Chapter 1

Streaming and Batch Data Ingestion

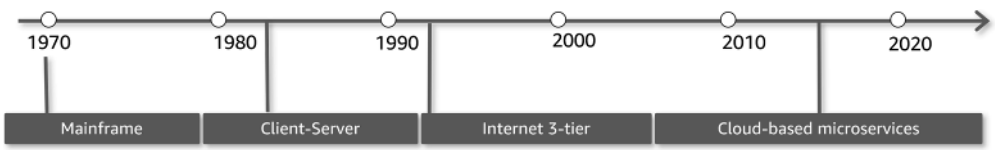
THE AWS CERTIFIED DATA ENGINEER – ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

- ✓ **Domain 1: Data Ingestion and Transformation**
  - Task 1.1 Perform data ingestion

The Evolution of Application Architectures and Data Stores

In order to understand batch and streaming data ingestion, we need to understand how data architectures have evolved over time and why the exam focuses on certain topics. To understand the evolution of data architecture, we should look at the applications that are typically the data source for the data engineering pipelines and understand how the application architectures have evolved (see Figure 1.1).

FIGURE 1.1 The evolution of application architectures



Application architectures have actually changed quite a bit in recent decades. Originally, companies used on-premises mainframes to handle their critical applications. These mainframes combined all aspects of an application—compute and storage—and ran for years without interruption.

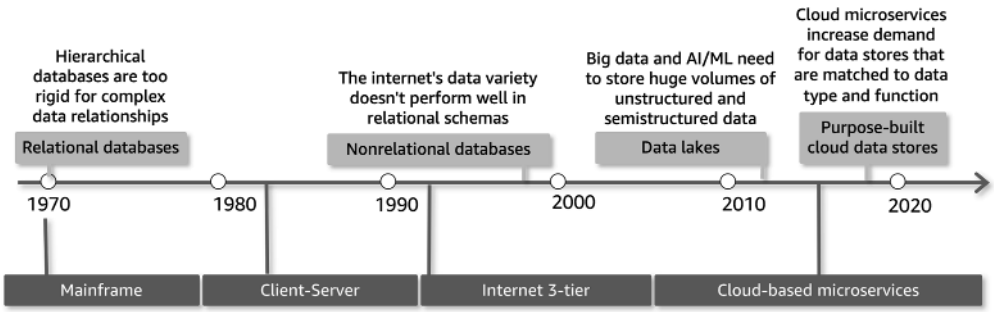
Then, applications were split into pieces using a client-server architecture. The server responded to requests from various clients, which allowed for more distributed systems. The clients and servers could be on the same computer, but the separation allowed for more scalable systems when needed. Clients and servers could be split up so they would not compete for resources.

With the rise of the Internet, the three-tier architecture became prominent. Applications were split into groups by function: a presentation tier served as the user interface; an application tier handled the business logic and processing; and a data tier provided long-term, persistent storage. Again, this increased the scalability of applications because each tier could be scaled independently.

Each of these moves focused on splitting up your application to improve scaling and resiliency. The most recent architectural trend, which aligns well to cloud best practices, is to move to microservices. With microservices, you split your application into different services based on functionality. For example, rather than having one application that handles both your inventory data and order history data, you might split those into two separate services that are focused on their domain. This split allows the two services to scale independently and provides more agility between development teams.

Parallel to the evolution of application architecture is the evolution of how we store and access data. This evolution is worth a quick review to provide context to the tools and best practices that apply to modern data architectures (see Figure 1.2).

FIGURE 1.2 The evolution of data stores



Relational databases replaced hierarchical databases, which had limited abilities to define relationships among data. Relational databases have been around since 1970 and continue to be the workhorse of many applications.

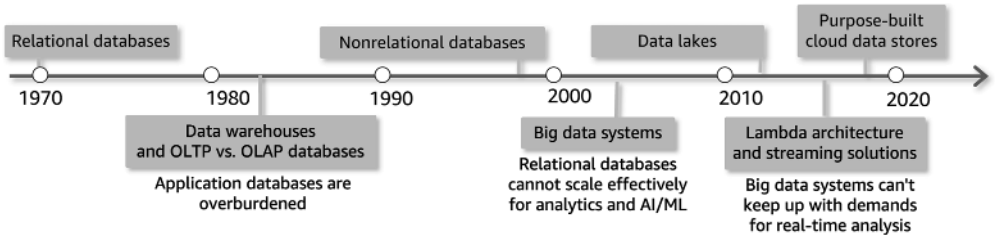
As the Internet era took off, organizations started to get more varied types of data, which they wanted to be able to analyze. This new variety of data was less structured and needed less rigid rules. This led to the development of nonrelational databases in the late 1990s.

With the growth of big data and artificial intelligence and machine learning (AI/ML) applications, organizations realized that they wanted to collect as much data as possible for potential use without the rigors of transforming it into a formal structure. Organizations wanted to simplify the collecting and querying of data in its most raw form. With a data lake, raw data could be loaded into the data store, where it could be held for a variety of use cases.

Cloud computing gave organizations the freedom to scale data stores up and down based on actual usage. Coupled with a move toward microservices, the flexibility of the cloud made it attractive to connect different application components to different databases, rather than relying on a single multipurpose one. Purpose-built data stores emerged to let developers optimize the storage that is connected to a component to match the data type and processing of that component. For example, developers might use a ledger database for financial transactions.

While data stores were evolving to handle different types of data, data architectures were also evolving to handle a continued increase in volume and velocity (see Figure 1.3).

FIGURE 1.3 The evolution of data architectures



In the 1980s, data warehouses evolved as a way to separate operational reporting, which requires read-heavy querying across a full dataset, from the application's transactional database, which is focused on fast reads and writes in a much smaller set of records. Data warehouses are still relational databases, but they have been optimized for reporting and analytics. Online analytical processing (OLAP) databases are optimized for reporting, whereas online transaction processing (OLTP) databases are designed for transactions, such as creating an order or making an ATM withdrawal. The extract, transform, and load (ETL) process was introduced to extract data from OLTP databases, transform it, and then load it into the data warehouse.

As noted earlier, the rise of the Internet brought new data to be collected and analyzed. Even with a data warehouse dedicated to analysis, keeping up with the volume and velocity of incoming data created database bottlenecks. Administrators could scale vertically (i.e., increase the size and speed of the database), but there wasn't an easy way to scale horizontally (i.e., to distribute the load across multiple databases). Big data systems or frameworks addressed this shortcoming in the 2000s. Big data frameworks were designed to distribute data across multiple nodes and handle any failures automatically. These frameworks also allowed the big data systems to handle many ETL transformations, which helped to increase the speed with which analysis could be done.

Work was still mostly done by batching at some scheduled interval. Batch processing created a lag between the arrival of new data and its being included in analytics results. The continually increasing volume and velocity of new data meant that the time between batches created an increasingly large gap in the timeliness of data being processed. The volume and velocity also prevented more real-time analysis and decisions. As noted earlier, data has its greatest value when it can be analyzed as early as possible to when it was created. But as you also learned, the most valuable, predictive insights are more difficult to derive. To find a balance between value and complexity, Nathan Marz proposed the Lambda architecture—an approach that combines the use of batch processing with stream processing to support close-to-real-time insights. This approach has become a standard way to process big data.

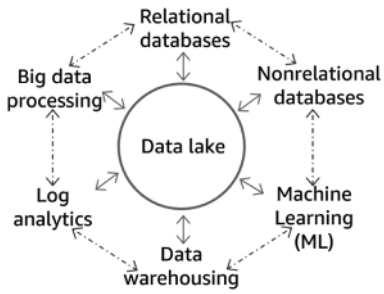
So, which of these data stores or data architectures is the best one for your data pipeline? The reality is that a modern architecture might include all of these elements. The key to a modern data architecture is to apply the three-pronged strategy that you learned about earlier. *Modernize* the technology that you are using. *Unify* your data sources to create a single source of truth that can be accessed and used across the organization. And *innovate* to get higher value analysis from the data that you have.

Introduction to the Modern Data Architecture

The goal of the modern data architecture is to store data in a centralized location and make it available to all consumers to perform analytics and run AI/ML applications. But that doesn't mean that you have one data store—only that you have a single source of truth. As illustrated in Figure 1.4, a data lake provides the centralized repository of data and is integrated with the other types of data stores and data processing systems that were described in the previous section. Data might be queried directly from the lake, or it might be moved to and from other purpose-built tools for processing.

Figure 1.4 provides a conceptual look at the components of a modern data architecture and how they are connected to each other. The diagram doesn't reflect the actual architecture that accommodates these connections.

FIGURE 1.4 A conceptual diagram of a modern data architecture



The movement of data among the lake and other integrated services falls into three general types: outside in, inside out, and around the perimeter.

Outside in is when an organization that stores data in purpose-built data stores, such as a data warehouse or a database, moves data into the lake to run analysis on it. For example, they copy query results for product sales in a given region from their data warehouse into their data lake to run product recommendation algorithms against a larger dataset by using ML.

Inside out is when an organization stores data in a data lake and then moves a portion of that data to a purpose-built data store to do additional ML or analytics. For example, they collect clickstream data from web applications directly into a data lake and then move a portion of that data out to a data warehouse for daily reporting.

Around the perimeter is when an organization moves data directly between the other data store components that are integrated with the data lake without needing to access the data lake. For example, they might copy the product catalog data stored in their database to their search service to make it easier to look through their product catalog and offload the search queries from the database.

The modern data architecture approach addresses the strategies of modernizing, unifying, and innovating with your data architecture, but it also introduces a lot of complexity to constructing your data pipeline. The main challenge with a data lake architecture is that raw data is stored with no oversight of the contents. In order for a data lake to make data usable, it needs to have defined mechanisms to catalog and secure data. Without these elements, data cannot be found or trusted, which results in a *data swamp*.

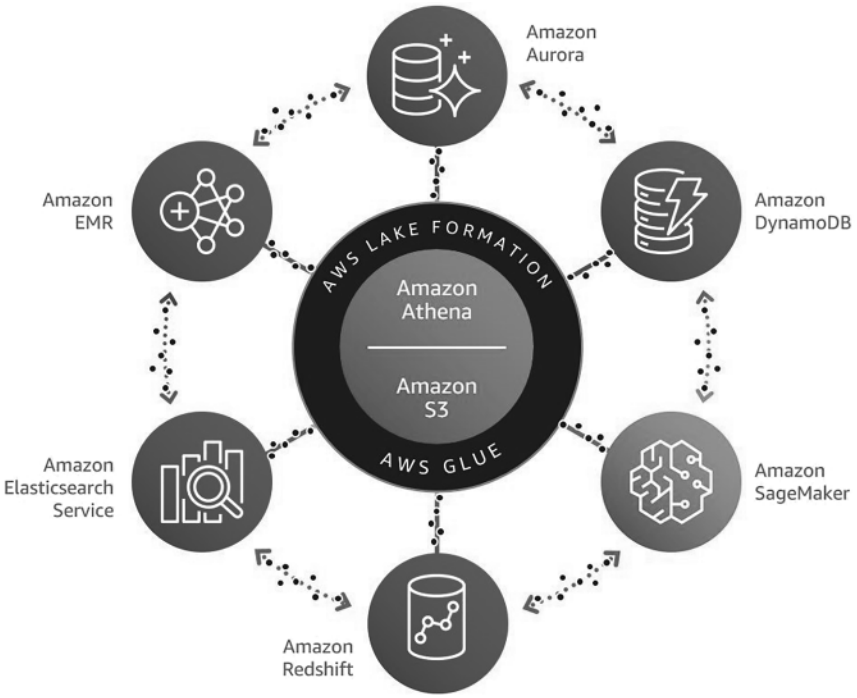
To build a successful modern data architecture, your design must incorporate the following features:

- The data lake should be able to easily scale as data grows. Use a scalable, durable data store that supports multiple ways to bring data in. For each component, select scalable services that balance the fastest performance at the lowest cost for the use case. Choose purpose-built tools. One of the key exam objectives is around choosing specific tools for specific use cases. You will continually assess options that could increase performance or reduce costs.
- Your design should also support the seamless data movement into and out of the data lake and around the perimeter—and, whenever possible, provide direct access to the data.
- Your design must also ensure the consistency of data across all components of the architecture. As data in your various data stores and data lake continues to grow, it becomes more difficult to move all of that data around securely and in a governed way. This is referred to as *data gravity*. In this architecture, it's critical to have robust mechanisms for authorization and auditing, with a centralized location to define policies and enforce them.

In the conceptual diagram shown in Figure 1.5, you see AWS services that map to the conceptual elements of the modern data architecture and support its key design considerations. You will learn more about each of these components throughout the book and will see how these connections are reflected in reference architectures that support this conceptual view.

Amazon Simple Storage Service (Amazon S3) provides storage for structured and unstructured data and is the storage service of choice to build a data lake on AWS. With Amazon S3, you can cost-effectively build and scale a data lake of any size in a secure environment with high durability. With Amazon S3, you can also use native AWS services to run big data analytics and AI/ML applications.

FIGURE 1.5 Modern data architecture on AWS



Amazon Athena is shown directly on the data lake to illustrate that it provides interactive querying of data directly in Amazon S3.

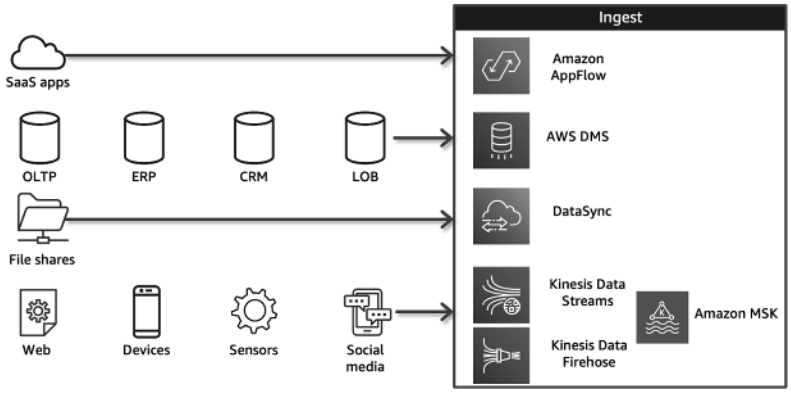
The architecture illustrates other AWS purpose-built services that integrate with Amazon S3 and map to each AWS service illustrated in Figure 1.5:

- Amazon Redshift is a fully managed data warehouse service.
- Amazon OpenSearch Service is a purpose-built data store and search engine that is optimized for real-time analytics, including log analytics.
- Amazon EMR provides big data processing and simplifies some of the most complex elements of setting up big data processing.
- Amazon Aurora provides a relational database engine that was built for the cloud.
- Amazon DynamoDB is a fully managed nonrelational database that is designed to run high-performance applications.
- Amazon SageMaker is an AI/ML service that democratizes access to ML processing.

To meet the goal of seamless data movement, Athena and Amazon Redshift both support federated queries and the ability to run queries across different data stores without needing to move the data between stores to perform the queries.

This chapter focuses mainly on streaming and batch ingestion. Figure 1.6 shows, at a high level, all the ingestion services available within the analytics ecosystem at the time of this writing, and also gives you a quick summary of which services to choose when. We'll discuss more of these later in the chapter.

FIGURE 1.6 Matching ingestion services to variety, volume, and velocity



Introduction to Data Ingestion

Data ingestion, a critical phase in the data engineering lifecycle, involves collecting data from various sources and moving it to a platform where it can be stored, processed, and analyzed. This chapter delves into the nuances of data ingestion, focusing on AWS services, and covers key concepts such as throughput, latency, data ingestion patterns, and the intricacies of both streaming and batch data ingestion.

We'll use the data engineering lifecycle (Figure 1.7) to focus our energy on the key job requirements from a data engineering role, and explain how they relate to the exam requirements from the AWS Data Engineering – Associate certification exam. In essence, we would like to start working backward from the core objectives of data engineering and the end goal that needs to be achieved. If you look at a typical data engineering lifecycle, you move from the left-hand side of Figure 1.7 to the right, which is:

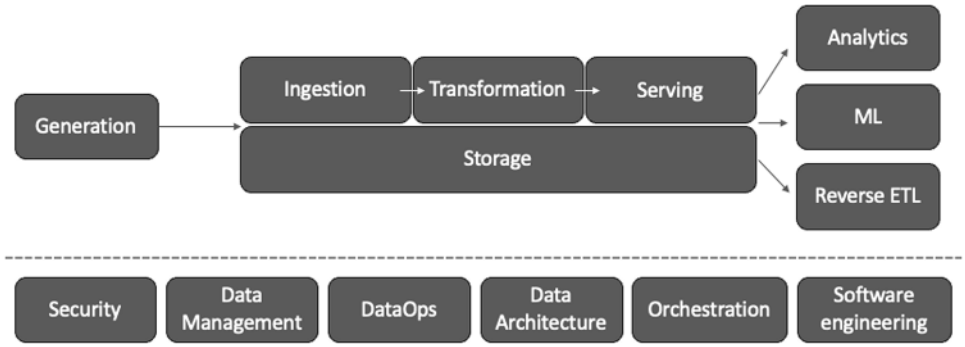
- Generation
- Storage
- Ingestion
- Transformation
- Serving

The layer below the dotted line includes critical components that will be relevant for the entire lifecycle, irrespective of the particular phase.

Throughout the book, we'll touch upon the different components of the data engineering pipeline.

Let's start by discussing data generation.

FIGURE 1.7 The data engineering lifecycle



Data Generation

At the heart of data engineering is the process of data generation, where the journey of raw data begins. This stage is fascinating because it pulls in data from everywhere—like social media, online transactions, sensors, and user interactions—creating a huge, fast-moving, and diverse flow of information. Think of this as the data coming at us in all shapes, sizes, and speeds, challenging us with its sheer amount (volume), its rapid arrival rate (velocity), and its wide range (variety), from simple numbers to complex videos.

For a data engineer, this wild world of data is something they can't control. It's like trying to manage a river in flood; they can't control how much data is coming, how fast it's arriving, or what kind of data it is. But what they can do is quite remarkable: they can create systems to organize this chaos, making sure the data can be used effectively. They tackle the big waves of data (volume), keep up with its speed (velocity), and make sense of its many forms (variety). It's a mix of creativity and tech know-how, ensuring all this data is neatly collected, stored, and ready for the next steps.

Getting this right is crucial because it sets the stage for everything that follows in turning data into something valuable. Well-handled data can lead to insights that help make smart decisions and spark innovative ideas. But if it's not managed properly, opportunities are missed, and the data's potential is lost. The beginning phase of generating data, with all its challenges, is key to the whole data engineering process, affecting how data is stored, processed, analyzed, and visualized later on. How well a data engineer navigates this phase is essential for unlocking the value of the data, leading to powerful analytics, machine learning, and informed decisions that drive an organization forward. The exam will present you with simulated real-world situations, often actual customer problems, and present you with a list of solutions that might be suited for the particular scenario.

If you summarize the typical types of sources we work with, you will see data in three broad categories:

- Existing transactional systems (such as CRM systems or POS systems), which are typically based on databases like Aurora, MySQL, Oracle, Microsoft SQL Server: Data can arrive in batches or in a streaming fashion.

- Streaming data coming from IoT devices, sensors, and social media: Data mostly arrives in streaming fashion.
- Files coming from web servers (e.g., logs): Data can arrive in batches or in a streaming fashion.

Each of these sources can provide data with different volumes and velocities, depending on the type of the source. For example, existing transactional systems would typically provide structured data, which is pretty consistent in terms of its structure, whereas data coming from sensors and IoT devices can often change its structure based on the latest OS versions, application types, and so on.

Understanding Data Sources and Storage

Before ingesting data, it's essential to understand the origin and storage of the data. The data might originate from diverse sources like IoT devices, transactional databases, or online transaction processing systems. Recognizing how data is generated and stored, including the data's schema, frequency, and volume, is foundational for designing an effective ingestion strategy.

You need to ensure that you understand how the source system works, the way it generates data, the frequency and velocity of data, and the types of data being generated. Here are just a few questions to consider at this stage:

- What are the characteristics of the data (e.g., structured, semi-structured, unstructured, data types, data formats)?
- How is that data persisted in the source system (e.g., database, flat files, message queues)?
- Will there be any duplicates, what is the schema of that source data, and does that schema change?
- What is the volume of data being generated, and is it expected to grow over time?
- Are there any data quality issues or inconsistencies in the source data that need to be addressed?
- Are there any data governance or security requirements for the source data?
- Are there any dependencies or integrations with other systems that need to be considered?
- What are the data access and extraction mechanisms available for the source system?
- Are there any specific data transformation or cleansing requirements before loading the data into the target system?
- What are the expected service-level agreements (SLAs) or performance requirements for data extraction and loading?
- Are there any specific data retention or archiving requirements for the source data?

