data into an output stream. The Connector API is used to enable Kafka to work with existing applications.

If you are migrating an on-premises service that uses Kafka and you want to replace self-managed Kafka with a managed service, then the Cloud Pub/Sub would meet that need. If you plan to continue to use Kafka, you can link Cloud Pub/Sub and Kafka using the CloudPubSubConnector, which is a bridge between the two messaging systems using Kafka Connect. CloudPubSubConnector is an open source tool maintained by the Cloud Pub/Sub team and is available here:

`https://github.com/GoogleCloudPlatform/pubsub/tree/master/kafka-connector`

CLOUD DATAFLOW

Cloud Dataflow is a managed stream and batch processing service. It is a core component for building pipelines that collect, transform, and output data. In the past, developers would typically create a batch or stream processing pipeline—for example, the hot path and a separate batch processing pipeline; that is, the cold path. Cloud Dataflow pipelines are written using the Apache Beam API, which is a model for combined stream and batch processing. Apache Beam incorporates *Beam runners* in the data pipeline; the Cloud Dataflow runner is commonly used in GCP. Apache Flink is another commonly used Beam runner.

Cloud Dataflow does not require you to configure instances or clusters—it is a no-ops service. Cloud Dataflow pipelines are run within a region. It directly integrates with Cloud Pub/Sub, BigQuery, and the Cloud ML Engine. Cloud Dataflow integrates with Bigtable and Apache Kafka.

Much of your work with Cloud Dataflow is coding transformations in one of the languages supported by Apache Beam, which are currently Java and Python. For the purpose of the exam, it is important to understand Cloud Dataflow concepts.

Cloud Dataflow Concepts

Cloud Dataflow, and the Apache Beam model, are designed around several key concepts:

- Pipelines
- PCollection
- Transforms
- ParDo
- Pipeline I/O
- Aggregation
- User-defined functions
- Runner
- Triggers

Windowing concepts and watermarks are also important and were described earlier in the chapter.

Pipelines in Cloud Dataflow are, as you would expect, a series of computations applied to data that comes from a source. Each computation emits the results of computations, which become the input for the next computation in the pipeline. Pipelines represent a job that can be run repeatedly.

The *PCollection* abstraction is a dataset, which is the data used when a pipeline job is run. In the case of batch processing, the PCollection contains a fixed set of data. In the case of streaming data, the PCollection is unbounded.

Transforms are operations that map input data to some output data. Transforms operate on one or more PCollections as input and can produce one or more output PCollections. The operations can be mathematical calculations, data type conversions, and data grouping steps, as well as performing read and write operations.

ParDo is a parallel processing operation that runs a user-specified function on each element in a PCollection. ParDo transforms data in parallel. ParDo receives input data from a main PCollection but may also receive additional inputs from other PCollections by using a *side input*. Side inputs can be used to perform joins. Similarly, while a ParDo produces a main output PCollection, additional collections can be output using a *side output*. Side outputs are especially useful when you want to have additional processing paths. For example, a side output could be used for data that does not pass some validation check.

Pipeline I/Os are transforms for reading data into a pipeline from a source and writing data to a sink.

Aggregation is the process of computing a result from multiple input values. Aggregation can be simple, like counting the number of messages arriving in a one-minute period or averaging the values of metrics received over the past hour.

User-defined functions (UDF) are user-specified code for performing some operation, typically using a ParDo.

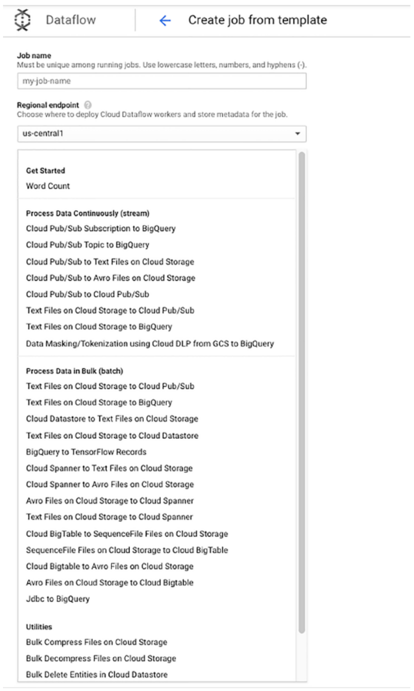
Runners are software that executes pipelines as jobs.