- Which error handling and recovery mechanisms are in place for the data extraction process?
- Are there any specific monitoring or auditing requirements for the data extraction process?

Ingestion Patterns and AWS Services

Ingestion patterns vary based on the nature and requirements of the data. Typically, data ingestion from source is broadly categorized into batch and streaming ingestion.

To generalize the characteristics of batch processing, batch processing is a data handling approach that involves the systematic execution of automated jobs that retrieve information from a source, transfer the resulting datasets to a secure storage location within the data pipeline, and then apply the necessary transformations tailored to the particular requirements of the given project. The scope of these transformations can vary significantly. For simpler use cases, it might involve basic cleansing and formatting to facilitate the integration of data into a data lake system. However, for more intricate objectives, such as those aimed at supporting extensive data queries, big data analytics, or machine learning (ML) applications, the process might encompass a more elaborate sequence of enrichment, augmentation, and intensive processing.

Batch jobs can be triggered in several ways: they may be initiated manually upon request, set to run according to a predetermined timetable, or launched automatically in response to certain triggers or events. The conventional extract, transform, and load (ETL) framework relies on batch processing techniques, though the alternative extract, load, and transform (ELT) approach may also employ batch processing methods.

One of the principal advantages of batch processing is its proficiency in handling vast volumes of accumulated transactional data over specific intervals. It excels particularly in executing complex tasks that require extended processing times. When immediate data analysis is not a pressing concern, batch processing becomes a particularly attractive option due to its cost-effectiveness and the relative simplicity of its management. This efficiency makes it an optimal choice for organizations looking to process large datasets without the need for real-time results.

In direct contrast to the batch processing model, stream processing activates a continuous flow of data. In this model, records are constantly fed into and transmitted through the stream. Consumers of this data, which can be systems or applications, then read and process these records in real time or near-real-time, within a dynamically moving time window. This method not only facilitates real-time analytics but also ensures that processed data can be promptly made available for immediate review and action. Furthermore, processed records may be stored in durable systems for subsequent use and analysis.

Streaming processing's standout feature is the ability to deliver processed results almost instantaneously, allowing for immediate action based on the data analyzed. This low-latency analysis is especially beneficial for handling high volumes of fast-moving, unstructured data streams. Stream processing is particularly designed to accommodate scenarios where the speed and immediacy of data processing are crucial.

AWS offers a range of services tailored for different ingestion needs, which we will discuss over the course of this chapter.

We have a lot of content to cover in this chapter, so the standard approach that we will take is to first introduce the different services covered in the exam, and then discuss their implications during the ingestion process.

Data Ingestion

There are two major types of data ingestion:

- **Batch ingestion**: This approach deals with data in finite sets. Jobs are often executed according to a predetermined schedule, such as hourly or nightly. It's akin to a bakery preparing dozens of loaves of bread at once and then selling them the next morning.
- **Streaming ingestion**: Here, data flows continuously, akin to a stream of water. This model is more suitable for tasks that require immediate attention, such as monitoring live financial transactions or managing the flow of data from millions of IoT devices.

The remainder of this chapter covers the AWS services that can be used for streaming and batch ingestion.

Streaming Ingestion

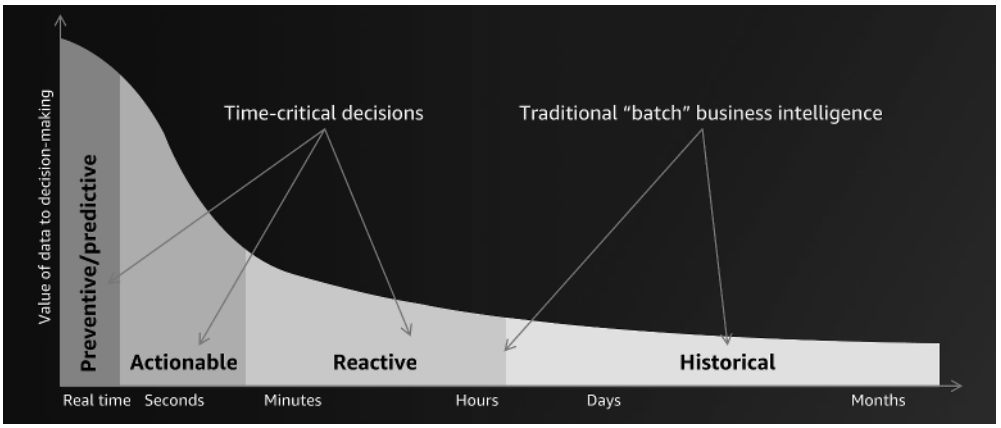
By 2025, IDC estimates there will be 175 zettabytes of data globally (that's 175 with 21 zeros), with 80% of that data being unstructured. As data grows exponentially, so does its velocity, necessitating tools that can swiftly capture, process, analyze, and derive actionable insights from this torrent of information.

Streaming data systems are particularly valuable because they can deliver ongoing insights with minimal delay, even when dealing with data influx that is both high in volume and variable in rate. The urgency of processing data is underscored by the concept that data's value diminishes as it ages—what's referred to as the *shelf life* of data. This concept doesn't imply that all data inevitably loses value, but it emphasizes that certain scenarios demand extremely low latency to maintain relevance. Figure 1.8 demonstrates the perishable nature of insights and the need to act upon the information quickly to realize optimal value.

Take, for instance, the scenario of fraud detection in credit card transactions. Here, the speed at which a fraudulent activity is identified is critical. Detecting fraud quickly not only positions a company to address and mitigate risks effectively but also prevents potential financial losses. Figure 1.8 explains how the value of data decreases over time in relation to decision-making. When time is of the essence and you have a time-sensitive situation at hand, Amazon Kinesis or Amazon MSK can demonstrate their strength. By enabling real-time data processing, streaming applications help organizations act swiftly, preserving the

high value of immediate insights. The platform ensures that the valuable opportunity to act upon data at the peak of its relevance is not missed, as illustrated by the concept of the *perishable nature of insights*, which suggests that the sooner information is acted upon, the greater the benefit or value derived from it.

FIGURE 1.8 Information half-life in decision-making



Source: Perishable Insights, Mike Gualtieri, Forrester

In summary, streaming services are designed to handle the complexities of high-velocity, unstructured data streams, providing businesses with the ability to process and analyze data in real time. This capability is not just a technical achievement but a strategic asset for any data-driven organization operating in today's fast-paced environment.

Now, let's delve into the nuances of stream processing and understand why it might be the superior choice in certain situations:

- **Decoupling data collection from processing**: Stream processing acts as a middle layer that separates the incoming flow of data (from producers) and the mechanisms that process this data (the consumers). This system acts like a flexible buffer zone, accommodating data at its own pace, which can be critical in managing variable data rates. Decoupling producers from consumers is a best practice when it comes to data management.
- **Unifying data streams**: Imagine a scenario with a million sensors each sending temperature data to be analyzed. Stream processing enables all these individual streams to converge seamlessly into a single, manageable channel.

- **Maintaining the order of events**: Certain sequences, such as bank transactions where the order of debit and credit is crucial, must be preserved. Stream processing ensures that the sequence remains unchanged from the moment it's sent to when it's processed.
- **Parallel processing**: Much like multiple lanes on a highway that allow cars to move simultaneously, stream processing allows for concurrent operations on the same dataset. This enables diverse applications to work side by side, each at its own pace, which can significantly accelerate time to market.

During the exam, you should look out for the following key attributes of stream processing applications to understand the use cases where a stream processing application make sense:

- **Ultra-low latency**: Stream processing is about speed, ensuring that data is processed almost as quickly as it's generated.
- **Guaranteed message delivery**: Just as a courier service ensures your parcel arrives, stream processing ensures data gets where it needs to go.
- **Support for the Lambda architecture**: This is a framework for data processing that provides the robustness of batch processing along with the speed of stream processing.
- **State management**: This capability allows you to keep track of the context of ongoing transactions within the data stream.
- **Time-based and count-based windows**: These windows help in organizing data for processing based on time intervals or quantities, which can be fixed or rolling.
- **Resilience to failure**: A stream processing system is designed to withstand problems, continuing to operate smoothly in the face of disruptions.

Despite these advantages, implementing streaming capabilities can be challenging, which is why organizations often struggle with setting up and scaling such systems. As data volumes skyrocket from thousands to billions of events, the complexity increases. Ensuring high availability and seamless integration with existing systems can also be challenging, not to mention the management complexities and maintenance costs.

In response to these challenges, AWS has expanded its offerings to simplify the streaming data ecosystem. This includes:

- **Amazon Kinesis**: A fully managed service designed to handle real-time data streams.
- **Amazon Managed Streaming for Kafka (MSK)**: This service offers a managed environment for Apache Kafka, which is widely used for building real-time data pipelines and streaming apps.
- **Amazon Managed Flink**: A service that provides a managed environment for Apache Flink, an open-source stream processing framework.

These services are designed to abstract away the intricacies of stream processing, allowing businesses to leverage the power of real-time data analysis without the technical overhead.

Amazon Kinesis Introduction

Amazon Kinesis is AWS's flagship solution for real-time data streaming and analytics. It's designed to handle massive streams of data in real time, enabling businesses to analyze and respond to data quickly. The service is crucial for applications that require continuous data intake and processing, such as log and event data collection, real-time analytics, machine learning model inference, and more.

Amazon Kinesis provides multiple components, such as Amazon Kinesis Data Streams, Amazon Data Firehose (formerly Kinesis Data Firehose), and Amazon Managed Service for Apache Flink. Figure 1.9 shows the different streaming options on the Amazon Kinesis console page.

FIGURE 1.9 The Amazon Kinesis console page

Let's dive into each service one by one. If you're preparing for an exam, it's crucial to understand the functions of these services, along with their capacity for handling data and their response times. Also, know when it's best to use them over alternatives, and be familiar with the methods for inputting and retrieving streaming data.

Amazon Kinesis Data Streams

Amazon Kinesis Data Streams is a massively scalable, near-real-time data streaming service that allows you to scale GBs of data per second from thousands of sources.

Imagine you have a bustling digital marketplace, much like a busy city center, but instead of people, you have a flood of data rushing in from every corner: clicks, video views, app activity, and more. Just as city planners manage traffic, you need a way to manage this data. This is where Amazon Kinesis Data Streams comes in—it's like a series of high-speed roadways designed for data.

Kinesis Data Streams is a service offered by Amazon Web Services that's built to handle massive waves of data in real time. Think of it as a set of superhighways for data that lets you collect, process, and analyze information as quickly as it comes in, allowing for instantaneous decision-making and action-taking. Whether it's temperature readings from air sensors or clicks on a website, Kinesis can handle it all with minimum configurations.

The service works by creating data streams, which you can picture as individual lanes on a highway. Each lane, or *shard*, as it's referred to in Kinesis, can carry a certain amount of data. If you need more capacity, you can add more shards. This makes Kinesis incredibly flexible and scalable—you can adjust the number of shards as the traffic of your data changes.

What sets Kinesis Data Streams apart is its ability to process and analyze data almost as soon as it's generated. So, if you're an online retailer, you can instantly track how customers are interacting with your site and adjust your strategies in real time. If you're a game developer, you can see how players are experiencing your game and fix issues immediately.

Amazon Kinesis was built to be accessible, and with some basic setup, you can start streaming data while paying only for what you use. You do not need to invest in heavy-duty servers or infrastructure, as Amazon Kinesis grows with your needs, whether you're a small startup or a large corporation.

In short, Amazon Kinesis Data Streams is like the traffic control system for data, ensuring that every bit of information gets where it needs to go quickly and efficiently, helping businesses make better decisions faster.

As Figure 1.10 illustrates, you have the ability to capture and store your data, after which you can direct it to a variety of applications for further processing. This includes leveraging services like Amazon Data Firehose, Amazon Managed Service for Apache Flink, employing Spark on Amazon EMR, or utilizing AWS Lambda for custom processing workflows. Once your data has been processed, you can bring it into your preferred business intelligence (BI) tool to create visualizations that can inform decision-making.

FIGURE 1.10 Kinesis Data Streams overview

Source: Amazon Web Services, Inc.

Use Cases for Amazon Kinesis Data Streams

Amazon Kinesis Data Streams is adept at handling high-speed data collection and aggregation from multiple sources. It stands out in several key scenarios:

- **Quick capture and processing of log data:** Web and application servers, as well as IoT devices, generate a continuous flow of log data that needs to be gathered and processed quickly. Kinesis Data Streams offers a solution to capture this information with minimal

delay, allowing for prompt data processing, which can help mitigate issues like server malfunctions or data loss.

- **Real-time metrics for business insight**: Businesses often require immediate visibility into their operations. For example, manufacturers may need to track the number of active machines on a shop floor and their output in real time. Delays in identifying issues could lead to costly downtime. Kinesis Data Streams provides the capability to monitor these metrics as they happen, which is crucial for timely decision-making and maintaining operational efficiency.
- **Immediate data analytics**: Kinesis Data Streams is particularly useful for scenarios requiring instant data analysis, such as identifying anomalies or detecting fraudulent transactions. By processing data as it's received, businesses can intervene rapidly when they detect unusual patterns that may indicate a problem.
- **Managing complex data flows**: In situations where data comes from various sources and needs to be combined—or aggregated—Kinesis Data Streams excels. It can handle the complexity of processing these diverse data streams, ensuring that they're orchestrated efficiently and effectively for downstream applications.
- **Integration with data warehousing**: A typical application of Kinesis Data Streams is to collect data and distribute it for further processing. One part of the system may transform the data or take immediate action based on its contents. Another part can group the data into batches and store it in a data lake, such as Amazon S3, from which it can be transferred to a data warehouse like Amazon Redshift for more in-depth analysis.
- **Enhancing search and analytics applications**: Logs from applications are vital for understanding user behavior and system performance. Kinesis Data Streams can funnel these logs into search and analytics platforms, such as OpenSearch. This enables rapid searching and visualization, and with tools like Kibana, you can create dashboards to track and display key events or trends.

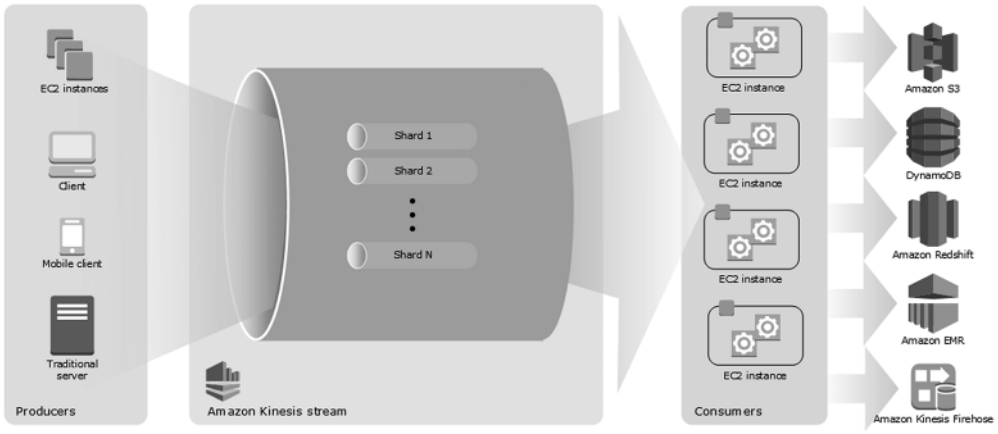
In all these scenarios, Kinesis Data Streams acts as a powerful conduit for data, ensuring it is captured, processed, and made ready for whatever analysis or action is needed, swiftly and efficiently.

Kinesis Data Streams—Flow

As you may have understood by now, Amazon Kinesis Data Streams is a scalable service that scales elastically for near-real-time processing of streaming big data. The service stores large streams of data (gigabytes of data per second) in durable, consistent storage, reliably for near-real-time (few seconds) data processing by an elastically scalable fleet of data processing servers.

Streaming data processing has two layers: a storage layer and a processing layer (see Figure 1.11). The storage layer must support specialized ordering and consistency semantics that enable fast, inexpensive, and repayable reads and writes of large streams of data.

FIGURE 1.11 Kinesis Data Streams data flow



The processing layer is responsible for reading data from the storage layer, processing that data, and notifying the storage layer to delete data that is no longer needed.

Customers can compile the Amazon Kinesis library into their data processing application. Amazon Kinesis notifies the application (the worker) when there is new data to process. The control plane works with workers to solve scalability and fault tolerance problems in the processing layer.

Let's look at some of the key concepts of Kinesis Data Streams:

Kinesis Data Streams—Deep Dive

In this section, we'll take a close look at many aspects of Kinesis Data Streams.

What Is a Kinesis Data Stream?

We touched upon the concept of a "shard" earlier in the chapter. So, what is a "shard" actually? A shard is a logical concept that holds a series of data records in order, much like a queue. Every data record in this queue has a unique sequence number that Kinesis assigns to keep track of it. A Kinesis data stream is actually a collection of shards.

Data Record Basics

A data record is the smallest unit of information within a Kinesis data stream. Think of it as a packet of data that includes a sequence number (kind of like a tracking ID), a partition key (which helps organize the data), and a data blob. This blob is just a fancy term for a chunk of data, which can be as large as 1 MB. Kinesis doesn't peek into or alter this blob; it just stores it exactly as it receives it.

Choosing Your Stream's Capacity Mode

When setting up a Kinesis data stream, you'll decide on a capacity mode. This affects how the stream's capacity is managed and how you're billed. You can go with either on-demand or provisioned modes:

- **On-demand mode** means Kinesis automatically handles the stream's capacity to match your needs, billing you only for what you use. It's flexible and adjusts to your throughput requirements, whether they increase or decrease.
- **Provisioned mode** requires you to specify upfront how many shards you need. Your billing is based on the number of shards. You can adjust the shard count as your needs change.

Retention Period: How Long to Keep Data

By default, data in your stream is kept for 24 hours. However, you can extend this up to 365 days or shorten it back to the 24-hour minimum. Longer retention periods cost extra.

Producers and Consumers

While their names are self-explanatory, it is important to understand what producers and consumers are.

- **Producers** are sources that add records to your stream. For example, a web server logging activity could be a producer.
- **Consumers** take records from your stream to process them. They're typically applications designed to read and act on the streamed data.

Kinesis Data Streams Applications

These are specific consumers designed to work with the stream's data. They can be deployed across multiple EC2 instances and can feed data into another stream or AWS services.

Shard Details

A shard is like a lane in a highway; it directs a portion of the traffic. Each shard supports a certain number of transactions per second for both reading and writing data. The overall capacity of your stream depends on how many shards it has. You can adjust the number of shards based on your data traffic.

Partition Keys and Sequence Numbers

It's important to understand the role that partition keys and sequence numbers play:

- **Partition keys** help organize data into shards. Each record is tagged with a partition key, which Kinesis uses to determine which shard the record belongs to.
- **Sequence numbers** are unique identifiers for records within a shard, assigned by Kinesis after a record is added. They help keep track of the order of records.

Sequence numbers can't be used to categorize data within the stream. For organizing different sets of data, use partition keys or separate streams.

Working with the Kinesis Client Library

The Kinesis Client Library (KCL) helps your application process data from your stream efficiently and reliably. It ensures there's a process running for each shard, simplifying data consumption. The KCL uses a DynamoDB table for managing state information.

Naming Your Application

Each Kinesis Data Streams application needs a unique name within your AWS account and region. This name is used for DynamoDB table names and CloudWatch metrics.

Encrypting Your Data

Kinesis Data Streams supports automatic encryption for sensitive data as it enters the stream, using AWS Key Management Service (KMS) keys for security.

Example Code Snippets

Let's look at a couple of code snippets that deal with writing data to and reading data from Amazon Kinesis.

Adding a Record to a Stream

Here's how you can add a record to a stream:

```
import boto3
kinesis_client = boto3.client('kinesis', region_name='your-region')
# Create a data record
data = '{"example": "data"}'.encode('utf-8')
partition_key = 'examplePartitionKey'
# Put the record into the stream
response = kinesis_client.put_record( StreamName='yourStreamName', Data=data,
PartitionKey=partition_key )
print(response)
```

Consuming Records from a Stream

This example assumes you're using the Kinesis Client Library. The library abstracts away the details of fetching records from the stream, allowing you to focus on processing the data.

```
public class YourRecordProcessor implements IRecordProcessor {
    public void initialize(String shardId) {
        // Initialization logic here
    }

    public void processRecords(List<Record> records, IRecordProcessor
Checkpointer checkpointer) {
        for (Record record : records) {
            // Process each record
```

```
        }
    }

    public void shutdown(IRecordProcessorCheckpointer checkpointer,
ShutdownReason reason)
```

Writing Data to Streams

To elaborate on writing data to Amazon Kinesis Data Streams and the considerations for ensuring efficient data transfer and processing, let's look at the different ways in which you can write data to Amazon Kinesis Data Streams and dive deeper into the methods and best practices for developing and managing Kinesis producers.

Producers in Amazon Kinesis Data Streams

A producer is essentially any application that sends data to Kinesis Data Streams. These can be built using the AWS SDK, specifically with Java in mind, and the Kinesis Producer Library (KPL), which offers a more streamlined interface for sending data to your streams.

The following are the different ways you can write data to an Amazon Kinesis Data Stream:

- Using the AWS SDK for Java and the Kinesis Producer Library (KPL):
  - Simplifies the development of producer applications.
  - Provides high write throughput, automated retries, record batching (aggregation), and seamless integration with the Kinesis Client Library (KCL).
- Directly using the Amazon Kinesis Data Streams API with the AWS SDK for Java:
  - Offers fine-grained control over stream operations, including creating streams, resharding, and managing records.
  - Requires handling of retries and record batching manually.
- Writing to Amazon Kinesis Data Streams using Kinesis Agent:
  - Kinesis Agent is a standalone Java software application that offers an easy way to collect and send data to Kinesis Data Streams.
  - It handles file rotation, checkpointing, and retries upon failures automatically.
  - It delivers data in a reliable, timely, and simple manner.
- Writing to Kinesis Data Streams using other AWS services:
  - AWS services like AWS Lambda, AWS IoT, Amazon CloudWatch Logs, and others can be configured to automatically write data to Kinesis Data Streams.
- Using third-party integrations:
  - Tools like Apache Flink, Fluentd, Logstash, and others support sending data to Kinesis Data Streams, expanding the ecosystem of data producers.

- Developing custom applications:
  - Custom applications developed using AWS SDKs in various programming languages can put data into Kinesis Data Streams.
  - These applications can run on EC2 instances, on-premises servers, or even as part of mobile or web applications.

Each method provides unique benefits and trade-offs in terms of simplicity, control, and integration with other AWS services or third-party tools.

Understanding Basic Requirements

To initiate data transfer to a stream, a producer must specify the following:

- **Stream name**: Identifies which stream the data should be sent to.
- **Partition key**: A key that determines the shard within the stream where the data is stored. The choice of partition key affects how data is distributed across shards and is crucial for optimizing data retrieval and processing.
- **Data blob**: The actual data payload you want to send to the stream.

The Importance of Partition Keys

Just like any distributed system where data distribution plays a key role in performance, partition keys play a vital role in data distribution across shards. The Kinesis service uses the partition key to assign the data to a specific shard. A well-thought-out partition key strategy is crucial to avoid uneven data distribution, which can lead to hot shards and affect the stream's ability to scale and process data efficiently.

Using the Kinesis Producer Library

The Kinesis Producer Library (KPL) simplifies the development of Kinesis producers by providing features such as high write throughput, automated retries, and data aggregation. This library batches multiple data records into a single Kinesis record, enhancing throughput and reducing the number of API calls needed. It's essential to monitor the KPL's performance and keep it updated to leverage improvements and bug fixes. We'll look at the KPL in more detail later in this section.

Monitoring and Troubleshooting

Amazon CloudWatch offers monitoring capabilities for tracking the operational metrics of your Kinesis producers. This is vital for identifying and resolving issues such as increased latency or throughput bottlenecks.

Advanced Topics for Kinesis Data Streams Producers

There are two advanced topics to understand about the KPL:

Handling Retries and Rate Limiting The KPL automatically retries failed data records, using a strategy that considers both the time-to-live of the record and the maximum buffered time. Additionally, it implements rate limiting to prevent excessive retries from

overloading the system. This is particularly important when dealing with partial failures and ensures that data is eventually written to the stream without overwhelming the shards.

KPL Aggregation Aggregation is a powerful feature of the KPL that bundles multiple smaller records into a larger one, improving data transfer efficiency. However, this necessitates the use of de-aggregation logic on the consumer side to process the original records individually.

Key Considerations for Optimizing Producer Performance

When optimizing producer performance, you should know the following:

- **Security and IAM policies**: Ensure that producers have the necessary permissions to write to streams, which involves configuring IAM roles and policies correctly.
- **Optimal use of partition keys:** Design your partition key strategy to ensure a balanced distribution of data across shards, avoiding throttling and maximizing throughput.
- **Monitoring and analytics:** Utilize CloudWatch to monitor key metrics related to your producers and streams, allowing for proactive adjustments and optimization.
- **Update and maintain KPL**: Regularly update the KPL to benefit from performance improvements, new features, and bug fixes, ensuring your producers operate efficiently.

While AWS provides a simple interface to write data to an Amazon Kinesis stream, effectively writing data to data streams requires a comprehensive understanding of how producers work, optimal use of the KPL, and ongoing monitoring and optimization efforts to handle data at scale. The certification exam will typically have a couple of questions on KPL, so understanding it can help you with scoring those key certification points.

KPL Deep Dive

Think of the Kinesis Producer Library (KPL) as a smart assistant for your data. When you have a lot of information to send to different places, it can get tricky.

Maybe you're trying to:

- Send the same piece of information to several spots at once
- Keep trying to send data if it didn't work the first time
- Deal with a huge number of small bits of data all at once
- Make sure that the data is easy to use once it gets where it's going
- Keep track of how well the data sending and receiving is going

Doing all this manually can be a real headache. That's where KPL comes in to save the day. It's good at handling a lot of data all at once, and it does it quickly.

What's Great About the KPL?

Let's say you have a bunch of machines in a factory, and each one is sending tiny updates. You have to collect all these tiny updates and send them to a data stream. If you were to send each update one by one, it would take forever. The KPL helps you by grouping a lot of

these tiny updates into one big update, allowing you to send multiple updates at once. This not only makes things faster but also ensures you don't hit the limits on how much data you can send at once.

With the KPL, you also don't have to worry about the details of making sure your data gets sent properly. It's smart enough to keep trying if there's a problem without you having to tell it to.

On the other side, when you want to use the data you've collected, the KPL makes that easier, too. It works well with the Kinesis Consumer Library (KCL), which is like KPL but for taking data out of the stream. If you're not using the KCL, you can still get your data out in a useful way with the KPL's tools.

The KPL also keeps an eye on how well your data is being sent and lets you see this information in CloudWatch. That means you can check on how fast and reliably your data is moving.

The KPL works in the background, so when you send data to it, it doesn't slow you down. It takes care of sending the data while you go on with other things. Later, it lets you know how it went.

In a nutshell, the KPL is like a behind-the-scenes wizard for your applications, making sure the data they create gets to the Kinesis data streams without a fuss.

Example of Using KPL

Here's how you might write some simple code to send data with KPL:

```
kinesis = KinesisProducer()

for i in range(100):
    data = ByteBuffer.wrap("myData".encode("UTF-8"))
    kinesis.addUserRecord("mystream", "mypartitionkey", data)
```

This little loop just sends the message `"myData"` to your stream a hundred times. The KPL takes care of the rest, making sure it all gets where it's meant to go.

Easily Send Data with Kinesis Agent

If you want to get your data into a stream without a fuss, Kinesis Agent is here to help. It's a program you can set up on your computer to do several helpful things:

- It watches your files and sends any new data to your Kinesis stream.
- It handles file rotation, which means it can switch to a new file when the old one gets too big or old.
- It remembers the last bit of data it processed, so it always knows where it left off.
- If there's a problem sending data, it tries again until it works.
- It also sends updates to CloudWatch so you can see how things are going.

Normally, Kinesis Agent looks for a new line (you know, when you press Enter on your keyboard) to figure out when one piece of data ends and the next begins. However, you can tweak it if your data is a bit more complex.

How to Get Your Data Flowing with Kinesis Agent

Here's how to install your agent and get it running:

Set Up the Agent First, you have to install it. If you're using Amazon Linux, type the following in your computer's command line:

```
sudo yum install -y aws-kinesis-agent
```

For Red Hat Linux, it's a bit different. You'll download it directly from a web link like this:

```
bashCopy code
sudo yum install -y https://s3.amazonaws.com/streaming-data-agent/aws-kinesis-agent-latest.amzn1.noarch.rpm
```

Or, if you like using GitHub, here's how to do it:

- Grab the agent from the awslabs/amazon-kinesis-agent page on GitHub.
- Install it with the following command:

```
sudo ./setup --install
```

Get the Agent Ready and Running Now, you need to tell the agent what to do. There's a file you'll need to edit called `/etc/aws-kinesis/agent.json`. In this file, you'll set two main things:

- `filepattern`: This is how the agent knows which files to look at.
- `kinesisstream`: This is where you tell it which stream to send data to.

Kinesis Agent is smart enough to notice when you have new files, as long as they don't all pop up in the same second.

To start the agent, you can do it manually each time with:

```
sudo service aws-kinesis-agent start
```

Or, to make sure it starts whenever your computer does, you can set it up like this:

```
sudo chkconfig aws-kinesis-agent on
```

Once it's running, Kinesis Agent works quietly in the background, keeping an eye on the files you told it to and sending new data to your stream.

Using the AWS SDK for Java to Transmit Data to a Kinesis Stream

Leveraging the AWS SDK for Java presents a reliable method for pushing data to an Amazon Kinesis stream. The SDK provides two distinct operations to facilitate adding data to a stream: `PutRecords`, for batch data transmission, and `PutRecord`, for individual data record transmission.

PutRecords: Batch Data Transmission

The `PutRecords` operation permits the simultaneous transmission of multiple data records in a singular HTTP request. This approach is conducive to achieving higher data throughput to a Kinesis data stream.

The `PutRecords` operation is capable of handling up to 500 records per request, each record up to 1 MB in size, with the cumulative request size capped at 5 MB, partition keys inclusive.

Here is a conceptual demonstration of how you might use the `PutRecords` API in Java:

1. Initialize the Kinesis client builder with the appropriate regional settings and authentication credentials.
2. Construct a `PutRecordsRequest` and populate it with a list of `PutRecords RequestEntry` objects, each representing a data record.
3. Submit the `PutRecordsRequest` to the Kinesis stream and await the response to confirm successful data submission.

PutRecord: Individual Data Record Transmission

Compared to `PutRecords`, the `PutRecord` operation is designed to dispatch a single data record per HTTP request. Employing this operation, each data record is appended to the stream with a unique sequence number, an identifier assigned by Kinesis Data Streams. While the `PutRecord` operation is available, the use of `PutRecords` is generally advocated for efficiency, given its capacity for batch processing. Data within the stream is categorized using a partition key, which also determines the shard assignment within the stream.

The utilization of the aforementioned operations within the AWS SDK for Java streamlines the process of data integration into Kinesis streams, ensuring a seamless and efficient data management workflow.

Writing to Kinesis Data Streams Using Third-Party Options

If you're looking to funnel data into Kinesis Data Streams, you have several nifty third-party tools at your disposal that play well with Kinesis:

- **Apache Flink**: This is more than just a tool; it's a full-on framework for processing data streams, whether they're constantly flowing or just a finite batch. To connect it with Kinesis Data Streams, there's a specific connector you can use.
- **Fluentd**: Imagine a Swiss Army knife for log management—that's Fluentd. It helps you bring your logging game to the next level and can work in tandem with Kinesis to streamline data processing.
- **Debezium**: If you're into keeping tabs on your databases and reacting to every little change, Debezium is your go-to. It's all about capturing those changes, and you can pipe them straight into Kinesis Data Streams.
- **Oracle GoldenGate**: Need to shuffle data from one database to another? Oracle GoldenGate is like a data-moving concierge that can also funnel your data right into Kinesis.

- **Kafka Connect**: Kafka's great at shuttling messages around, and Kafka Connect bridges the gap to Kinesis, making data transfer a breeze.
- **Adobe Experience Platform**: For those who are serious about enhacing the customer experience, Adobe's platform lets you centralize customer data and use it to create personalized experiences. And yes, it can send data to Kinesis, too.
- **Striim**: This platform is like a data relay race—it collects, transforms, analyzes, and sends data off in real time. It's another great way to get your data streaming smoothly into Kinesis.
- **Amazon Pinpoint**: Not just for marketing, Amazon Pinpoint can pinpoint (pun intended) the right data and send it over to Kinesis for further action.

Each of these tools has its own way of connecting to Kinesis Data Streams, so you'll want to check out the specific guides on how to set them up. Whether you're looking to capture changes as they happen, shuffle data between systems, or just make your logs more manageable, there's a solution that can help you get your data where it needs to go.

Reading Data from Kinesis Data Streams

Reading data from Amazon Kinesis Data Streams is a fundamental aspect of building streaming applications on AWS. The following sections provide an overview of the various ways to read data.

Data Viewer in the Kinesis Console

You can use the Data Viewer feature in the Kinesis console to quickly inspect the data in your stream. This is useful for debugging or verifying data ingestion. However, this method is not meant for processing large amounts of data or for use in production environments due to its limitations in throughput and the manual nature of the operation.

Querying Data Streams in the Kinesis Console

The Kinesis console provides a querying feature that allows you to run simple SQL-like queries against the stream. This is good for quick tests and inspections, but, similar to the Data Viewer, it's not designed for processing high volumes of data or complex queries.

Using AWS Lambda

Lambda functions can be set up to process Kinesis data streams. Here's an example of how to set up a Lambda function to process records from a Kinesis stream in Node.js:

```
exports.handler = async (event, context) => {
event.Records.forEach(record => {
 const payload =Buffer.from(record.kinesis.data,'base64').toString('ascii');
 console.log('Decoded payload:', payload);
});
return `Successfully processed ${event.Records.length} records.`;
};
```

Be mindful of the Lambda timeout and concurrency limits, and ensure that the batch size and window are configured properly to handle your stream's throughput.

Using Managed Service for Apache Flink

Amazon Kinesis Data Analytics is the easiest way to process data streams with Apache Flink. You write your stream processing applications in SQL or Java, and Kinesis Data Analytics takes care of everything required to run your queries continuously and automatically scales to match the volume and throughput of your incoming data.

Using Amazon Data Firehose

Amazon Data Firehose allows you to deliver data streams to destinations such as S3, Redshift, OpenSearch, and Splunk. The setup is straightforward and managed by AWS, so there is no need for custom coding or resource management. However, Firehose has throughput limits and buffering, which can introduce latency.

Using the Kinesis Client Library

The Kinesis Client Library (KCL) is a Java-based library provided by AWS to simplify the process of consuming and processing data from Kinesis data streams. The KCL handles complex issues such as fault tolerance, scaling in or out, distributing stream records among multiple consumers (workers), coordinating record processing, and checkpointing. By checkpointing, we mean that the KCL tracks the progress of record processing so that it can resume from the last point in case of failures.

The Relationship Between KCL and KPL

The Kinesis Producer Library (KPL) is a complementary library to KCL but for the opposite side of the data stream flow. While KCL is used for consuming data from Kinesis data streams, KPL is used for efficient and reliable production of data records to the stream. KPL allows for batching, retrying, and monitoring capabilities, enhancing the ability to publish data to Kinesis data streams at high throughput and with low latency. KPL aggregates records to increase throughput and manages the low-level details of sending data to Kinesis data streams.

Here's a snippet for a KCL 2.x consumer in Java:

```
public class RecordProcessor implements ShardRecordProcessor {
    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (KinesisClientRecord record : processRecordsInput.records()) {
            String data = StandardCharsets.UTF_8.decode(record.data())
.toString();
            System.out.println(data);
        }
    }
    // Implement other interface methods...
}
```

Processing Multiple Data Streams with KCL 2.x for Java

KCL 2.x supports processing multiple Kinesis data streams within the same application. Each stream will have its own set of resources, and you can run multiple KCL applications within the same JVM process. Here's how you would configure a KCL application for processing multiple streams:

```
// For Stream 1
KinesisClientLibConfiguration kclConfig1 = new KinesisClientLibConfiguration(
        "applicationName1",
        "streamName1",
        credentialsProvider,
        workerId)
        .withInitialPositionInStream(InitialPositionInStream.TRIM_HORIZON);

// For Stream 2
KinesisClientLibConfiguration kclConfig2 = new KinesisClientLibConfiguration(
        "applicationName2",
        "streamName2",
        credentialsProvider,
        workerId)
        .withInitialPositionInStream(InitialPositionInStream.TRIM_HORIZON);

Worker worker1 = new Worker.Builder()
        .recordProcessorFactory(new RecordProcessorFactory())
        .config(kclConfig1)
        .build();

Worker worker2 = new Worker.Builder()
        .recordProcessorFactory(new RecordProcessorFactory())
        .config(kclConfig2)
        .build();

// Start workers for each stream in their own thread
new Thread(worker1).start();
new Thread(worker2).start();
```

Each application maintains its own checkpoints and operates independently, allowing for the concurrent processing of different streams.

Lease Tables

Lease tables are DynamoDB tables used by the KCL to track the assignment of shards to workers. They help synchronize the state across multiple instances of an application. Each

record in a lease table represents a lease on a shard within the data stream. The lease owner, typically a KCL worker, is responsible for processing the records from that shard.

Tracking Processed Shards

You can use a lease table to track the shards processed by the KCL consumer application and thus KCL to achieve fault tolerance and load balancing. Here's what happens under the hood:

- **Lease acquisition**: When a KCL application starts, it attempts to acquire leases for the shards it will process.
- **Lease stealing**: In a multi-worker setup, if some workers are overloaded while others are idle, the KCL may redistribute the leases so the load is evenly balanced.
- **Lease renewal**: Each worker continuously renews its leases to prevent them from expiring and being taken over by other workers.
- **Lease checkpointing**: After successfully processing a batch of records, the worker checkpoints its progress by writing the sequence number of the last record processed into the lease table.

The KCL consumer application uses this lease information to resume processing from the last checkpointed position in case of a restart or failure. This ensures that no data is lost and the processing is exactly once or at least once, based on how you handle the checkpointing logic.

Here's how you might interact with the lease table in your application:

```
public class RecordProcessor implements ShardRecordProcessor {
    private Checkpointer checkpointer;

    @Override
    public void initialize(InitializationInput initializationInput) {
        checkpointer = initializationInput.checkpointer();
    }

    public void processRecords(ProcessRecordsInput processRecordsInput) {
        for (KinesisClientRecord record : processRecordsInput.records()) {
            String data = StandardCharsets.UTF_8.decode(record.data())
.toString();
            // Process record logic here
        }
        try {
            // Checkpoint once all records are processed
            checkpointer.checkpoint();
        }
```

Custom Consumers and Enhanced Fan-Out

Two more considerations are custom consumer applications and enhanced fan-out:

- **Developing custom consumers:** You can build your own consumer applications using the AWS SDK. This allows for shared throughput across multiple consumers but is subject to the stream's read limits.
- **Enhanced fan-out:** Enhanced fan-out is a key concept and often discussed during certification exam questions on streaming data involving Kinesis. This feature provides dedicated throughput to consumers by allowing them to receive records from a stream with lower latency and at a higher rate than standard consumers. When using KCL 2.x, you can create enhanced fan-out consumers, which offer improved performance.

```
// Example of creating an Enhanced Fan-Out consumer with KCL
ConsumerConfig consumerConfig = new ConsumerConfig.Builder()
        .streamName(streamName)
        .applicationName(applicationName)
        .enhancedFanOut()
        .build();
```

Limitations

There are several limitations to be aware of:

- Each method has its throughput limits, and it is important to understand these when architecting your solution.
- Lambda has payload size limits and execution duration limits, which may require additional batching logic.
- For the KCL, careful consideration must be given to worker distribution and checkpointing to avoid data loss.

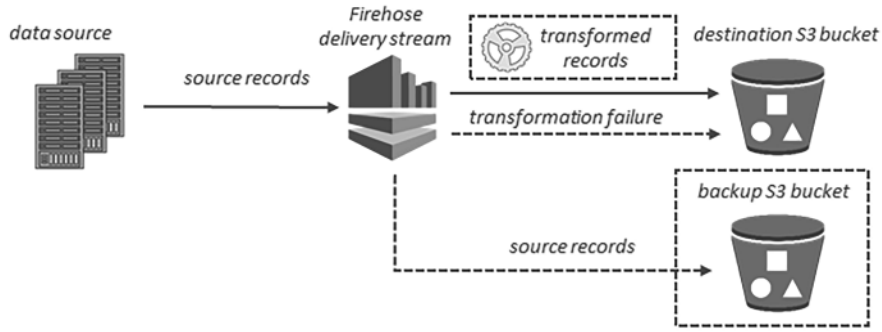
Amazon Data Firehose

Processing data in a stream is achieved using Amazon Kinesis Data Streams and AWS Lambda in combination; however, often there is a need to persist the data for either long-term storage or further analysis. While Amazon Kinesis Data Streams was the first service that was built, the team soon realized that one of the most popular use cases for a streaming application was to capture data being emitted from these devices in real time and buffer it before storing it for longer-term analysis. Amazon, with its usual customer obsession, decided to build Amazon Data Firehose (previously known as Amazon Kinesis Data Firehose), a fully managed service, to allow users to deliver streaming data to a number of destinations, including Amazon S3, Amazon Redshift, Amazon OpenSearch Service, and a number of partner services like Coralogix, Datadog, Dynatrace, Elastic, custom destinations using an HTTP endpoint, Honeycomb, LogicMonitor, Logz.io, MongoDB Cloud, New Relic, Snowflake, Splunk, and Sumo Logic.