Triggers are functions that determine when to emit an aggregated result. In batch processing jobs, results are emitted when all the data has been processed. When operating on a stream, you have to specify a window over the stream to define a bounded subset, which is done by configuring the window.

Jobs and Templates

A *job* is an executing pipeline in Cloud Dataflow. There are two ways to execute jobs: the traditional method and the template method.

With the traditional method, developers create a pipeline in a development environment and run the job from that environment. The template method separates development from staging and execution. With the template method, developers still create pipelines in a development environment, but they also create a template, which is a configured job specification. The specification can have parameters that are specified when a user runs the template. Google provides a number of templates, and you can create your own as well. See Figure 3.9 for examples of templates provided by Google.



Job name
Must be unique among running jobs. Use lowercase letters, numbers, and hyphens (-).
my-job-name

Regional endpoint
Choose where to deploy Cloud Dataflow workers and store metadata for the job.
us-central1

Get Started
Word Count

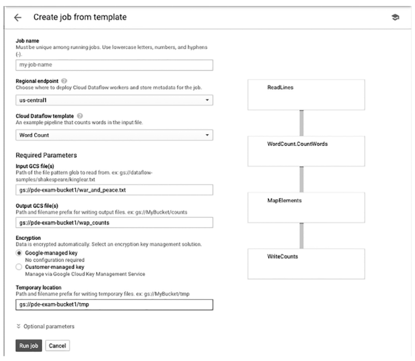
Process Data Continuously (stream)
Cloud Pub/Sub Subscription to BigQuery
Cloud Pub/Sub Topic to BigQuery
Cloud Pub/Sub to Text Files on Cloud Storage
Cloud Pub/Sub to Avro Files on Cloud Storage
Cloud Pub/Sub to Cloud Pub/Sub
Text Files on Cloud Storage to Cloud Pub/Sub
Text Files on Cloud Storage to BigQuery
Data Masking/Tokenization using Cloud DLP from GCS to BigQuery

Process Data in Bulk (batch)
Text Files on Cloud Storage to Cloud Pub/Sub
Text Files on Cloud Storage to BigQuery
Cloud Datastore to Text Files on Cloud Storage
Text Files on Cloud Storage to Cloud Datastore
BigQuery to TensorFlow Records
Cloud Spanner to Text Files on Cloud Storage
Cloud Spanner to Avro Files on Cloud Storage
Avro Files on Cloud Storage to Cloud Spanner
Text Files on Cloud Storage to Cloud Spanner
Cloud BigTable to SequenceFile Files on Cloud Storage
SequenceFile Files on Cloud Storage to Cloud BigTable
Cloud Bigtable to Avro Files on Cloud Storage
Avro Files on Cloud Storage to Cloud Bigtable
Jdbc to BigQuery

Utilities
Bulk Compress Files on Cloud Storage
Bulk Decompress Files on Cloud Storage
Bulk Delete Entities in Cloud Datastore

Figure 3.9 Creating a Cloud Dataflow job in the console using a template

After selecting a template, you can specify parameters, such as source and sink specifications. Figure 3.10 shows the parameters and transformations used in the Word Count Template.



Create job from template

Job name
Must be unique among running jobs. Use lowercase letters, numbers, and hyphens (-).
my-job-name

Regional endpoint
Choose where to deploy Cloud Dataflow workers and store metadata for the job.
us-central1

Cloud Dataflow template
An example pipeline that counts words in the input file.
Word Count

Required Parameters

Input GCS file(s)
Path of the file pattern glob to read from. ex: gs://dataflow-samples/shakespeare/kinglear.txt
gs://pde-exam-bucket1/war_and_peace.txt

Output GCS file(s)
Path and filename prefix for writing output files. ex: gs://MyBucket/counts
gs://pde-exam-bucket1/wap_counts

Encryption
Data is encrypted automatically. Select an encryption key management solution.
[✓ Google-managed key] No configuration required
Customer-managed key — Manage via Google Cloud Key Management Service

Temporary location
Path and filename prefix for writing temporary files. ex: gs://MyBucket/tmp
gs://pde-exam-bucket1/tmp

Figure 3.10 Specifying parameters for the Word Count Template

Jobs can be run from the command line and through the use of APIs as well. For example, you could use the `gcloud dataflow jobs run command` to start a job. An example of a complete `job run` command looks like this:

```
gcloud dataflow jobs run pde-job-1 \
--gcs-location gs://pde-exam-cert/templates/word-count-template
```

This command creates a job named `pde-job-1` using a template file called `word-count-template` located in the `pde-exam-cert/templates` bucket.