Amazon Data Firehose allows you to create a delivery stream, which can have the data written to it by various producers (max record length of 1,000 KB), and buffer the data (configurable parameter in MB or seconds) before delivering it to a target destination.

One of the most popular use cases is customers transforming the data arriving to a Data Firehose stream by fanning the transformed and raw data to different destinations (see Figure 1.12).

FIGURE 1.12 Data flow—S3 destination



The process of streaming data varies depending on the destination due to the inherent characteristics and requirements of the target systems. For instance, when the target is an OpenSearch cluster, the data is directly streamed to the cluster, as it is designed to handle real-time data ingestion. However, in the case of Redshift, which is a data warehousing solution optimized for analytical workloads, the data is first delivered to an S3 bucket, and then a Copy command is executed to transfer the data from the S3 bucket to Amazon Redshift in a more efficient and optimized manner (refer to Figure 1.13 and Figure 1.14). This approach allows for better management and processing of large volumes of data before loading it into Redshift.

FIGURE 1.13 Amazon Redshift as a destination for Data Firehose

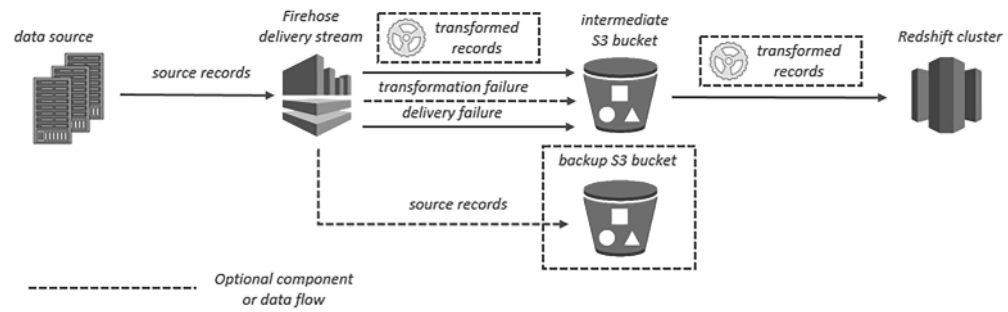
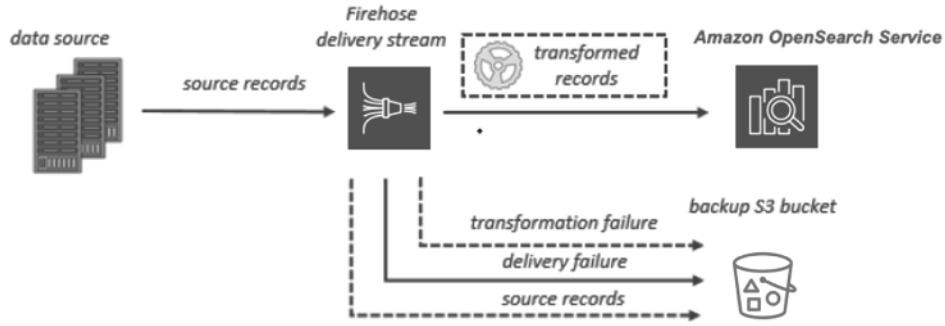
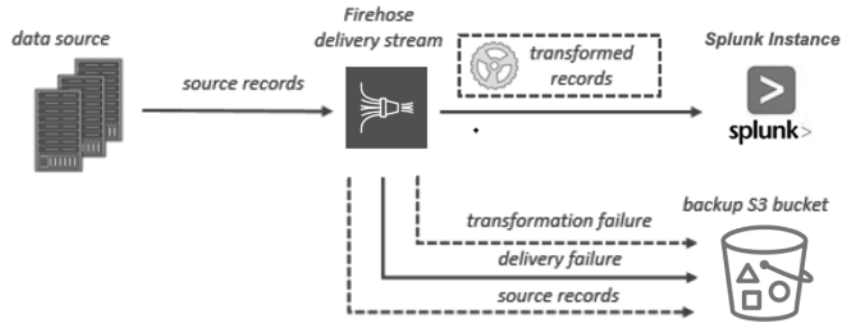


FIGURE 1.14 Amazon OpenSearch as a destination for Data Firehose



When the destination is Splunk, streaming data is delivered to Splunk but can also optionally be backed to S3 (see Figure 1.15).

FIGURE 1.15 Splunk as a destination for Data Firehose

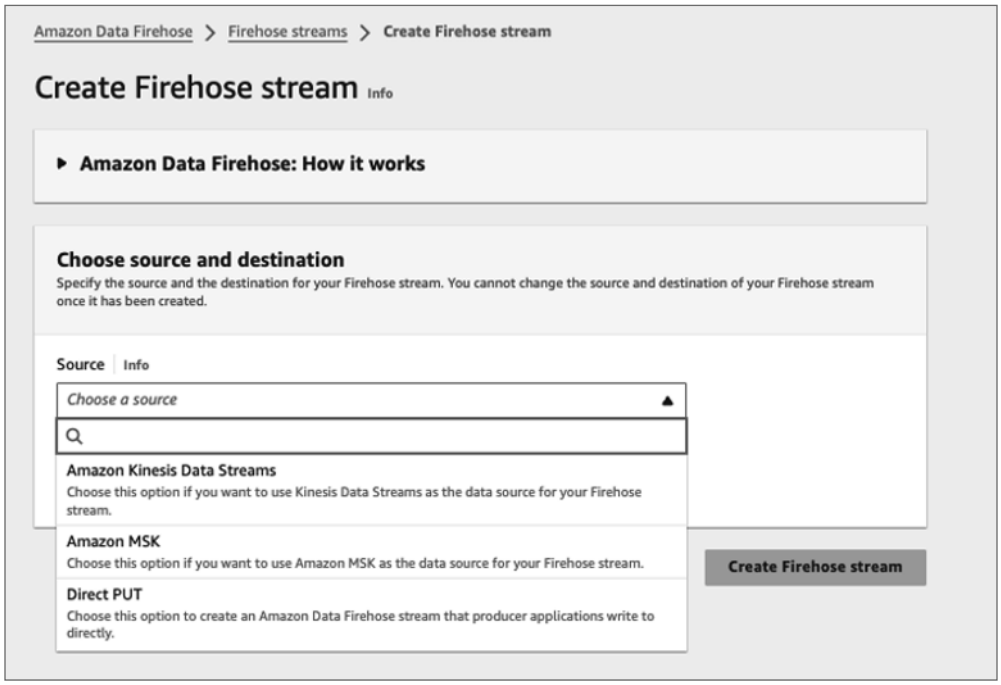


While Amazon Data Firehose supports multiple destinations, it also provides a number of different native sources on the platform that can write to the AWS Data Firehose (see Figure 1.16). Using the Kinesis API, you can also build connectors for new sources as well.

The native sources are as follows:

- **Kinesis Data Streams**
  - If your source Kinesis data stream has aggregation configured, Amazon Data Firehose will de-aggregate the records before delivering them to a destination.
  - When Kinesis Data Streams is used as a source for Data Firehose, the `putRecord()` and `putRecordBatch()` operations are disabled. If you want to use `putRecord()` or `putRecordBatch()`, you can use them on the Kinesis data stream itself.
  - Amazon Data Firehose starts reading records from the *latest* position of your Kinesis stream, and more than one Data Firehose delivery stream can read from the same Kinesis stream.

FIGURE 1.16 Native sources (console options) for Amazon Data Firehose



- **Amazon MSK**
  - Amazon Data Firehose also supports Amazon MSK as a data source. You can choose between MSK-provisioned and MSK-serverless clusters. With Amazon Data Firehose, you can effortlessly extract data from a chosen Amazon MSK cluster and topic, and then transfer it to a designated S3 destination.
    - Kinesis Agent
    - AWS SDK
    - CloudWatch Logs
    - CloudWatch Events
    - AWS IoT

Data Transformation with Amazon Data Firehose

While capturing data is critical, you will often need to perform data transformations for data streaming at high velocity. You can enable data transformation with Amazon Data Firehose. When you turn on data transformation in Firehose, it stores incoming data temporarily. The amount of data it can store before processing ranges from 0.2 MB to 3 MB. By default, it

aims to gather 1 MB of data for processing for most destinations, but for Splunk and Snowflake, this default is smaller, at 256 KB. The time Firehose waits before sending data for processing ranges from 0 to 900 seconds, with a default of 60 seconds for most destinations. Snowflake is an exception, with a default wait time of 30 seconds.

To change these settings, use the `ProcessingConfiguration` option in the `CreateDeliveryStream` or `UpdateDestination` API, specifying the desired buffer size and time interval. Firehose then sends the data in batches to a Lambda function for processing. Once the data is processed, Firehose sends it to the final destination either when it accumulates enough data to meet the buffer size or when the specified time interval passes, whichever occurs first.

You can also use pre-built Lambda blueprints with Amazon Data Firehose. Amazon Data Firehose provides a number Lambda blueprints that you can use for the most common transformations, including but not limited to:

- **Process records sent to Amazon Data Firehose stream (Node.js, Python)**: This blueprint shows a basic example of how to process data in your Firehose data stream using AWS Lambda.
- **Process CloudWatch logs sent to Firehose**: This blueprint is deprecated and will not appear in the exam.
- **Convert Amazon Data Firehose stream records in syslog format to JSON (Node.js)**: This blueprint shows how you can convert input records in RFC3164 Syslog format to JSON.

AWS Serverless Application Repository contains a number of different blueprints you can use to simplify processing with Amazon Data Firehose. To view a list of available blueprints, visit `https://serverlessrepo.aws.amazon.com/applications` and search for "Firehose."

Converting Input Record Formats in Amazon Kinesis

One of the most common analytical requirements is to transpose data from row-based to columnar format to enhance query performance. Parquet and ORC are widely favored by users for this purpose due to their efficient data compression and superior analytical query execution. Amazon Data Firehose facilitates this transformation of JSON data into either Parquet or ORC formats. To accomplish the format conversion, Amazon Data Firehose requires the following three key components:

- **Deserializer**: This is employed to interpret the JSON structure of your incoming data. There are two options available for deserialization:
  - Apache Hive JSON SerDe
  - OpenX JSON SerDe
- **Schema definition**: To understand and map your data accurately, a schema is essential. AWS Glue can be utilized to automatically deduce a schema from your data and catalog it within the AWS Glue Data Catalog. Amazon Data Firehose will then consult this

schema for data interpretation, which can also be applied across other AWS services that are schema-compatible.

- **Serializer:** For the final transformation into columnar storage, you can employ either ORC SerDe or Parquet SerDe.

By configuring these elements in Amazon Data Firehose, you can streamline your data for more effective analysis.

If you enable record format conversion, you can't set your Amazon Data Firehose destination to be Amazon OpenSearch Service, Amazon Redshift, or Splunk. With format conversion enabled, Amazon S3 is the only destination that you can use for your Firehose stream.

Amazon Managed Service for Apache Flink

Now we'll examine Amazon Managed Service for Apache Flink.

What Is Apache Flink?

Apache Flink is an open-source stream processing framework for distributed, high-performing, always-available, and accurate data streaming applications. It is one of the most widely adopted open-source services and is used by companies like Netflix, Uber, Alibaba, Pinteret, and X (formerly Twitter).

Apache Flink stands out in the world of data processing for several compelling reasons.

It provides a suite of expressive APIs, including SQL API, Table API, DataStream API, and the ability to create stateful functions, offering flexibility and a rich set of functionalities for developers.

In terms of processing guarantees, Flink assures exactly-once state consistency, crucial for accurate data processing, and supports event-time processing and late data handling, enabling robust stream processing applications.

Its scale-out architecture allows it to adapt easily to the desired throughput, making it an ideal choice for applications that need to scale dynamically. This architecture can also support terabytes of state, facilitating the handling of large-scale data processing tasks.

Apache Flink boasts a vibrant open-source community that contributes to its continuous improvement and a broad set of connectors, enabling Flink to integrate with a wide variety of data systems, enhancing its interoperability and ease of use in diverse environments.

What Is Amazon Managed Service for Apache Flink?

Despite its robust capabilities in handling large-scale data processing, managing a Flink deployment introduces several complexities. These include ensuring high availability, managing consistent stateful computations across clusters, and scaling to meet fluctuating demands, all of which require substantial operational expertise and infrastructure management. Customers have often asked for an out-of-the-box offering of Apache Flink

that is faster and easier to deploy, fully managed, and offers a pay-as-you-go model. To alleviate these challenges, Amazon introduced a managed Flink service known as Amazon Managed Service for Apache Flink (Managed Apache Flink).

This managed service abstracts away the operational toil, enabling developers to focus on application logic rather than infrastructure management. Consumers benefit from this offering through simplified operations, automatic scaling, adaptive resource provisioning, and a pay-as-you-go pricing model, which collectively lower the barrier to entry for implementing real-time analytics and complex event processing with Flink. It allows users to analyze streaming data interactively using managed Apache Zeppelin notebooks with Amazon Managed Service for Apache Flink Studio. You can use Amazon Managed Service for Apache Flink to build applications in SQL, Java, Python, or Scala, to perform joins, filters, and aggregations over time windows, and more.

Amazon Managed Service for Apache Flink doesn't directly handle data management itself. Instead, it focuses on processing the streaming data you provide. However, there are multiple ways to get data into an Apache Flink application running on Amazon Managed Service for Apache Flink:

- **AWS services**: You can configure your application to read data from various AWS services that act as sources for your streaming data. Some examples include Kinesis Data Streams, SQS, DynamoDB Streams, and MSK.
- **External sources**: You can also set up your application to receive data from external sources outside of AWS. This could involve messaging systems like Kafka or custom connectors built specifically for your data source.

How Does It Work?

Figure 1.17 shows how Amazon Managed Service for Apache Flink works.

FIGURE 1.17 An overview of Amazon Managed Service for Apache Flink

Integrations of Amazon Managed Service for Apache Flink

Amazon Managed Service for Apache Flink provides a number of connectors to a variety of sources, including but not limited to:

- Amazon Kinesis Data Stream
- Amazon Managed Streaming for Apache Kafka
- Rabbit MQ
- Apache Kafka
- Flink connectors

Destinations of the Managed Flink service (also known as sinks) include but are not limited to:

- Amazon Kinesis data streams
- Amazon Managed Streaming for Apache Kafka
- Amazon Daga Firehose
- Amazon S3
- Amazon DynamoDB
- RabbitMQ
- Apache Cassandra
- Elasticsearch

Pricing Model

Amazon Managed Service for Apache Flink has no upfront costs and there are no resources to provision. The service is priced using KPUs (Kinesis Processing Units). Each KPU is a virtual bundle of 1 vCPU compute capacity and 4 GB of memory. The usage of Managed Flink is an hourly billing rate based on the number of KPUs used to run the application, and the service automatically scales the number of KPUs required by your stream processing application.

For information about Managed Flink pricing, visit `https://aws.amazon.com/managed-service-apache-flink/pricing/?`.

Amazon Managed Streaming for Apache Kafka

Amazon Managed Streaming for Apache Kafka, commonly known as Amazon MSK, is a fully managed service that emerged to simplify the provisioning and maintenance of Apache Kafka clusters in the cloud. Apache Kafka, originally developed by LinkedIn and later open-sourced as part of the Apache Software Foundation, is a distributed streaming platform designed to handle high volumes of data, enabling real-time data processing.

Recognizing the growing need for scalable data streaming solutions, Amazon Web Services incorporated Kafka into its robust ecosystem, offering a seamless experience that takes

care of the underlying infrastructure management. Amazon MSK provides the same Kafka APIs that developers are familiar with, but without the overhead of cluster operations, allowing them to focus on building applications that efficiently handle data streams.

Some of the key benefits of Amazon MSK include:

- **Security, high availability, and accessibility**: Amazon Managed Streaming for Apache Kafka (Amazon MSK) transforms the open-source Apache Kafka into a robust, enterprise-grade streaming solution by providing comprehensive management capabilities and enhanced features. At its core, MSK handles all infrastructure management tasks, including automated cluster deployment, zero-downtime patching, and resource allocation, eliminating the operational burden from organizations. The service implements a sophisticated security architecture, featuring VPC isolation, AWS IAM integration, SASL/SCRAM authentication, and end-to-end encryption using AWS KMS and TLS 1.2. High availability is ensured through a multi-AZ architecture with automatic data replication, fault tolerance mechanisms, and disaster recovery options, including cross-region replication and point-in-time recovery. MSK seamlessly integrates with the broader AWS ecosystem, supporting various AWS services like Lambda, S3, and Kinesis, while maintaining compatibility with existing Kafka clients and tools. The service offers extensive monitoring and management capabilities through CloudWatch integration, custom metrics, and comprehensive APIs. Organizations benefit from flexible scaling options, including dynamic broker scaling and storage auto-scaling, along with cost optimization features such as pay-as-you-go pricing and resource optimization recommendations. MSK addresses compliance requirements through various certifications (SOC, PCI, HIPAA) and provides robust audit logging via CloudTrail. Performance optimization is achieved through configurable throughput limits, storage options, and continuous monitoring. The service includes 24/7 AWS enterprise support, proactive maintenance, and regular security updates, making it an ideal choice for organizations seeking to leverage Kafka's capabilities without the operational overhead. This comprehensive solution particularly benefits organizations requiring enterprise-grade streaming capabilities, strict security controls, high availability, and efficient scaling while significantly reducing the total cost of ownership and accelerating time-to-market for streaming applications. By bridging the gap between open-source Kafka and enterprise requirements, MSK enables organizations to focus on building applications and processing data streams rather than managing infrastructure, making it a valuable solution for businesses of all sizes.
- **Best practices**: Amazon MSK embeds industry best practices into its core design philosophy, automatically implementing optimal configurations and operational standards without requiring deep Apache Kafka expertise. The service intelligently sets default configurations based on proven production patterns, including appropriate partition counts, replication factors, and broker settings that ensure optimal performance and reliability. Through automation, MSK handles critical operational tasks such as broker placement across availability zones, network configuration, security group setup, and resource allocation according to established best practices. The service automatically implements crucial security measures, including encryption at rest and in transit, secure

authentication methods, and proper network isolation. Performance optimization is built into the platform through automated monitoring, smart scaling decisions, and proactive maintenance scheduling. MSK's automated backup systems, failure detection, and recovery processes follow disaster recovery best practices, ensuring data durability and system availability. The service automatically manages cluster health through continuous monitoring and self-healing capabilities, applying patches and updates during maintenance windows that minimize impact on production workloads. Resource provisioning follows efficient sizing guidelines, with built-in alerts and recommendations when adjustments are needed. Additionally, MSK integrates logging, monitoring, and alerting best practices through its CloudWatch integration, providing meaningful metrics and actionable insights by default. These automated best practices extend to compliance and security configurations, ensuring that clusters meet regulatory requirements and security standards from the moment they're deployed. By embedding these best practices into the service's DNA, Amazon MSK significantly reduces the risk of misconfigurations, performance issues, and security vulnerabilities that often plague self-managed Kafka deployments, allowing organizations to focus on their business objectives rather than operational complexities.

- **Prioritize building applications**: Amazon MSK liberates developers from the complex, time-consuming tasks of managing Apache Kafka infrastructure by automating cluster provisioning, scaling, maintenance, and security configurations. This automation encompasses critical operational aspects like broker management, monitoring, patching, and backup processes, allowing development teams to concentrate their energy on building and enhancing their streaming applications rather than wrestling with infrastructure concerns. Moreover, the service's seamless integration with familiar AWS tools and services, combined with its support for standard Kafka APIs and tools, enables developers to maintain their existing workflows while benefiting from enterprise-grade reliability and security features without the associated operational overhead.
- **Eliminate cluster management**: MSK Serverless simplifies operations by removing the need for managing capacity, scaling, and partitioning.

Use Cases for Amazon MSK

Amazon MSK is essentially a managed Kafka offering and hence supports the use cases that are sweet spot for streaming services and applications. Customers typically use MSK for the following use cases. This is important to understand from an exam perspective so that you can align the right service to the right use case.

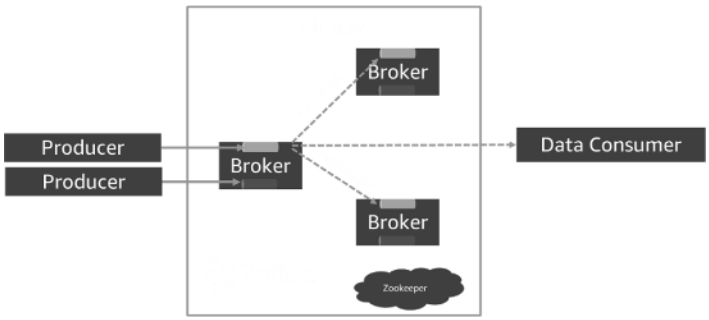
- Conducting analysis on web activity and logs as events occur.
- Capturing and storing sequences of events for transactional data.
- Facilitating communication between disconnected services through messages.
- Providing a flexible architecture by allowing services to operate independently.
- Enabling continuous data transformation and movement between systems.
- Gathering and synthesizing performance metrics and logs for systems.

Apache Kafka High-Level Architecture

Apache Kafka is structured as a distributed system for handling streaming data. At its core, data producers send records to the Kafka cluster, which consists of brokers responsible for maintaining published data. Each broker can act as a leader or follower in a process known as *replication*, ensuring high availability and fault tolerance. Data consumers then pull records from the brokers as needed. Underneath this architecture, Apache Kafka relies on Zookeeper for broker coordination and cluster management. This robust framework is designed to handle high-volume data streams in real time, making Kafka a vital component for applications that require reliable, fast, and scalable messaging solutions.

Figure 1.18 shows the Apache Kafka architecture at a high level.

FIGURE 1.18 Apache Kafka architecture

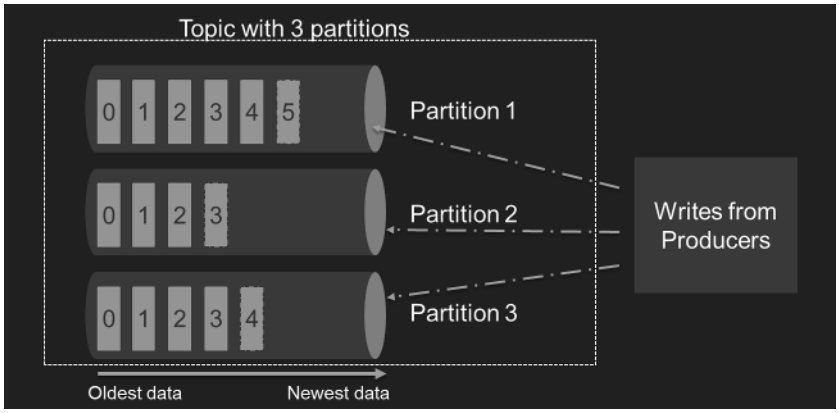


Apache Kafka—Anatomy of a Write

In the distributed architecture of Apache Kafka, data streaming is managed efficiently through the division of topics into multiple partitions. Each partition functions as an ordered, immutable sequence of records that are continually appended. For example, imagine a Kafka topic designated for tracking user activity on a website, segmented into three distinct partitions. Producers, which can be thought of as publishers, send records of data, such as user clicks or page views, to these partitions. The records are assigned to partitions typically based on certain criteria like a key value or round-robin assignment when no key is present.

Figure 1.19 shows a scenario where a user activity topic has three partitions. A producer responsible for user click events might send a new record to partition 1, which already contains records labeled with incremental offset numbers 0 through 5, indicating the order of arrival. If a second click event is sent without specifying a key, it may be assigned to partition 2 following the default partitioning logic, adding to the sequence of records in that partition. As new data flows in, Kafka maintains the order by appending each new piece of information to the end of the chosen partition, preserving the sequence in which events are produced. This allows consumer applications to process streams of data sequentially, thereby ensuring a coherent data history is maintained.

FIGURE 1.19 The anatomy of a Kafka write



There are two main approaches for writing data to Amazon Managed Streaming for Apache Kafka (MSK):

- Using a producer application:
  - This method involves developing a producer application specifically designed to send data to your MSK cluster. You can write this application in any programming language with Kafka client libraries available. Popular choices include Java, Python, and Go.
  - The producer application will need to connect to your MSK cluster using the bootstrap servers and utilizing the Kafka Producer API to publish messages to your desired topics.
- Using streaming services:
  - AWS offers several services that can act as data sources and automatically push data into your MSK cluster. Here are a couple of options:
    - **Amazon Data Firehose**: This service allows you to configure data delivery from various sources like S3 buckets, CloudWatch logs, or even databases directly to your MSK cluster.
    - **Lambda functions**: You can develop Lambda functions that process data and then publish it to your MSK topics using the Kafka Producer library within the Lambda environment.

Apache Kafka—Anatomy of a Read

In the Apache Kafka ecosystem, the process of reading data is just as structured and well-organized as the process of writing it. To read the data efficiently, we have the concept of a consumer group. A consumer group is a powerful concept that enables scalable

and fault-tolerant consumption of messages from Kafka topics. It consists of one or more consumer instances working together to process data from one or more topics (see Figure 1.20). This group structure allows for parallel processing, as multiple consumers can simultaneously read from different partitions of the same topic, significantly increasing throughput. Kafka automatically balances partitions across all consumers in a group, ensuring efficient load distribution. If a consumer fails, its partitions are reassigned to other active group members, maintaining processing continuity. Each consumer group manages its own offset in each partition it consumes, allowing different groups to process the same data at varying rates. This design supports scalability, as adding more consumers to a group can increase processing capacity up to the number of partitions in the topic.

Kafka guarantees at-least-once delivery to a consumer group unless the group's offset is explicitly modified. The system employs a group coordinator to manage membership, handle joins and leaves, and trigger rebalances when necessary. Each group is identified by a unique group.id, and multiple groups can independently read from the same topic, each maintaining its own offset. This flexible and robust architecture makes consumer groups essential for building scalable and resilient stream processing applications with Kafka.

In Figure 1.20, for example, a topic is divided into three partitions that correspond to different geographical regions' sales data. In this setup, we have three consumers in a consumer group, each tasked with consuming data from one partition. The consumer aligned with partition 1 will read sales records starting from the earliest unread message, marked by its next offset, say 0. As it reads and processes records, it advances to the next, ensuring that no data is missed and each record is processed once and only once by any consumer in the group.

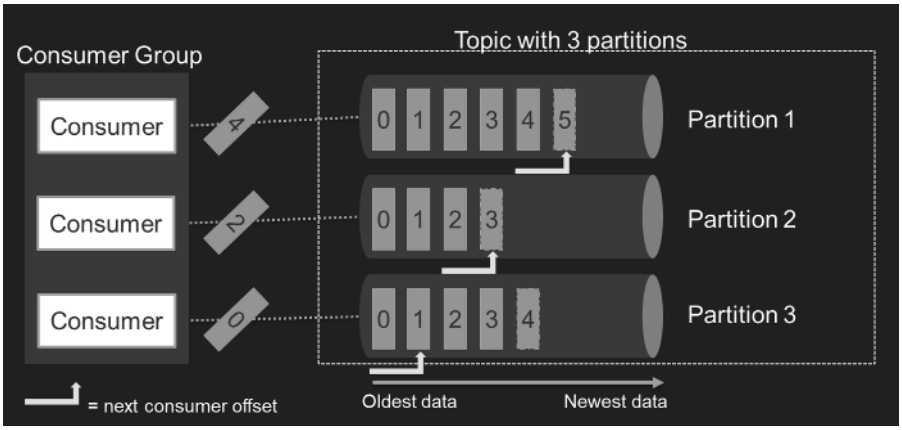
Consumer 2, associated with partition 2, operates in tandem, perhaps beginning its read sequence from offset 1, if it has previously consumed the record at offset 0. Similarly, consumer 3 starts from its respective offset in partition 3. The consumers work independently and in parallel, yet cohesively, within the boundaries of their partitions. This parallel processing allows Kafka to manage vast streams of data, providing consumers with the ability to process data sequentially and in real time, ensuring that every message is acted upon swiftly, a key attribute for applications like live-tracking systems or real-time analytics.

Let's understand the concept of a consumer group with a simple everyday example. Imagine a bustling café with a long queue of orders. A consumer group is like a team of baristas, where each member is responsible for making specific types of drinks to ensure that all orders are fulfilled quickly and no two baristas make the same drink at the same time. In Kafka, each barista represents a consumer, and each type of drink corresponds to a partition within the topic. The consumer group ensures that all orders (messages) are processed, and by dividing the work (partitions) among the team members (consumers), it increases the speed and efficiency of order (message) processing.

Consumer groups are vital for Kafka's scalability, allowing multiple consumers to read in parallel and thereby handle a high volume of data. Additionally, if a consumer fails, Kafka can reassign the partitions to other consumers in the group, maintaining continuous processing without data loss. This mechanism is essential for distributed systems that require fault tolerance and scalability.

Figure 1.20 illustrates the anatomy of a Kafka read.

FIGURE 1.20 The anatomy of a Kafka read



There are three different ways you can read data from Amazon Managed Streaming for Apache Kafka (MSK) topics:

- Using a consumer application:
  - This is the most common approach. You develop a consumer application specifically designed to read data from your MSK cluster. This application can be written in any programming language with Kafka client libraries available (e.g., Java, Python, etc.).
  - The consumer application connects to your MSK cluster using the bootstrap servers and utilizes the Kafka Consumer API to subscribe to specific topics.
  - Once subscribed, the consumer will receive a stream of messages from the chosen topics and process them as needed.
- Using streaming services: Similar to writing data, AWS provides services that can consume data from MSK and deliver it to other destinations for further processing or storage.
  - **Kinesis Data Streams**: You can configure Kinesis Data Streams to act as a consumer for your MSK topics. The data will be continuously streamed from MSK to Kinesis, where you can then process it using Kinesis applications or deliver it to other AWS services like S3 or Redshift.
  - **Amazon EMR**: If you're using Amazon EMR for big data processing, you can configure your EMR cluster to directly read data from MSK topics. This allows you to leverage EMR's capabilities like Spark or Hadoop to analyze the streaming data.
  - **AWS Lambda**: You can develop Lambda functions that subscribe to MSK topics and trigger upon receiving new messages. This allows for serverless processing of the streaming data within the Lambda environment.

- Monitoring tools:
  - Amazon CloudWatch provides integration with MSK, allowing you to monitor the health and performance of your cluster. While not directly reading data for processing, CloudWatch can display message throughput and consumer lag for each topic, helping you understand data flow within your MSK cluster.

Amazon MSK—Operating Modes and Key Features

Amazon MSK, a fully managed Apache Kafka service, offers two distinct deployment options to meet varying operational requirements: MSK Provisioned and MSK Serverless.

MSK Provisioned provides tiered storage for cost-effective scalability, allowing users to manage their Kafka clusters with greater control. This option is ideal for workloads with consistent and predictable demand, where manual environment control is preferred.

MSK Serverless automatically handles upgrades and partition rebalancing, significantly reducing management overhead. It introduces an adaptable model where the cluster scales automatically with application usage, making it suitable for unpredictable or variable workloads. MSK Serverless also features Graviton Support, enhancing both performance and cost efficiency.

Both options prioritize security, ensuring data protection across the platform. MSK Connect enables automatic scaling of Kafka clusters to meet demand, while providing comprehensive monitoring tools for system health and performance tracking.

The service integrates seamlessly with other AWS services, enhancing data workflows and analytics capabilities. It supports real-time analytics, enabling instant insights from streaming data.

Recent enhancements to Amazon MSK include:

- **Managed replication**: Amazon MSK streamlines data synchronization across different clusters or regions.
- **Glue schema registry integration**: Amazon MSK manages schema definition and versioning, improving data interaction capabilities.
- **Managed data delivery to Amazon S3**: Amazon MSK simplifies the process of storing large volumes of data, offering durable and cost-effective storage solutions.

MSK Serverless offers several unique advantages:

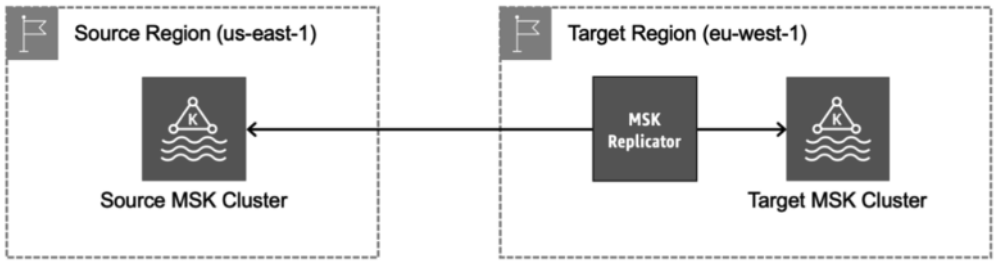
- Minimal base cost with pay-as-you-go pricing, making it cost-effective for smaller or sporadic workloads
- Automatic data replication at no additional charge, ensuring high availability without added complexity or cost
- Automatic partition placement and rebalancing, reducing operational overhead
- Compatibility with open-source Apache Kafka, maintaining expected features and performance
- Automatic distribution of resources across multiple Availability Zones, ensuring high availability without manual intervention

MSK Serverless is ideal for scenarios prioritizing ease of use, cost-effectiveness, and automatic scaling. It's particularly beneficial for users new to Kafka or with limited operational resources. MSK Provisioned, conversely, is better suited for environments requiring more manual control over consistent, predictable workloads.

Amazon MSK—Managed Replication

Amazon MSK's Managed Replication feature is like a bridge that connects your data across the world. Imagine you have a cluster of data in North Virginia, and you want to make sure that the same data can be accessed quickly and efficiently by your team in Ireland. Managed Replication does just that—it seamlessly duplicates your data from one Amazon MSK cluster to others across different regions (see Figure 1.21). This means your teams can write data in their local region and still share it globally without the wait, ensuring everyone has access to the same information at the same speed. It's not just about convenience; it's about building a system that's resilient, that can keep running smoothly even if one part faces an outage, and that respects the rules of data storage across borders.

FIGURE 1.21 MSK Managed Replication



Amazon MSK—As a Long-Term Storage Engine

Consider a financial trading platform that offers algorithmic trading services. These services rely on the ability to make quick decisions based on historical trading patterns. For example, a trading algorithm might need to access historical trade and quote data at sub-second latency to execute a high-frequency trading strategy effectively. This strategy might involve analyzing patterns in the order book over the last few milliseconds to minutes to predict short-term market movements and place trades accordingly.

In such a scenario, accessing historical data with sub-second latency is crucial because even a delay of a few milliseconds could mean a significant financial difference.

In this scenario, you'd use Amazon MSK (Managed Streaming for Kafka) as the backbone of your data pipeline. Amazon MSK provides a managed Kafka service that handles real-time data feeds at a large scale. It offers the horsepower you need to process vast streams of social media events as they happen.

However, real-time analysis is only one part of your business. You also want to offer your clients the ability to look back at historical trends and events. Here's where the traditional setup might hit a snag—storage. Storing huge volumes of data for long periods can be costly and complex to manage, especially when you need to provide similar latencies on long-term stored data as the current dataset.

This is where Amazon MSK's Tiered Storage is helpful. It's a low-cost storage tier that lets you keep a more extended history of Kafka topics without breaking the bank. Amazon MSK Tiered Storage scales virtually unlimitedly, and you therefore don't have to worry about running out of space as your data grows. This storage solution also allows you to reprocess historical data in the exact order it was received, which is critical for accurate trend analysis.

The following are some of the pros and cons of MSK's Tiered storage feature:

Pros:

- Cost-efficient storage for large volumes of data.
- Scalable to handle growth without the need for constant provisioning.
- Retains data in Apache Kafka topics for as long as needed, surpassing the typical Kafka retention periods.
- Enables data reprocessing in the exact original order, which is crucial for consistent data analysis.
- Data transfer remains within the VPC, ensuring security and speed without exposure to the Internet.
- Speeds up partition rebalancing, which can be a performance bottleneck, especially with large datasets.

Cons:

- Could introduce complexity in managing another layer of storage.
- Potentially could have a learning curve for teams not familiar with tiered storage concepts.

Amazon MSK—Handling Scalability

Amazon MSK offers two types of scalability: horizontal and vertical.

Horizontal Scaling Horizontal scaling involves adding more Kafka brokers to the existing system. When scaling horizontally, you should add brokers in multiples of the Availability Zones (AZs) being used to maintain high availability and fault tolerance. This scaling approach only supports "scaling out" operations, which means that you can add more brokers to handle increased load but not reduce them in this manner. Horizontal scaling also requires you to reassign partitions across the new set of brokers, which can be a manual or automated process depending on your setup.

Vertical Scaling Vertical scaling is about changing the size or type of the Kafka brokers you're currently using. This could mean upgrading to more powerful instances or switching to a different family of instances that better suits your workload demands.

Vertical scaling allows both "scale-up" and "scale-down" operations, giving you the flexibility to adjust the resources depending on your current needs. A significant advantage of vertical scaling is that it typically doesn't involve any cluster I/O interruption, allowing for a seamless scaling process without downtime.

Both methods are crucial for accommodating the evolving data throughput requirements in a system, and the choice between horizontal and vertical scaling depends on the specific use cases and the growth pattern of the Kafka workload.

Sizing an MSK Cluster

When implementing Amazon MSK for analytics purposes, one of the most frequent challenges customer's face is determining the appropriate cluster size. This decision is crucial, as it directly impacts both performance and cost-effectiveness.

On the one hand, overprovisioning resources by setting up an excessively large cluster from the outset can quickly deplete your budget due to unnecessary expenses. On the other hand, starting with an undersized cluster may seem cost-effective initially but can lead to performance bottlenecks when scaling becomes necessary. It's important to note that expanding a cluster is not an instantaneous process; it requires careful planning and can temporarily affect performance during the scaling operation.

To achieve optimal performance and cost-efficiency, best practices for sizing an Amazon MSK cluster involve meticulous planning and a thorough assessment of resource requirements.

When discussing cluster sizing with our customers, we typically consider the following factors:

- **Partition counts per broker**: Depending on the broker type, you should adhere to the recommended partition limits. For example, a kafka.t3.small broker handles up to 300 partitions, while larger brokers like kafka.m5.4xlarge can handle up to 4,000 partitions. Exceeding these recommendations can lead to issues such as difficulties in updating the cluster configuration or the Apache Kafka version, and potential problems with CloudWatch metrics reporting.
- **Determining the number of brokers**: Use tools like the MSK Sizing and Pricing spreadsheet to get an initial estimate for the size of your MSK cluster and its costs. However, these should be treated as starting points, and actual performance should be validated through testing.
- **Throughput optimization for larger instances**: When utilizing larger instance types such as m5.4xlarge or m7g.4xlarge, fine-tuning specific configuration parameters can significantly enhance throughput. Two key parameters to focus on are
  - `num.io.threads`
  - `num.network.threads`

    These parameters are crucial for throughput optimization because they directly control the number of threads dedicated to I/O operations and network communication,

respectively. Properly adjusting these values allows the system to more efficiently utilize the increased resources available in larger instances.

Best practices for tuning include the following:

- Prioritize increasing `num.io.threads` first, as this parameter governs the number of threads handling disk I/O operations, which are often the bottleneck in high-throughput scenarios.
- Exercise caution when increasing `num.network.threads`. While it can improve network handling capacity, setting it too high may lead to network congestion and diminishing returns.
- Incrementally adjust these parameters and monitor performance to find the optimal configuration for your specific workload and instance type.

- **Using the latest Kafka AdminClient**: To avoid issues like topic ID mismatches, especially with Kafka 2.8.0 or higher, use an AdminClient version that's up-to-date or ensure that you use the `--bootstrap-servers` flag instead of the deprecated `--zookeeper` flag for administrative tasks.
- **Building high availability**: Set up a Multi-AZ cluster with a replication factor of at least 3, and set `min.insync.replicas` to `RF-1` to maintain high availability, especially during updates or when brokers are replaced.
- **Monitoring CPU usage**: Keep the total CPU utilization under 60% to allow for load redistribution when needed. Use CloudWatch alarms to monitor CPU usage, and scale up or out based on these metrics.
- **Broker resource monitoring**: Keep an eye on individual broker loads to ensure even distribution. Use tools like Cruise Control for ongoing load management.
- **Latency monitoring**: Track produce and consume latency, as they can increase with CPU usage.
- **Disk space management**: Set alarms to monitor disk usage, and have strategies in place to prevent running out of space, such as automatic scaling, adjusting retention policies, or deleting unused topics.
- **Log recovery**: Increase `num.recovery.threads.per.data.dir` to expedite log recovery post-unclean shutdowns.
- **Memory monitoring**: Watch the `HeapMemoryAfterGC` metric to avoid memory-related outages, and adjust configurations like `transactional.id.expiration.ms` to manage memory usage effectively.
- **Broker management**: Don't add non-MSK brokers using ZooKeeper commands, as it can lead to data loss and incorrect cluster information.
- **Encryption**: Always ensure that in-transit encryption is enabled for security.
- **Rebalancing partitions**: Use `kafka-reassign-partitions.sh` to move partitions and rebalance them across new brokers when scaling the cluster.