CLOUD DATAPROC

Cloud Dataproc is a managed Hadoop and Spark service where a preconfigured cluster can be created with one command line or console operation. Cloud Dataproc makes it easy to migrate from on-premises Hadoop clusters to GCP. A typical Cloud Dataproc cluster is configured with commonly used components of the Hadoop ecosystem, including the following:

- **Hadoop:** This is an open source, big data platform that runs distributed processing jobs using the map reduce model. Hadoop writes the results of intermediate operations to disk.
- **Spark:** This is another open source, big data platform that runs distributed applications, but in memory instead of writing the results of map reduce operations to disk.
- **Pig:** This is a compiler that produces map reduce programs from a high-level language for expressing operations on data.
- **Hive:** This is a data warehouse service built on Hadoop.

When working with Cloud Dataproc, you must know how to manage data storage, configure a cluster, and submit jobs.

Cloud Dataproc allows the possibility to use “ephemeral” clusters, where a large cluster can be created to run a task and then destroyed once the task is over in order to save costs.

Managing Data in Cloud Dataproc

When running Hadoop on premises, you store data on the Hadoop cluster. The cluster uses the Hadoop Distributed File System (HDFS), which is part of Hadoop. This is a sound approach when you have dedicated hardware implementing your Hadoop cluster. In the cloud, instead of having a single, long-running Hadoop cluster, you typically start a Cloud Dataproc cluster for each job and shut down the cluster when the job completes. This is a better option than maintaining a long-running cluster in the cloud because Cloud Dataproc clusters start in about 90 seconds, so there is little disincentive to shutting down clusters when idle.

Since you tend to use ephemeral clusters when working with Cloud Dataproc, if you wanted to use HDFS here you would have to copy your data from Cloud Storage each time you started a cluster. A better approach is to use Cloud Storage as the data store. This saves the time and the cost of having to copy data to the cluster.

Configuring a Cloud Dataproc Cluster

Cloud Dataproc clusters consist of two types of nodes: master nodes and worker nodes. The master node is responsible for distributing and managing workload distribution. The master node runs a system called YARN, which stands for Yet Another Resource Negotiator. YARN uses data about the workload and resource availability on each worker node to determine where to run jobs.

When creating a cluster, you will specify a number of configuration parameters, including a cluster name and the region and zone to create the cluster. Clusters can run on a single node, which is a good option for development. They can also run with one master and some number of worker nodes. This is known as *standard mode. High availability mode* uses three master nodes and some number of workers. You can specify a machine configuration for both the master and worker nodes, and they do not have to be the same. Worker nodes can include some preemptible machines, although HDFS storage does not run on preemptible nodes, which is another reason to use Cloud Storage instead of HDFS.

Initialization scripts can be run when the cluster is created by specifying script files located in a Cloud Storage bucket.

Clusters can be created using the console or the command line. The following `gcloud dataproc clusters create` command, for example, will create a cluster with the default configuration:

```
gcloud dataproc clusters create pde-cluster-1
```

Once a cluster is created, it can be scaled up or down. Only the number of worker nodes can change—master nodes are fixed. You can manually scale the size of a cluster using the `gcloud dataproc clusters update` command, as follows:

```
gcloud dataproc clusters update pde-cluster-1 \
--num-workers 10 \
--num-preemptible-workers 20
```

This snippet will scale the cluster to run 10 regular workers and 20 preemptible worker nodes.

Cloud Dataproc also supports autoscaling by creating an autoscaling policy for a cluster. An autoscaling policy is specified in a YAML file and includes parameters such as

- `maxInstances`
- `scaleUpFactor`
- `scaleDownFactor`
- `cooldownPeriod`

Autoscaling works by checking Hadoop YARN metrics about memory use at the end of each `cooldownPeriod`. The number of nodes added or removed is determined by the current number and the scaling factor.