Comparison of Streaming Services

When deciding among AWS's streaming services, it is essential to understand how each service aligns with your specific needs. Kinesis Data Streams stands out for real-time custom data processing, offering robust integrations with many AWS services, making it an excellent choice for those looking for built-in AWS support. Amazon Data Firehose serves well as a streamlined data collection and delivery pipeline, ideal for gathering and preparing data for downstream analytics engines. Amazon Managed Streaming for Kafka (MSK) is the go-to option if your current environment is built around Apache Kafka or you require extensive integration capabilities with a variety of sources and targets. This knowledge can be incredibly useful, particularly when faced with questions about service selection in an exam setting.

Table 1.1 provides a quick comparison between the streaming services. I would highly recommend using this as an aid for your certification preparation.

TABLE 1.1 Comparison of AWS streaming services

Feature	Kinesis Data Streams	Amazon Data Firehose	Amazon MSK (Managed Streaming for Apache Kafka)
Record payload	Up to 1 MB per record	Up to 1 MB per record	Configurable, default max 1 MB
Operation size	1 MB per `PutRecord`, 5 MB per `PutRecords` (up to 500 records)	5 MB per `PutRecordBatch` (up to 500 records)	Configurable, typically up to 1 MB per message
Storage duration	24 hours to 365 days (configurable)	No storage (near-real-time delivery to destinations)	Configurable, typically 7 days by default
Throughput	1 MB/sec or 1,000 records/sec per shard	Automatically scales to match input	Configurable, can handle millions of records per second
Latency	Sub-second	Buffer time configurable (minimum 60 seconds)	Sub-millisecond
Processing	Custom processing with various stream processing frameworks	Minimal built-in processing, mainly for delivery	Custom processing with Kafka-compatible tools
Characteristics	Real-time streaming Custom processing Multiple consumers Durable storage	Fully managed Easy integration with AWS analytics services Automatic scaling-Data transformation capabilities	Fully managed Apache Kafka High throughput Multi-AZ replication BYOK encryption

Feature	Kinesis Data Streams	Amazon Data Firehose	Amazon MSK (Managed Streaming for Apache Kafka)
High availability	3 AZ	3 AZ	Configurable
Delivery (deduplication)	At least once	At least once	At least once (default), Exactly once (customisable)
Guaranteed ordering	Yes	No	Yes

Batch Ingestion

We've looked at a number of AWS services that support streaming ingestion. AWS offers a variety of services for batch-ingestion (e.g., AWS Glue, Amazon EMR, and even Amazon Redshift). This chapter focuses on AWS Glue, but we'll also touch upon other large scale data transfer solutions like the AWS Snow Family and AWS DataSync. We'll also discuss how DirectConnect can improve connectivity and simplify the data transfer process adding additional layer of security in the data transfer process.

AWS Glue

AWS launched AWS Glue, a fully managed extract, transform, and load (ETL) service, in 2016, with the intent of tackling the inherently complex and laborious nature of ETL processes. Despite the existence of numerous ETL tools and partners, a significant portion of ETL tasks were still being hand-coded, leading to brittle, error-prone, and maintenance-heavy code.

By providing a simple, serverless interface, AWS Glue automates much of the time-consuming data preparation tasks for analytics, such as data discovery, cataloging, and schema mapping. It allows users to discover and connect to various data sources, transform data to fit their analytics needs using a visual interface or custom code, and load it into AWS data stores like Amazon S3, Amazon RDS, and Amazon Redshift. With AWS Glue, you can create and run ETL jobs with less infrastructure to manage and at lower costs, thereby accelerating their data integration and analytics projects.

AWS Glue has improved a lot over the past eight years, and while it still retains its serverless nature, it now provides a comprehensive suite of capabilities to streamline the processing, governance, and management of data. It features a scalable data integration engine with built-in data transformations and a flexible execution engine, allowing users to transform and move data efficiently. Monitoring tools are included to keep track of ETL jobs and workflows.

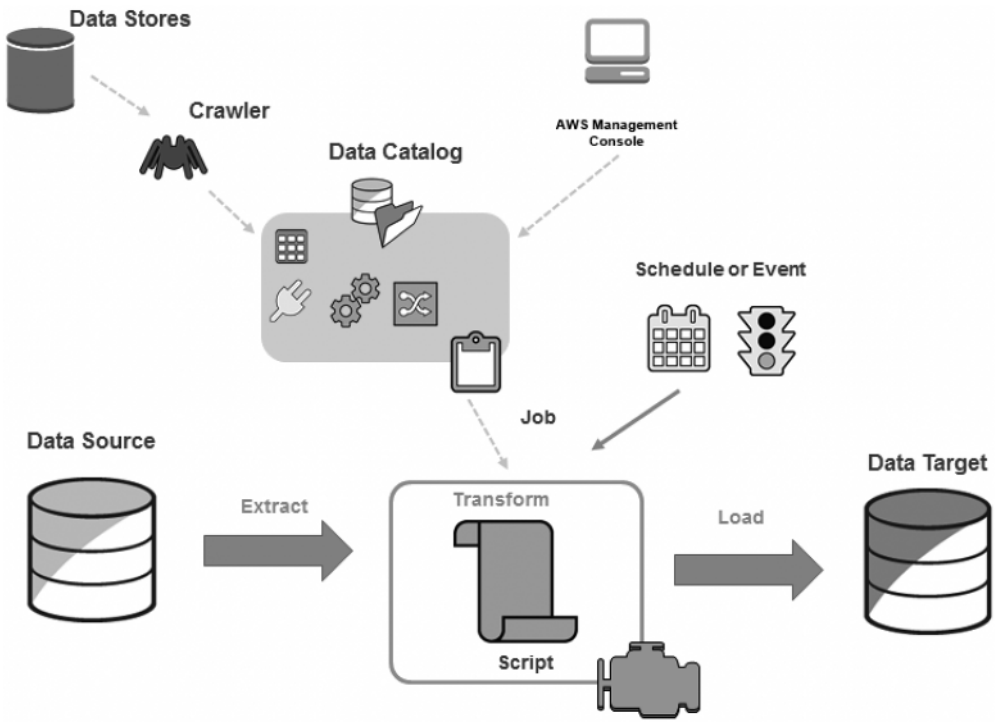
Data governance is a key aspect of AWS Glue which centralizes data management tasks through the AWS Glue Data Catalog, AWS Glue Data Quality, and AWS Lake Formation integration. This ensures that data across various AWS services is cataloged, quality is maintained, and metadata management is unified.

A comprehensive ETL tool offers a variety of connectors, and hence Glue also simplifies the connection and ingestion of data through Glue connectors and the Glue connector marketplace, allowing seamless data flow from various sources. These connectors and various interfaces facilitate the integration of diverse datasets into AWS.

Furthermore, AWS Glue enhances user productivity with persona-specific tools tailored to different roles and responsibilities within data operations. Additional productivity tools and data ops tools are available to optimize the ETL process, allowing teams to customize their workflows and enhance efficiency.

Figure 1.22 shows the different components of AWS Glue and how a Glue job operates.

FIGURE 1.22 AWS Glue flow



AWS Glue User Interfaces

AWS Glue is designed with all types of users in mind, which is why it is built to meet the needs of ETL developers, data engineers, business analysts, and data scientists with the following UI tools.

AWS Glue Studio

AWS Glue Studio is a visual interface that simplifies the creation, execution, and monitoring of ETL jobs. This powerful tool is designed to accommodate both experienced ETL developers and newcomers to data integration or Apache Spark.

Key features of AWS Glue Studio include:

- **Intuitive visual interface:** The platform offers a "boxes-and-arrows" approach, allowing users to design ETL workflows without writing complex code.
- **Versatile data management**: Users can easily transform and move data across various sources, leveraging AWS Glue's serverless, Spark-based ETL capabilities.
- **Comprehensive job run dashboard**: This feature provides a detailed overview of ETL operations, from execution metrics to resource utilization, enabling effective workflow monitoring and performance optimization.
- **Accessibility**: The user-friendly design makes it easier for professionals of all skill levels to harness the power of AWS Glue for data integration tasks.
- **Seamless integration**: As part of the AWS ecosystem, Glue Studio integrates smoothly with other AWS services, enhancing its capabilities and extending its usefulness in diverse data processing scenarios.

AWS Glue Studio empowers users to create and manage ETL jobs more efficiently, ultimately streamlining the data integration process within the AWS environment.

AWS Glue DataBrew

AWS Glue DataBrew is a visual data preparation tool specifically designed for data analysts and data scientists who need to clean and normalize data efficiently without writing code. This innovative solution addresses the often time-consuming and complex task of data preparation, allowing these professionals to focus more on analysis and insights rather than data wrangling.

Key features tailored for data analysts and scientists include:

- **Visual interface**: An intuitive, no-code environment that allows direct interaction with data from various sources, including data lakes, warehouses, and databases.
- **Built-in transformations**: Over 250 ready-to-use transformations that enable quick and easy data cleansing and normalization without requiring programming skills.
- **Data profiling**: Built-in capabilities to assess and profile data quality, helping analysts quickly understand their datasets.
- **Automation**: The ability to save and systematically apply transformations to new datasets, streamlining repetitive tasks.
- **Serverless architecture**: Eliminates the need for infrastructure management, allowing data professionals to focus on their core competencies.
- **Usage-based pricing**: Enables scalability without upfront costs, making it accessible for projects of all sizes.

AWS Glue DataBrew significantly reduces the time spent on data preparation. This allows these professionals to dedicate more time to extracting valuable insights and driving data-informed decisions within their organizations. Whether working on small-scale projects or large enterprise initiatives, data professionals can leverage DataBrew to enhance their productivity and effectiveness in the data preparation phase of their workflows.

AWS Glue Notebooks

AWS Glue Studio's notebooks are tailored for data engineers and ETL developers who prefer an interactive, code-centric approach to creating ETL jobs. This feature combines the familiarity of Jupyter Notebooks with the power of AWS Glue's serverless ETL capabilities, offering a seamless environment for those comfortable with writing and testing code.

Key features for data engineers and ETL developers include:

- **Interactive coding environment**: A Jupyter Notebook interface that allows for real-time code execution and output visualization.
- **Serverless infrastructure**: Eliminates the need for cluster management and configuration, allowing developers to focus solely on writing ETL logic.
- **AWS Glue ETL runtime consistency**: Ensures that code written in the notebook will behave identically when run as a full ETL job.
- **Seamless job conversion**: Easily transform notebook code into production-ready AWS Glue jobs without leaving the AWS Glue Studio interface.
- **Integrated development experience**: Write, test, and debug ETL code in a single environment, streamlining the development process.
- **Version control compatibility**: Save work as both notebooks and scripts, facilitating collaboration and version management.

AWS Glue Studio's notebooks cater to data engineers and ETL developers who value hands-on coding and iterative development. This tool allows them to leverage their programming skills while benefiting from the scalability and ease of management offered by AWS Glue's serverless architecture. Whether you are prototyping new ETL processes or refining existing ones, you can use the notebook interface to efficiently develop and deploy robust data integration solutions within the AWS ecosystem.

AWS Glue Interactive Sessions

AWS Glue's interactive sessions are primarily designed for data engineers and Spark developers who need to iteratively develop and test ETL jobs in a flexible, code-first environment. The key use case for these sessions is rapid prototyping and debugging of data processing tasks without the overhead of managing infrastructure or waiting for full job runs.

Key features supporting this use case include:

- **Real-time development**: Enables immediate execution and testing of ETL scripts, allowing for quick iterations and refinements.
- **Serverless Spark environment**: Provides access to Apache Spark's power without the need to manage clusters or infrastructure.

- **Flexible coding options**: Offers both a coding interface and a visual interface for constructing ETL scripts, catering to different developer preferences.
- **IDE integration**: Supports work in familiar development environments like Microsoft VS Code through Jupyter integration, enhancing productivity.
- **API access**: Allows programmatic creation and management of Spark applications, facilitating automation and integration with existing workflows.
- **Cost-effectiveness**: Pay only for compute resources used during active development, making it economical for iterative work.

AWS Glue's interactive sessions address the common challenge of lengthy development cycles in ETL job creation. They allow data professionals to quickly test ideas, troubleshoot issues, and optimize their data processing logic before committing to full production runs.

Crawlers/Classifiers

Imagine you're a librarian whose job is to organize a vast, ever-growing collection of books, which, in this case, are data files. You need to categorize them by genre, author, or content type. This is essentially what AWS Glue crawlers do for your data. They are the librarians of the data world, scanning through various data repositories, like a mix of modern e-books and classic hardcovers, which, in the data realm, are formats such as JSON, CSV, and data from relational databases housed in Amazon S3, Amazon RDS, and Amazon Redshift.

AWS Glue crawlers come with a set of pre-made classifiers—think of these as the rules of categorizing books by their covers or ISBNs. Sometimes, however, you get a unique book that doesn't fit the standard categories. For those, AWS Glue lets you create your own classifiers, like writing a new rule for a hybrid genre.

Once deployed, a crawler begins its journey through the data, applying these classifiers to understand what each file is—determining whether a column is a phone number, a date, or maybe a Social Security number. After this, it creates a detailed index card for each file, like a catalog entry, complete with all the schema details, and stores it in the AWS Glue Data Catalog. This catalog then acts as a guide for any data transformation processes you might need, ensuring that they are as accurate and efficient as possible.

Crawlers are also quite smart. When they revisit a previously scanned library section (or data store), they can spot if a new book has been added or if an existing one has changed editions (schema changes), and update the catalog accordingly, keeping everything current.

At this point, I would highly recommend you log in to the Glue console to create a crawler for your favorite data source. Perform the following step-by-step instructions:

1. Log in to the AWS Management Console and open the AWS Glue service.
2. Go to the Crawlers page and click Add Crawler.
3. Name your crawler—something that easily identifies the data it will catalog.
4. Specify the data store by choosing the type (S3, JDBC, etc.) and providing the path to your data.
5. Choose an IAM role that gives AWS Glue permissions to access your data.

6. Set up the schedule for the crawler. You can run it on demand or set it to crawl at regular intervals.
7. Configure the output by specifying the database in the AWS Glue Data Catalog where you want the tables to be created.
8. Review the settings, make any necessary adjustments, and then click Finish to create the crawler.
9. Run the crawler by selecting it from the list and clicking Run Crawler.
10. Monitor the crawler's progress and, once it's finished, check the AWS Glue Data Catalog to see your newly created tables.

Remember, as your data evolves, you might need to tweak the classifiers or adjust the crawler's schedule to keep your data catalog up to date. It's like the library's catalog that needs constant updates because new books keep arriving!

Data Catalog

The AWS Glue Data Catalog serves as a centralized metadata repository in the cloud for each AWS account, unique to each region. It's designed to scale, organizing data into databases and tables that represent your structured or semi-structured data stored across various services. It's essential for managing metadata from different systems, aiding in consistent data querying and transformation. It works hand-in-hand with AWS IAM and Lake Formation to fine-tune access, maintaining tight data security and compliance. For detailed governance features, including auditing with CloudTrail, check out the AWS documentation. The Data Catalog integrates with a host of AWS services, including Athena, Redshift Spectrum, and EMR, and is compatible with the Apache Hive metastore.

AWS Glue ETL and AWS Glue Jobs System

AWS Glue leverages the metadata stored in the Data Catalog to streamline ETL tasks by automatically generating scripts in Scala or PySpark with custom AWS Glue extensions. These scripts can be customized to perform a variety of data processing tasks, such as transforming a CSV file into a structured format suitable for analysis and storage in a service like Amazon Redshift.

AWS Glue offers a robust jobs system to manage your ETL workflow. You can automate your data processing by scripting AWS Glue jobs that handle the extraction, transformation, and relocation of your data. These jobs can be set to run on a schedule, in sequence, or in response to certain triggers, such as the arrival of new data. Detailed monitoring of these jobs ensures smooth execution, while the Jobs API allows for programmable interaction with the AWS Glue Jobs system.

This book is not a deep-dive on AWS Glue. I would highly recommend visiting the AWS documentation for in-depth guidance on AWS Glue's ETL capabilities and how to program Spark ETL scripts, including job monitoring and the API.

Streaming ETL in AWS Glue

AWS Glue facilitates ETL processes for streaming data, utilizing the Apache Spark Structured Streaming engine. It can handle data ingestion from sources like Amazon Kinesis, Apache Kafka, and Amazon MSK, supporting near-real-time data cleaning and transformations.

AWS Glue Streaming also supports a variety of data targets, including but not limited to Amazon S3, Amazon Redshift, MySQL, PostgreSQL, Oracle, MS SQL Server, Snowflake, Apache Iceberg, Delta, and Apache Hudi, any database connectable via JDBC, and a number of AWS Glue Marketplace connectors.

Glue Streaming is suited for various real-time data processing scenarios, including near-real-time analytics, fraud detection, social media and sentiment analysis, IoT data processing, clickstream analysis, log monitoring, recommendation systems, and handling of event-driven data workloads. It allows for precise schema management and also supports schema auto-detection during streaming ETL jobs, accomodating both AWS Glue-specific and Apache Spark native transformations for comprehensive data processing.

If you are preparing for a certification exam, I would highly recommend you go through the following tutorial, which walks you through the process of creating a streaming ETL job using AWS Glue studio: `https://docs.aws.amazon.com/glue/latest/dg/streaming-tutorial-studio.html`.

You will likely see questions on the exam that ask about the best streaming service given a particular scenario. We've already covered Kinesis and Kafka, so you need to know when AWS Glue Streaming would be more suitable.

AWS Glue Streaming is particularly useful when you're already using AWS Glue or Spark for batch processing and want to leverage your existing setup for real-time stream processing without learning a new framework. It's also beneficial when you need a unified service to manage batch and real-time workloads, have complex transformations to perform, prefer visual job building, or are working with transactional data lakes like Apache Iceberg, Apache Hudi, or Delta Lake. Conversely, you'd directly use MSK or Kinesis when you need specialized streaming or queue management that might not require the additional ETL capabilities of AWS Glue.

AWS Glue Data Quality

AWS Glue Data Quality, utilizing the DeeQu framework, allows for advanced monitoring and measurement of data quality, which is essential for reliable business decisions. It uses DQDL for rule definition, offering machine learning for anomaly detection, and simplifies rule creation with auto recommendations.

This serverless solution supports data across AWS Glue and the Data Catalog, ensuring comprehensive data integrity with minimal setup. AWS Glue Data Quality comes with more than 25 out-of-the-box DQ rules to start from, allowing you to create the rules that suit your specific need, and helps you track bad data by identifying exact records that might have caused your overall data quality scores to drop.

How Does Glue Data Quality Work?

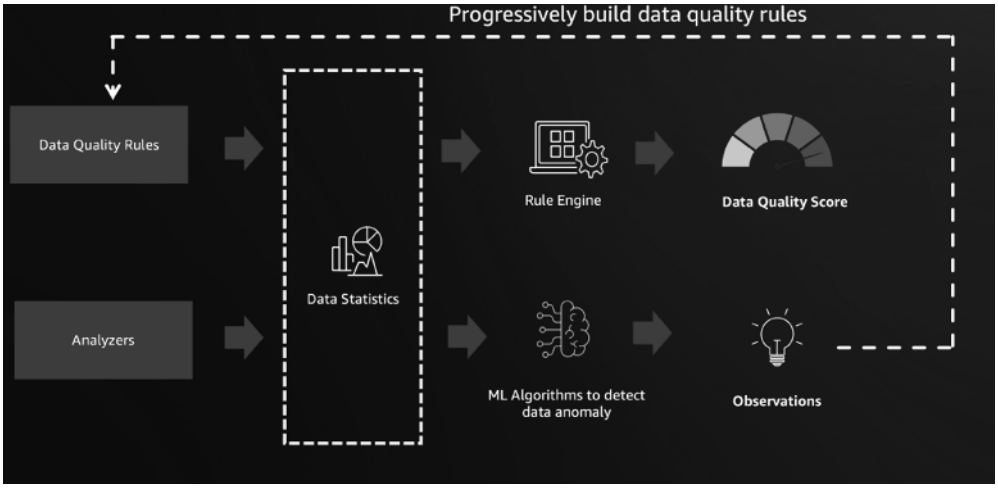
AWS Glue helps check if your data is up to scratch by keeping track of useful statistics. Let's say you want to know how many different values are in your dataset and if that number matches what you expect. AWS Glue does that by calculating these unique values and checking them against the standards you've set.

It's not just a one-off thing, either. AWS Glue keeps an eye on these stats continuously, which allows it to spot unusual trends. Think of it like it's getting smarter over time, learning what's normal for your data and pointing out anything odd. These little nudges, or "observations," are like hints from AWS Glue's machine learning brain, suggesting new rules to keep your data in line with what you're looking for.

To get the most out of it, you should run your checks regularly, like every hour or every day. If you're a bit hit-and-miss with when you run these jobs, the insights you get might not be as helpful.

Figure 1.23 shows the typical data flow with Glue Data Quality and indicates how the observations can lead to building better data quality tools over time.

FIGURE 1.23 AWS Glue Data Quality flow



AWS Glue Data Quality offers two unique entry points for the Data Catalog and ETL jobs, accommodating various data sources and integration needs, with support for dynamic and ML-based rules for nuanced data quality analysis.

- **Data Quality for AWS Glue Data Catalog:** This AWS Glue Data Catalog's Data Quality feature offers a straightforward approach for those without coding experience, like data stewards and business analysts, to establish data quality guidelines. This is ideal for scenarios where you need to assess data quality in already cataloged datasets. This feature is accessible through the AWS Glue management console and its APIs.

- **Data Quality for AWS Glue ETL jobs**: This feature is for proactive data quality tasks, to help you filter out bad data before loading datasets into a data lake. The typical use cases are when you want to incorporate data quality tasks into your ETL jobs, write code that defines data quality tasks in ETL scripts, or manage the quality of data that flows in your visual data pipelines. Typically, these data quality operations are performed from AWS Glue Studio, AWS Glue Studio notebooks, and AWS Glue interactive sessions. You also have the option to use AWS Glue libraries for ETL scripting and using AWS Glue APIs.

Change Data Capture with Glue Bookmarks

Quite often we need to capture data from sources in a variety of ways, such as full data extracts from the source or identifying delta records. Identifying delta records is often complicated because you need to ensure that only the updated data from the data source is captured, hence simplifying the remaining ETL pipeline and improving overall performance and reduce the total end-to-end runtime. AWS Glue uses Glue bookmarks to track the processed data from previous runs of an ETL job by persisting the state information from the job run.

This persisted state information is called a *job bookmark*. Job bookmarks help AWS Glue maintain state information and prevent the reprocessing of old data. Job bookmarks allow you to process new data when rerunning on a scheduled interval. A job bookmark is composed of the states for various elements of jobs, such as sources, transformations, and targets. For example, your ETL job might read new partitions in an Amazon S3 file. AWS Glue tracks which partition(s) the job has already processed successfully to prevent duplicate processing and duplicate data in the job's target data store.

Job bookmarks are implemented for some of the most common file formats on Amazon S3, including JSON, CSV, Apache Avro, XML, Parquet, and ORC.

While AWS Glue's job bookmarking feature works seamlessly for Amazon S3 input sources, it has certain limitations when dealing with relational data sources accessed through JDBC connections. The bookmarking mechanism's ability to capture changes (deltas) in relational sources is constrained by specific conditions.

For relational data sources, job bookmarks are only supported if the table's primary keys are sequentially ordered. This sequential ordering is crucial for the bookmarking system to accurately track changes. Additionally, job bookmarks can detect new rows added to the table, but they cannot identify updated rows. This limitation arises because bookmarks rely on primary keys, and if a primary key already exists in the target, AWS Glue cannot distinguish whether the corresponding row has been modified in the source.

It's important to note that at the time of writing this, the AWS Glue team was actively working on addressing this limitation, aiming to enhance the bookmarking capabilities for relational data sources.

Conversely, when Amazon S3 serves as the input source for AWS Glue jobs, the job bookmarking mechanism operates efficiently. In this scenario, the bookmarks track the last modified timestamps of the objects (files) in the input data source. During subsequent job runs, AWS Glue compares the current timestamps of the input files against the bookmarked timestamps from the previous run. Any input file with a more recent modification timestamp

is flagged for reprocessing, ensuring that only the changed files are subjected to additional computations.

This intelligent approach optimizes resource utilization and minimizes redundant processing, as unchanged files are skipped during job execution. The bookmarking feature for Amazon S3 input sources streamlines the data processing workflow, enabling efficient handling of incremental updates to the input data while avoiding unnecessary overhead.

Amazon Data Migration Service

AWS Data Migration Service (DMS) is a versatile tool and has already been used to migrate millions of databases to AWS. While originally built for database migration, it now supports a wide array of data sources and targets, which can be classified into different categories, such as relational databases, NoSQL, analytics, and data warehouses. This makes it a great tool for data ingestion as well.

- **Relational databases**: AWS DMS can handle a range of popular relational databases both as sources and targets. This includes heavyweight databases such as Oracle, SQL Server, PostgreSQL, MySQL, MariaDB, and SAP ASE. It also supports databases hosted on cloud platforms like Amazon RDS, GCP MySQL, and SQL Azure, as well as on-premises databases. For databases running on AWS, it can interact with Amazon Aurora and those running on Amazon EC2 instances.
- **NoSQL**: When dealing with NoSQL databases, AWS DMS supports MongoDB and Amazon DocumentDB as sources, allowing for migrations from these flexible schema systems. For targets, apart from MongoDB and DocumentDB, it can also migrate data to Amazon DynamoDB, which is designed to handle large-scale, high-traffic NoSQL data, and Amazon Neptune, which is a fast, reliable graph database service. Moreover, DMS can interact with Amazon Elasticsearch Service for search and analytics purposes and Amazon ElastiCache for in-memory data caching, which is often used to enhance the performance of existing databases.
- **Analytics**: From an analytics standpoint, AWS DMS can move data into Amazon S3, which serves as a centralized storage that can be used for big data analytics, and AWS Snowball, a data transport solution for transferring terabytes to petabytes of data into and out of AWS. These services are essential for handling vast datasets required for analytics workflows.
- **Data warehousing**: For data warehousing needs, which are integral to analytics because they provide centralized storage and retrieval of large volumes of structured data optimized for query and analysis, DMS supports a range of targets. These include traditional data warehouses such as Oracle, SQL Server, Greenplum, Teradata, Vertica, and Netezza, as well as modern cloud-based solutions like Amazon Redshift and Azure Synapse. Amazon Redshift, in particular, is a fast, scalable data warehouse that makes it simple and cost-effective to analyze all your data across your data warehouse and data lake.

- **Streaming data:** In terms of real-time analytics and streaming data, AWS DMS can target Amazon Kinesis Data Streams, which enables real-time processing of streaming data at scale, and Amazon Managed Streaming for Kafka (Amazon MSK), which is a fully managed service that makes it easy to build and run applications that use Apache Kafka to process streaming data.

The ability to migrate and replicate data across such a diverse range of sources and targets makes AWS DMS a powerful tool in the analytics space. It enables organizations to aggregate data from disparate databases into a central repository where it can be used for comprehensive analytics. The flexibility of DMS allows it to fit into various analytics architectures, whether you're moving legacy database data into a modern data warehouse for historical analysis or streaming real-time data into an analytics platform for instant insights. This capability to integrate various data forms into a single coherent system is vital for data-driven decision-making and is at the heart of modern analytics strategies.

Key Components of DMS

AWS Database Migration Service (DMS) is a tool that helps you move your data to AWS. It's like a moving truck for your data, taking it from where it is now to where you want it to be in AWS. Here's a breakdown of the different parts that make up AWS DMS:

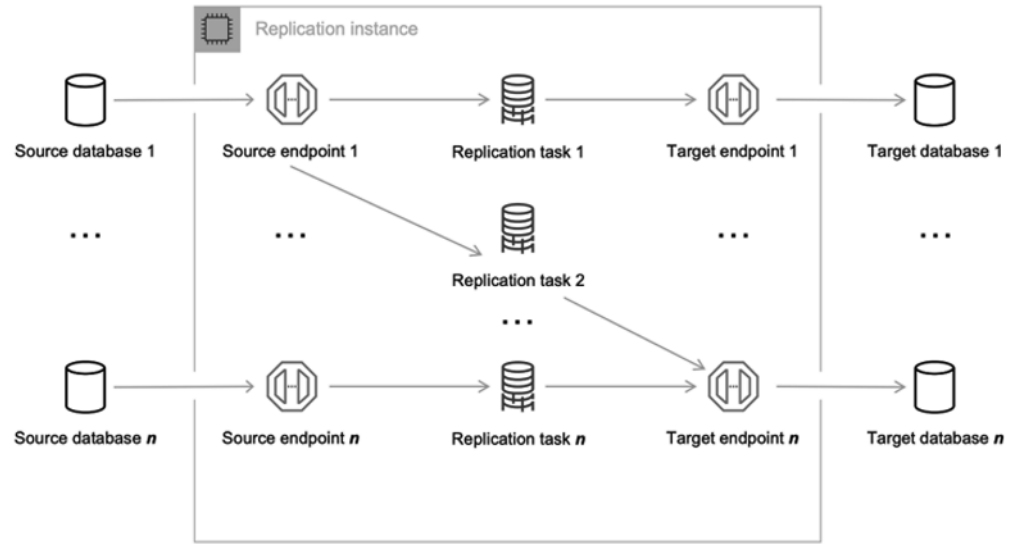
- **Finding databases**: DMS can discover the databases you have and help you figure out what's there.
- **Changing database formats**: If your databases speak different languages (e.g., Oracle or SQL), DMS can translate them so they understand each other.
- **The replication machine**: This is like the engine of the moving truck. It's where the actual moving happens.
- **Starting and ending points**: These are like the old and new homes for your data. DMS connects to where your data is coming from and where it's going to.
- **The moving task**: This is the to-do list for your move, telling DMS exactly what to move and how to move it.

Let's look at the different components that help you achieve all of the aforementioned functionalities:

- **Discovering your databases with DMS Fleet Advisor**: DMS Fleet Advisor takes a look at all your database systems and gives you a report. It works with different types of databases, such as Microsoft SQL Server and Oracle, and you don't have to install it everywhere—just in one place.
- **Changing database schemas with DMS Schema Conversion**: When you're moving from one type of database to another, DMS Schema Conversion helps you change the database design and code to fit the new place. It's like making sure your furniture will fit through the door of your new house.

- **The replication instance:** The replication instance is where the magic happens. It's a special place in AWS that does the heavy lifting of moving your data. You can pick different sizes of this replication instance depending on how big your move is. Figure 1.24 shows the workings of a replication instance. Note that it runs multiple replication tasks to connect between a variety of source and target endpoints.

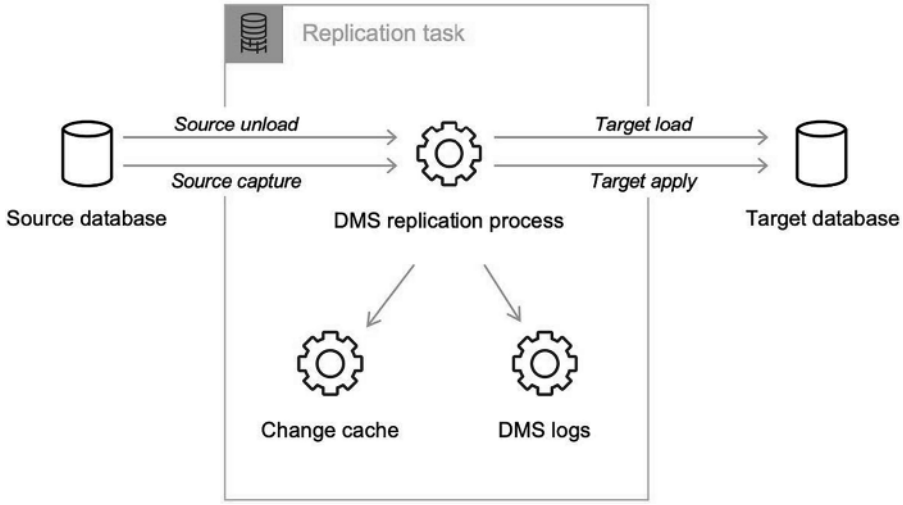
FIGURE 1.24 AWS DMS replication instance running replication tasks



- **Storage and high availability:** Your replication instance has storage space for your data during the move. If you're moving a lot, you might need more space. DMS also makes sure that there's a backup ready in case something goes wrong, so your data is safe.
- **Endpoints:** Endpoints are the old and new homes for your data (i.e., they are like the addresses for your old and new data homes). You tell DMS where to pick up your data and where to drop it off. DMS checks to make sure it can get into both places before starting the move.
- **The replication task:** The replication task is your moving plan. It tells DMS what data to pack up, how to handle special items, and where everything goes. You can also tell DMS to keep an eye on your data as it moves to make sure nothing gets lost or broken. Figure 1.25 shows the inner workings of a replication task and how it uses the Source Unload/Target load mechanism to move data from the sources to the targets while maintaining change caches and DMS logs.

In simple terms, AWS DMS is like a super-fast and simple moving service for your databases, helping you plan the move, pack up your data, and get it safely to its new home in AWS.

FIGURE 1.25 Functions of a DMS replication task



Ongoing Replication, or Change Data Capture (CDC) with DMS

AWS DMS supports three migration options, each having its own nuances.

- Existing data
  - AWS DMS creates files or tables in the target database.
  - AWS DMS populates the tables with data from the source.
  - Existing data migration is available via 'Migrate Existing Data' in the Console or 'Full Load' via API.
  - AWS DMS captures changes on the source during migration.
  - Once the initial migration completes, changes are applied to the target as units of completed transactions.
  - Full data migration plus ongoing replication is available via 'Migrate Existing Data and Replicate Ongoing Changes' in the Console or 'full-load-and-cdc' via API.
- Replicate changes only
  - AWS DMS reads the recovery file on the source database.
  - AWS DMS optimizes change replication by buffering and batch-applying grouped transactions to the target.
  - Ongoing changes replication is available via 'Replicate Data Changes Only' in the Console or 'cdc' via API.

For target preparation, you can choose between the following options with AWS DMS:

- **Do nothing:** In Do Nothing mode, AWS DMS assumes target tables are pre-created. In Full Load mode or Full Load Plus CDC mode, ensure that the target tables are empty before starting the migration.
- **Drop Tables On Target:** In Drop Tables On Target mode, AWS DMS drops the target tables and re-creates them before starting the migration. This ensures that the target tables are empty when the migration starts.
- **Truncate:** In Truncate mode, AWS DMS truncates all target tables before the migration starts.

From an exam perspective, it is important to know that DMS can actually be used to load data to Amazon S3, Amazon OpenSearch, Amazon Kinesis Data Streams, Amazon Kafka, and Amazon Redshift (Provisioned and Serverless). In addition, it gives better flexibility when it comes to capturing delta records.

AWS DataSync

While working with AWS customers, we see that customers want to migrate more and more workloads to the cloud, and the size of the datasets gets increasingly larger as well. I have yet to come across an organization where datasets are decreasing, and hence, there's a need for tools to help move the data quickly and efficiently. There are many open-source tools available for little or no cost that can move data to AWS.

Transferring a few files here and there is pretty easy to do. But when you start looking at copying large amounts of data, there are a lot of factors to consider. Many of our customers initially look at using DIY tools, which makes sense because many of these DIY tools are often "free" to use. It's also possible to write your own tools to do this on your own, but it takes a significant amount of work.

Tools that may initially appear to be "free" can actually cost enterprises a lot of money—and time—when applied to larger-scale migrations.

Most importantly, you don't want to find out that the files that took hours or days to transfer are corrupted, especially when you have a tight project timeline. This is where tools like AWS DataSync can help by performing data integrity checks both in transit and at rest to ensure data written to your destination matches the data read from your source.

AWS has a habit of always working backward from customer problems. After hearing from our customers about some of the challenges they were facing with trying to transfer data into AWS, we realized there was an opportunity to help them by removing a lot of the undifferentiated heavy lifting that comes with building your own migration tools, which is why we built AWS DataSync, a fully-managed service for online transfer of data to and from AWS.

Performance is critical to any migration tool, and DataSync has been built for performance. In fact, based on our internal tests, it runs up to 10x faster than common open-source tools, utilizing compression, parallel transfer, and other optimizations to get your

data to the cloud as fast as possible. In particular, DataSync was purpose built to handle the challenge of migrating millions or even billions of small files, a task that can be particularly difficult for do-it-yourself solutions. We also wanted to make DataSync easy to use by allowing customers to get up and running quickly, focusing on the task of getting their data to the cloud.

Security is and has always been our top priority at AWS, and we built DataSync to transfer data securely and reliably. DataSync encrypts data in-flight and supports encryption at rest on S3 and EFS. In addition, DataSync provides a number of options for validating your data, making sure that all data was transferred successfully.

DataSync integrates with common cloud management tools such as CloudWatch, CloudTrail, IAM, and others.

We also wanted DataSync to be cost-effective since customers want to keep the costs minimum. While exact pricing is not part of exam questions, you are often asked about finding a solution that is simple or cost-effective. DataSync can offer both simplicity and lower costs.

The following are some of the key characteristics of AWS DataSync:

- **Supports fast data transfer**: Up to 10 times faster than common open-source tools, using compression and parallel transfer
- **Easy to use**: Supports NFS and SMB protocols, installable on-premises or in-cloud
- **Secure and reliable**: Encryption in transit, encryption at rest on S3, EFS, and FSx, and built-in verification
- **Cloud integrated**: S3, EFS, FSx, CloudWatch, IAM, CloudTrail, etc.
- **Cost-effective**: 1.25 cents per GB transferred, pay only for the data copied

How Does DataSync Work?

The DataSync software agent connects to your Network File System (NFS) and Server Message Block (SMB) storage, so you don't have to modify your applications. To get started, all you have to do is deploy the DataSync agent, connect it to your file system, select your AWS storage resources, and start moving data between them. DataSync is managed centrally from the AWS Management Console.

Since security and logging are key parts of data migrations, DataSync also integrates with other key AWS tools such as: AWS Key Management Service (KMS), AWS Identity and Access Management (IAM), AWS CloudTrail, and AWS CloudWatch.

From an architectural standpoint, DataSync has two major components:

- DataSync Discovery
- DataSync Transfer

The Deep-Dive into DataSync Discovery

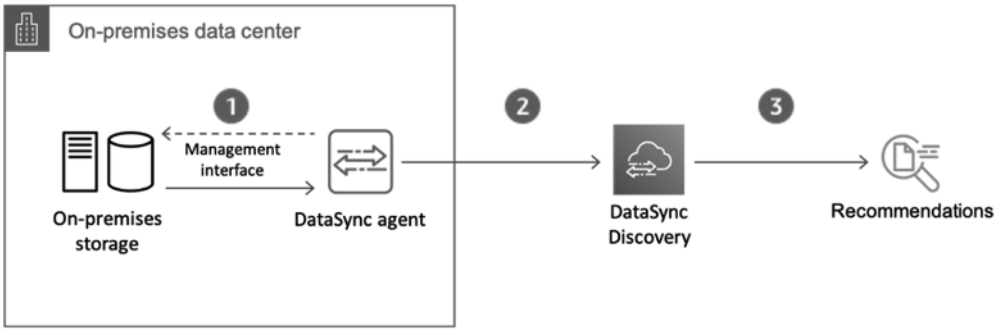
You can start the data migration process with AWS DataSync by deploying the DataSync agent, a software application that facilitates the secure transfer of data between on-premises

storage systems and AWS services. The agent software communicates with the local storage management system in a secure way, using port 443. After establishing a successful connection, the agent starts a discovery process to identify and understand the details and structure of the on-premises storage system.

Following the data collection phase, the DataSync agent transmits the gathered dataset descriptors to the DataSync Discovery service. This transfer occurs over a secure, public service endpoint, ensuring that the metadata related to the storage environment is relayed to AWS without compromising security.

DataSync Discovery, utilizing the ingested metadata, performs an analysis to identify the most suitable AWS storage targets for the customer's dataset. This recommendation process is informed by the specific requirements and configurations of the on-premises storage system. DataSync Discovery serves as a decision-support component, providing users with insights into optimal AWS storage solutions to facilitate an efficient migration pathway tailored to the customer's infrastructure. Figure 1.26 shows the three steps of the DataSync Discovery process.

FIGURE 1.26 DataSync Discovery



DataSync Transfer

DataSync can be used to transfer data between on-premises storage and AWS, between AWS storage services, and even between cloud storage systems and AWS storage services.

Transferring Data Between On-Premises Storage and AWS

The architecture diagram in Figure 1.27 illustrates how AWS DataSync facilitates the transfer of data from on-premises storage to AWS cloud storage services.