Submitting a Job

Jobs are submitted to Cloud Dataproc using an API, a `gcloud` command, or in the console. The `gcloud dataproc jobs submit` command runs a job from the command line. Here is an example command:

```
gcloud dataproc jobs submit pyspark \
--cluster pde-cluster-1 \
--region us-west-1  \
gs://pde-exam-cert/dataproc-scripts/analysis.py
```

This command submits a PySpark job to the `pde-cluster-1` cluster in the `us-west-1` region and runs the program in the `analysis.py` file in the `pde-exam-cert/dataproc-scripts` Cloud Storage bucket.

In general, it is a good practice to keep clusters in the same region as the Cloud Storage buckets that will be used for storing data. You can expect to see better I/O performance when you configure nodes with larger persistent disks and that use SSDs over HDDs.

CLOUD COMPOSER

Cloud Composer is a managed service implementing Apache Airflow, which is used for scheduling and managing workflows. As pipelines become more complex and have to be resilient when errors occur, it becomes more important to have a framework for managing workflows so that you are not reinventing code for handling errors and other exceptional cases.

Cloud Composer automates the scheduling and monitoring of workflows. Workflows are defined using Python and are directed acyclic graphs. Cloud Composer has built-in integration with BigQuery, Cloud Dataflow, Cloud Dataproc, Cloud Datastore, Cloud Storage, Cloud Pub/Sub, and AI Platform.

Before you can run workflows with Cloud Composer, you will need to create an environment in GCP. Environments run on the Google Kubernetes Engine, so you will have to specify a number of nodes, location, machine type, disk size, and other node and network configuration parameters. You will need to create a Cloud Storage bucket as well.

Migrating Hadoop and Spark to GCP

When you are migrating Hadoop and Spark clusters to GCP, there are a few things for which you will need to plan:

- Migrating data
- Migrating jobs
- Migrating HBase to Bigtable

You may also have to shift your perspective on how you use clusters. On-premises clusters are typically large persistent clusters that run multiple jobs. They can be complicated to configure and manage. In GCP, it is a best practice to use an ephemeral cluster for each job. This approach leads to less complicated configurations and reduced costs, since you are not storing persistent data on the cluster and not running the cluster for extended periods of time.

Hadoop and Spark migrations can happen incrementally, especially since you will be using ephemeral clusters configured for specific jobs. The first step is to migrate some data to Cloud Storage. Then you can deploy ephemeral clusters to run jobs that use that data. It is best to start with low-risk jobs so that you can learn the details of working with Cloud Dataproc.

There may be cases where you will have to keep an on-premises cluster while migrating some jobs and data to GCP. In those cases, you will have to keep data synchronized between environments. Plan to implement workflows to keep data synchronized. You should have a way to determine which jobs and data move to the cloud and which stay on premises.

It is a good practice to migrate HBase databases to Bigtable, which provides consistent, scalable performance. When migrating to Bigtable, you will need to export HBase tables to sequence files and copy those to Cloud Storage. Next, you will have to import the sequence files using Cloud Dataflow. When the size of data to migrate is greater than 20 TB, use the Transfer Appliance. When the size is less than 20 TB and there is at least 100 Mbps of network bandwidth available, then `distcp`, a Hadoop distributed copy command, is the recommended way to copy the data. In addition, it is important to know how long it will take to transfer the data and to have a mechanism for keeping the on-premises data in sync with the data in Cloud Storage.

Exam Essentials

Understand the model of data pipelines. A data pipeline is an abstract concept that captures the idea that data flows from one stage of processing to another. Data pipelines are modeled as directed acyclic graphs (DAGs). A graph is a set of nodes linked by edges. A directed graph has edges that flow from one node to another.

Know the four stages in a data pipeline. *Ingestion* is the process of bringing data into the GCP environment. *Transformation* is the process of mapping data from the structure used in the source system to the structure used in the *storage* and *analysis* stages of the data pipeline. Cloud Storage can be used as both the staging area for storing data immediately after ingestion and also as a long-term store for transformed data. BigQuery and Cloud Storage treat data as external tables and query them. Cloud Dataproc can use Cloud Storage as HDFS-compatible storage. Analysis can take on several forms, from simple SQL querying and report generation to machine learning model training and data science analysis.

Know that the structure and function of data pipelines will vary according to the use case to which they are applied. Three common types of pipelines are data warehousing pipelines, stream processing pipelines, and machine learning pipelines.

Know the common patterns in data warehousing pipelines. Extract, transformation, and load (ETL) pipelines begin with extracting data from one or more data sources. When multiple data sources are used, the extraction processes need to be coordinated. This is because extractions are often time based, so it is important that extracts from different sources cover the same time period. Extract, load, and transformation (ELT) processes are slightly different from ETL processes. In an ELT process, data is loaded into a database before transforming the data. Extraction and load procedures do not transform data. This kind of process is appropriate when data does not require changes from the source format. In a change data capture approach, each change is a source system that is captured and recorded in a data store. This is helpful in cases where it is important to know all changes over time and not just the state of the database at the time of data extraction.

Understand the unique processing characteristics of stream processing. This includes the difference between event time and processing time, sliding and tumbling windows, late-arriving data and watermarks, and missing data. Event time is the time that something occurred at the place where the data is generated. Processing time is the time that data arrives at the endpoint where data is ingested. Sliding windows are used when you want to show how an aggregate, such as the average of the last three values, change over time, and you want to update that stream of averages each time a new value arrives in the stream. Tumbling windows are used when you want to aggregate data over a fixed period of time—for example, for the last one minute.

Know the components of a typical machine learning pipeline. This includes data ingestion, data preprocessing, feature engineering, model training and evaluation, and deployment. Data ingestion uses the same tools and services as data warehousing and streaming data pipelines. Cloud Storage is used for batch storage of datasets, whereas Cloud Pub/Sub can be used for the ingestion of streaming data. Feature engineering is a machine learning practice in which new attributes are introduced into a dataset. The new attributes are derived from one or more existing attributes.

Know that Cloud Pub/Sub is a managed message queue service. Cloud Pub/Sub is a real-time messaging service that supports both push and pull subscription models. It is a managed service, and it requires no provisioning of servers or clusters. Cloud Pub/Sub will automatically scale as needed. Messaging queues are used in distributed systems to decouple services in a pipeline. This allows one service to produce more output than the consuming service can process without adversely affecting the consuming service. This is especially helpful when one process is subject to spikes.

Know that Cloud Dataflow is a managed stream and batch processing service. Cloud Dataflow is a core component for running pipelines that collect, transform, and output data. In the past, developers would typically create a stream processing pipeline (hot path) and a separate batch processing pipeline (cold path). Cloud Dataflow is based on Apache Beam, which is a model for combined stream and batch processing. Understand these key Cloud Dataflow concepts:

- Pipelines
- PCollection
- Transforms
- ParDo
- Pipeline I/O
- Aggregation
- User-defined functions
- Runner
- Triggers

Know that Cloud Dataproc is a managed Hadoop and Spark service. Cloud Dataproc makes it easy to create and destroy ephemeral clusters. Cloud Dataproc makes it easy to migrate from on-premises Hadoop clusters to GCP. A typical Cloud Dataproc cluster is configured with commonly used components of the Hadoop ecosystem, including Hadoop, Spark, Pig, and Hive. Cloud Dataproc clusters consist of two types of nodes: master nodes and worker nodes. The master node is responsible for distributing and managing workload distribution.

Know that Cloud Composer is a managed service implementing Apache Airflow. Cloud Composer is used for scheduling and managing workflows. As pipelines become more complex and have to be resilient when errors occur, it becomes more important to have a framework for managing workflows so that you are not reinventing code for handling errors and other exceptional cases. Cloud Composer automates the scheduling and monitoring of workflows. Before you can run workflows with Cloud Composer, you will need to create an environment in GCP.

Understand what to consider when migrating from on-premises Hadoop and Spark to GCP. Factors include migrating data, migrating jobs, and migrating HBase to Bigtable. Hadoop and Spark migrations can happen incrementally, especially since you will be using ephemeral clusters configured for specific jobs. There may be cases where you will have to keep an on-premises cluster while migrating some jobs and data to GCP. In those cases, you will have to keep data synchronized between environments. It is a good practice to migrate HBase databases to Bigtable, which provides consistent, scalable performance.

Review Questions

You can find the answers in the appendix.

1. A large enterprise using GCP has recently acquired a startup that has an IoT platform. The acquiring company wants to migrate the IoT platform from an on-premises data center to GCP and wants to use Google Cloud managed services whenever possible. What GCP service would you recommend for ingesting IoT data?

   1. Cloud Storage
   2. Cloud SQL
   3. Cloud Pub/Sub
   4. BigQuery streaming inserts

2. You are designing a data pipeline to populate a sales data mart. The sponsor of the project has had quality control problems in the past and has defined a set of rules for filtering out bad data before it gets into the data mart. At what stage of the data pipeline would you implement those rules?

   1. Ingestion
   2. Transformation
   3. Storage
   4. Analysis

3. A team of data warehouse developers is migrating a set of legacy Python scripts that have been used to transform data as part of an ETL process. They would like to use a service that allows them to use Python and requires minimal administration and operations support. Which GCP service would you recommend?

   1. Cloud Dataproc
   2. Cloud Dataflow
   3. Cloud Spanner
   4. Cloud Dataprep

4. You are using Cloud Pub/Sub to buffer records from an application that generates a stream of data based on user interactions with a website. The messages are read by another service that transforms the data and sends it to a machine learning model that will use it for training. A developer has just released some new code, and you notice that messages are sent repeatedly at 10-minute intervals. What might be the cause of this problem?

1. The new code release changed the subscription ID.
2. The new code release changed the topic ID.
3. The new code disabled acknowledgments from the consumer.
4. The new code changed the subscription from pull to push.

5. It is considered a good practice to make your processing logic idempotent when consuming messages from a Cloud Pub/Sub topic. Why is that?

1. Messages may be delivered multiple times.
2. Messages may be received out of order.
3. Messages may be delivered out of order.
4. A consumer service may need to wait extended periods of time between the delivery of messages.

6. A group of IoT sensors is sending streaming data to a Cloud Pub/Sub topic. A Cloud Dataflow service pulls messages from the topic and reorders the messages sorted by event time. A message is expected from each sensor every minute. If a message is not received from a sensor, the stream processing application should use the average of the values in the last four messages. What kind of window would you use to implement the missing data logic?

1. Sliding window
2. Tumbling window
3. Extrapolation window
4. Crossover window

7. Your department is migrating some stream processing to GCP and keeping some on premises. You are tasked with designing a way to share data from on-premises pipelines that use Kafka with GPC data pipelines that use Cloud Pub/Sub. How would you do that?

1. Use CloudPubSubConnector and Kafka Connect
2. Stream data to a Cloud Storage bucket and read from there
3. Write a service to read from Kafka and write to Cloud Pub/Sub
4. Use Cloud Pub/Sub Import Service

8. A team of developers wants to create standardized patterns for processing IoT data. Several teams will use these patterns. The developers would like to support collaboration and facilitate the use of patterns for building streaming data pipelines. What component should they use?

1. Cloud Dataflow Python Scripts
2. Cloud Dataproc PySpark jobs
3. Cloud Dataflow templates
4. Cloud Dataproc templates

9. You need to run several map reduce jobs on Hadoop along with one Pig job and four PySpark jobs. When you ran the jobs on premises, you used the department's Hadoop cluster. Now you are running the jobs in GCP. What configuration for running these jobs would you recommend?

1. Create a single cluster and deploy Pig and Spark in the cluster.
2. Create one persistent cluster for the Hadoop jobs, one for the Pig job and one for the PySpark jobs.
3. Create one cluster for each job, and keep the cluster running continuously so that you do not need to start a new cluster for each job.
4. Create one cluster for each job and shut down the cluster when the job completes.

10. You are working with a group of genetics researchers analyzing data generated by gene sequencers. The data is stored in Cloud Storage. The analysis requires running a series of six programs, each of which will output data that is used by the next process in the pipeline. The final result set is loaded into BigQuery. What tool would you recommend for orchestrating this workflow?

1. Cloud Composer
2. Cloud Dataflow
3. Apache Flink
4. Cloud Dataproc

11. An on-premises data warehouse is currently deployed using HBase on Hadoop. You want to migrate the database to GCP. You could continue to run HBase within a Cloud Dataproc cluster, but what other option would help ensure consistent performance and support the HBase API?

1. Store the data in Cloud Storage
2. Store the data in Cloud Bigtable
3. Store the data in Cloud Datastore

4. Store the data in Cloud Dataflow

12. The business owners of a data warehouse have determined that the current design of the data warehouse is not meeting their needs. In addition to having data about the state of systems at certain points in time, they need to know about all the times that data changed between those points in time. What kind of data warehousing pipeline should be used to meet this new requirement?

1. ETL
2. ELT
3. Extraction and load
4. Change data capture

Support / Sign Out

PREV
Chapter 2 Building and Operationalizing Storage Systems

NEXT
Chapter 4 Designing a Data Processing Solution