Here's how the process unfolds:

1. The journey begins with the DataSync agent, a specialized software deployed within your on-premises environment. This agent securely connects to your local storage systems' management interface, typically through the well-established port 443, ensuring a secure path via TLS (Transport Layer Security) encryption. This is similar to packing your valuables into a secure, armored truck for transportation.

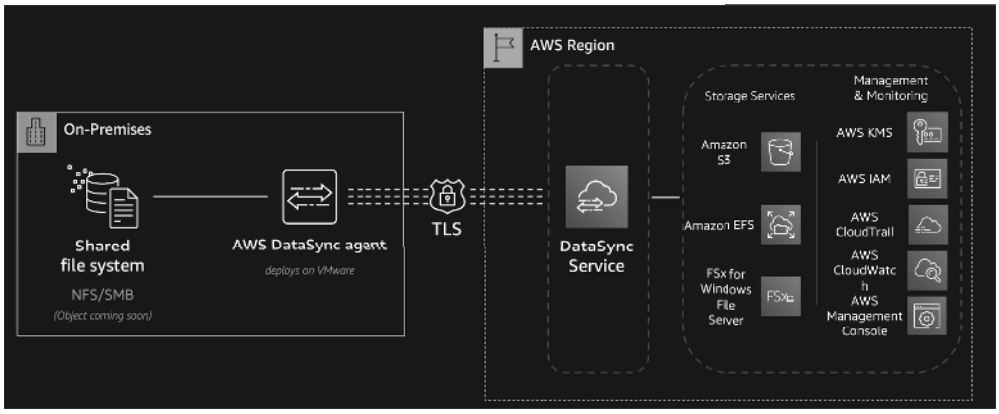
2. From there, the data embarks on its journey to the AWS Region, where the cloud resources are waiting. The DataSync agent transfers the data across the Internet, ensuring that it remains encrypted and secure during transit.
3. Upon reaching the AWS Region, the data is then distributed to the appropriate AWS storage services based on the recommendations. These services could include Amazon S3 buckets for object storage, Amazon EFS for file storage, or Amazon FSx or Windows File Server for specific Windows-based storage needs.

This architecture is particularly relevant for several use cases, such as:

- **Disaster recovery**: Regularly syncing critical data to AWS, ensuring that in the event of a disaster, your data is securely stored and readily available in the cloud.
- **Data archiving**: Moving large amounts of infrequently accessed data from on-premises storage to the cloud for long-term retention and compliance.
- **Content distribution**: Syncing content repositories to the cloud to serve global audiences with lower latency and higher availability.
- **Data lakes**: Aggregating data from various on-premises systems into a centralized AWS data lake for advanced analytics and machine learning.
- **Cloud migration**: As part of a broader cloud migration strategy, incrementally moving on-premises workloads and datasets to AWS services for enhanced scalability and innovation.

The AWS DataSync service streamlines and accelerates the data transfer process, simplifying what would otherwise be a complex and resource-intensive problem, allowing organizations to focus on deriving value from their data rather than the logistics of moving it.

FIGURE 1.27 Transferring between on-premises and AWS



Transferring Data Between AWS Services

The architecture diagram in Figure 1.28 represents a setup using AWS DataSync to transfer data between different AWS storage services within an AWS Region. At the core of this setup, we see the DataSync service, depicted as the central node that orchestrates the data flow. On either side of the DataSync node, there are AWS storage services represented, which could include Amazon S3 buckets, EFS file systems, or FSx for Windows File Server instances.

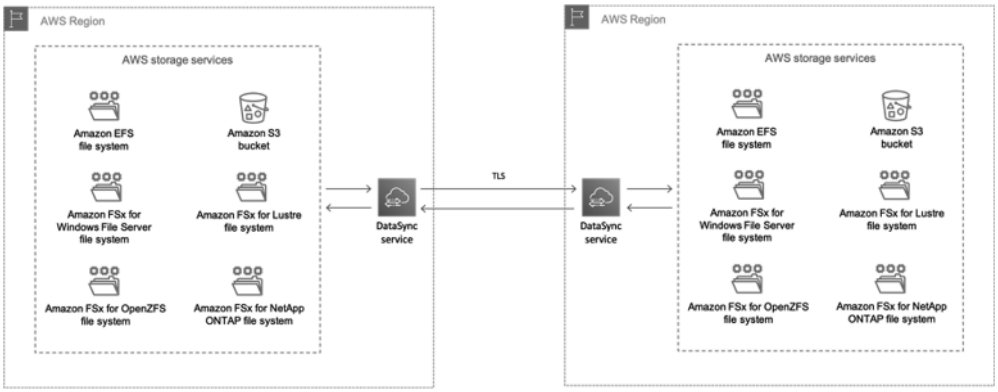
In a typical use case, you might have an application running in one part of AWS that generates large amounts of data—for example, a web application storing user data on EFS. You may want to back up this data to S3 or require processing with another service that requires the data to be in a different storage service, like FSx. DataSync automates the movement of data between these services securely and efficiently, using TLS to encrypt the data in transit, ensuring that your data is protected.

Here are some example scenarios where this architecture is applied:

- **Disaster recovery**: Replicating data across different storage services to ensure that in the event of a failure, the data remains available and the applications continue to function with minimal disruption.
- **Data lakes**: Consolidating data from various sources into a single S3-based data lake, enabling advanced analytics and machine learning processes on a centralized dataset.
- **Content distribution networks (CDNs)**: Synchronizing multimedia content across multiple storage locations to provide low-latency access to users distributed geographically.
- **Database backup and restoration**: Periodically backing up database snapshots to a secure storage service for recovery purposes, ensuring business continuity.
- **Hybrid storage environments**: Bridging on-premises storage systems with cloud storage to facilitate a gradual migration to the cloud or to extend the on-premises data storage capabilities.

By configuring DataSync within your environment, you can maintain a high-throughput, low-latency data sync mechanism, optimizing your cloud storage strategy for operational efficiency, cost, and performance.

FIGURE 1.28 Transferring data between AWS storage services



Transferring Data Between Cloud Storage Systems and AWS Storage Services

The architecture diagram in Figure 1.29 depicts the setup for AWS DataSync used to transfer data between cloud storage systems (Azure Blob Storage) and AWS storage services within an AWS Region. The process is anchored by the DataSync service, positioned as the operational node, ensuring data is transferred securely and efficiently.

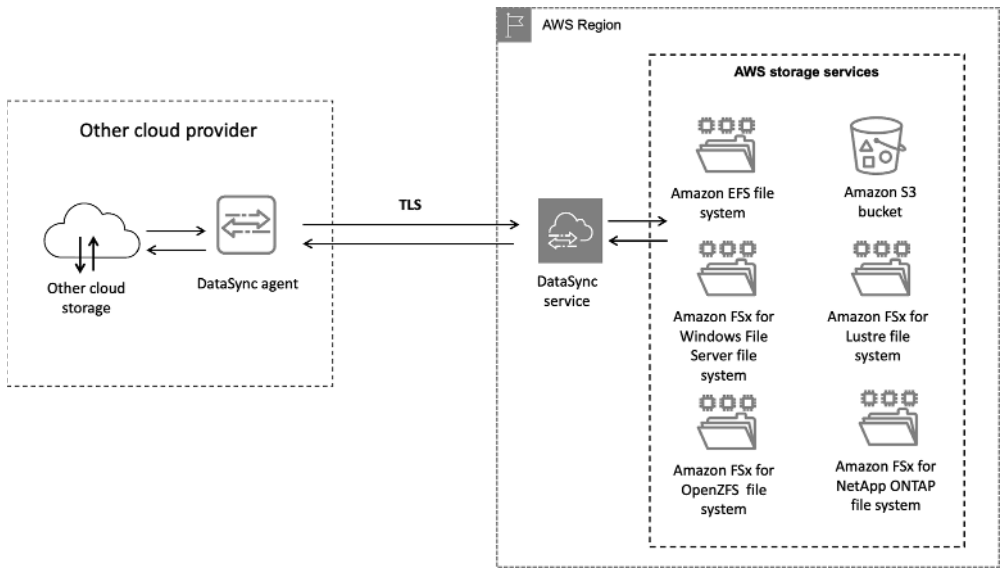
To the left of the DataSync node, we see a cloud storage system that represents the source of the data. It might be an existing AWS service, like an S3 bucket or EFS file system that you're using to store your data for various applications, or even object storage from Azure or GCP. The data from this cloud storage passes through the DataSync node, which handles all aspects of the transfer securely, thanks to the TLS (Transport Layer Security) protocol that encrypts the data in transit, adding a layer of security as it moves across the network.

On the right side of the DataSync node are the AWS storage services, which can be a variety of different AWS offerings, depending on your needs. Data might be directed to another S3 bucket, an EFS file system, or an FSx for Windows File Server. The target destination is determined based on the desired outcome of the transfer.

A typical use case for this architecture might be consolidating data from various cloud storage locations to a single data lake for analytics and processing. For instance, a company might be using multiple S3 buckets across different departments to store data and now wants to centralize this into a single S3 bucket or EFS file system for unified access and easier management. DataSync can seamlessly handle this consolidation, moving large datasets swiftly and securely, while the company pays only for the data transferred.

Another use case could involve transferring regularly updated datasets from ongoing operations to a dedicated analytics environment. Suppose a financial institution stores transaction data across various cloud storage services and wants to perform timely risk analysis. In that case, DataSync can be used to aggregate this data to a centralized storage location where complex analytics operations are executed.

FIGURE 1.29 Transferring data between cloud storage services and AWS services



AWS DataSync is a bridge that simplifies the movement of data across different storage services within the AWS ecosystem, optimizing for performance, cost, and ease of use in various data management scenarios.

Large-Scale Data Transfer Solutions

We have seen a number of different technologies that allow you to collect data from different data sources when the volumes are low to moderate. However, when you come across situations when you need to migrate large data volumes, such as when data sizes are over multiple terabytes or petabytes, transferring the data over an Internet connection could take a very long time. Those are the scenarios in which you need large-scale data transfer options like the AWS Snow Family, which offers specially designed devices that provide a secure and economical way to transfer massive amounts of data, up to petabytes, offline.

You can lease a Snow device according to your needs and easily move your data to the cloud. These devices are built to withstand even the toughest conditions, ensuring that your data is protected with high security no matter where you need to take it. With a range of devices offered by AWS, you can choose the best fit for environments where space or weight is a concern, for easy portability, or for a variety of networking needs. Whether you're working in a remote location or in a bustling city, the AWS Snow Family has a data transfer solution to match your situation.

AWS Snowcone

AWS Snowcone is a compact, tough device designed to help you move a lot of data (up to petabytes) to AWS without needing the Internet. It's super portable and can handle rough conditions, making it perfect for use almost anywhere—from remote work sites to busy offices. Snowcone is light enough for one person to carry and can even fit in a standard mailbox or be delivered by drone! Whether you're offline by sending the device back to AWS or online using AWS DataSync, Snowcone makes moving your data easy. It supports both edge computing with Amazon EC2 and local storage, even in challenging environments.

Snowcone comes in two types: one with a hard disk drive (HDD) offering 8 TB of storage, and another with a solid-state drive (SSD) with 14 TB of storage. It's about the size of a tissue box, weighs roughly 4.5 pounds, and is durable enough for any setting. Compared to its bigger cousin, AWS Snowball, Snowcone is much lighter and smaller.

For offline data transfer, you just load your data onto a Snowcone and ship it to AWS. Online transfers are smooth, too, thanks to AWS DataSync pre-installed on the device for easy data import/export with AWS storage services. Snowcone works well with file-based applications and connects to both wired and wireless networks. It's powered through a USB-C connection, compatible with standard adapters, and can even run on a battery for mobile uses.

Once your data job is done, Snowcone's built-in E Ink shipping label ensures it gets back to AWS safely. AWS takes care of all the shipping logistics, and the device is securely erased, following NIST standards after your data is transferred. Figure 1.30 shows a Snowcone device, which is built to make your big data transfers simple, secure, and efficient.

FIGURE 1.30 An AWS Snowcone device

AWS Snowball

AWS Snowball is a rugged device designed to handle large data transfers and perform computing tasks at the edge, where data is generated and used. It comes in two varieties:

- **Snowball Edge Compute Optimized**: This version is packed with computing power, making it suitable for tasks like machine learning and video analysis. It has 104 virtual CPUs, 416 GB of memory, and an optional NVIDIA Tesla V100 GPU for graphics-intensive work. It also offers 28 TB of solid-state storage.
- **Snowball Edge Storage Optimized**: This version focuses on providing massive storage capacity for large data migrations. It can offer up to 80 TB of hard drive storage or 210 TB of solid-state storage, making it perfect for moving huge amounts of data. Despite their differences, both Snowball devices are built to operate in remote or challenging environments like oil rigs or military bases.

Snowball devices are not just about storage and computing; they also offer fast and secure network connectivity, supporting speeds from 10 to 100 Gbps. All data is encrypted on the device itself to ensure security and speed up the transfer process. If you need even more power or space, you can connect multiple Snowball devices together in a cluster. Additionally, if you're using Kubernetes, you can set up an Amazon EKS Anywhere cluster on Snowball devices.

For data storage, Snowball works with Amazon S3-compatible storage, allowing applications to read and write data as if they were directly interacting with S3. For block storage needs, Snowball supports attaching volumes to EC2 instances running on the device, enabling you to develop applications in the cloud and run them on the edge without any changes.

Security is a top priority with Snowball. All data is encrypted, and the devices meet stringent standards like HIPAA and FedRAMP, making them suitable for sensitive environments. Managing Snowball is also easy, thanks to AWS OpsHub, which allows you to manage your devices without an Internet connection.

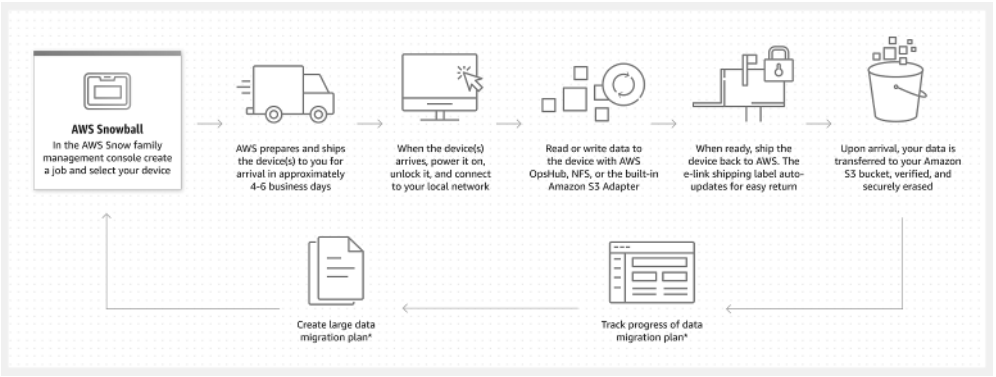
When you're ready to move your data, you can ship the Snowball device back to AWS, where your data will be securely transferred to the cloud. AWS handles all the shipping details, and with the device's innovative E-Ink shipping label, tracking is straightforward. After the job is completed, AWS ensures that the device is wiped clean according to strict standards, maintaining your data's privacy from start to finish.

Figure 1.31 shows the entire process of AWS Snowball.

AWS Direct Connect

In the AWS Data Engineering exam, you may encounter scenarios related to AWS Direct Connect (see Figure 1.32), a cloud service solution that simplifies the establishment of a dedicated network connection from your on-premises infrastructure to AWS. Understanding AWS Direct Connect and its use cases is crucial for the exam and proper AWS architecture.

FIGURE 1.31 The AWS Snowball process



AWS Direct Connect allows you to create private connectivity between AWS and your data center, office, or colocation environment. This dedicated connection can often reduce network costs, increase bandwidth throughput, and provide a more consistent network experience compared to internet-based connections.

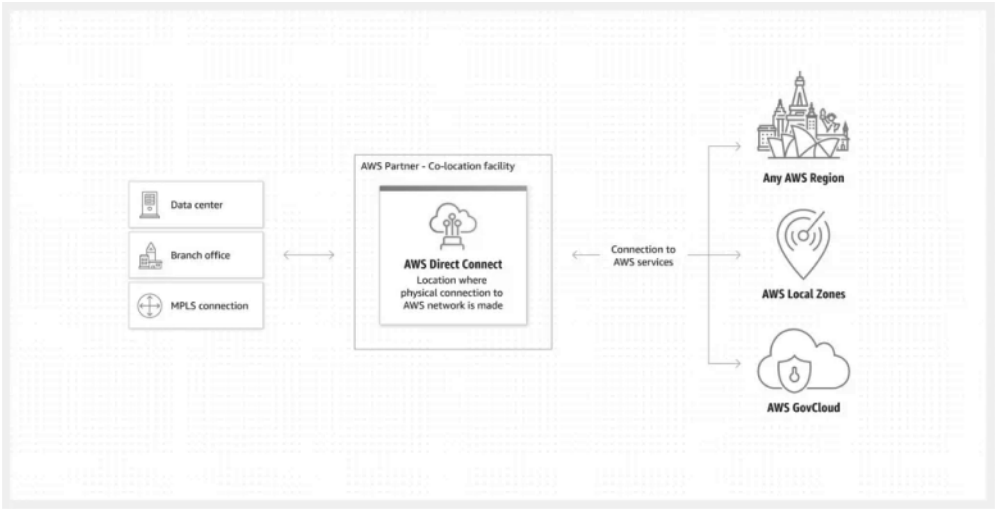
As shown in Figure 1.32, AWS Direct Connect enables you to establish a dedicated network connection between your network and one of the AWS Direct Connect locations. Leveraging industry-standard 802.1q VLANs, this dedicated connection can be partitioned into multiple virtual interfaces. This partitioning allows you to use the same connection to access public resources, such as objects stored in Amazon S3, using public IP address space, and private resources, like Amazon EC2 instances running within an Amazon Virtual Private Cloud (VPC) using private IP space, while maintaining network separation between the public and private environments. These virtual interfaces can be reconfigured at any time to meet your changing needs.

The key benefits of AWS Direct Connect, which may be relevant in the AWS Data Engineering Exam, include:

- **Reduced bandwidth costs**: By establishing a dedicated network connection, you can potentially reduce the costs associated with transferring large amounts of data over the Internet.
- **Consistent network performance**: AWS Direct Connect provides a more reliable and consistent network experience compared to Internet-based connections, which can be crucial for data engineering workloads that require predictable performance.
- **Compatibility with all AWS services**: AWS Direct Connect seamlessly integrates with various AWS services, enabling you to leverage the full range of AWS offerings for your data engineering needs.
- **Private connectivity to your Amazon VPC**: AWS Direct Connect allows you to establish private connectivity to your Amazon VPC, ensuring secure and isolated communication between your on-premises infrastructure and AWS resources.
- **Elastic scaling between 1 Gbps to 10 Gbps**: AWS Direct Connect offers flexible bandwidth options, allowing you to scale your dedicated connection from 1 Gbps to 10 Gbps, accommodating your evolving data transfer and processing requirements.

In the AWS Certified Data Engineering Associate exam, questions related to AWS Direct Connect may focus on scenarios involving large-scale data transfers, secure connectivity between on-premises and AWS environments, or optimizing network performance for data-intensive workloads. Understanding the capabilities and benefits of AWS Direct Connect can help you effectively address such scenarios and make informed decisions in the exam.

FIGURE 1.32 AWS DirectConnect overview



Summary

This chapter discussed the various components of the data pipeline when it comes to ingesting data onto the AWS cloud. We looked at ingesting data from a variety of sources using AWS services that are purpose-built for streaming and batch ingestion. We looked at streaming services like Amazon Kinesis, Amazon Managed Streaming for Kafka, Managed Streaming for Apache Flink, and AWS Glue Streaming. We also discussed batch ingestion using AWS Glue and AWS Data Migration Service, and we touched upon some large-scale data transfer solutions like the Amazon Snowball Family of devices for data transfers that are too huge to be transmitted over a standard Internet connection, and AWS Direct Connect for direct connectivity between your data center and AWS.

While we touched on the theoretical concept of different services, the AWS Certified Data Engineering Associate exam expects practical knowledge of these services. I would highly recommend that you open up these AWS services and run some of the workshops provided in the References section of this chapter.

Exam Essentials

Understand the core streaming data services and their use cases Know the differences between Amazon Kinesis Data Streams, Amazon MSK, and Amazon Managed Service for Apache Flink. Kinesis is ideal for real-time data processing with sub-second latency; MSK suits Apache Kafka workloads requiring high throughput and retention; and Managed Flink excels at complex stream processing with SQL or Java. Understanding these services' characteristics and limitations is crucial for choosing the right streaming solution.

Know the batch ingestion services and when to use them Fully understand AWS Glue's capabilities for ETL jobs, including crawlers, job bookmarks for incremental loads, and data quality features. Understand AWS DMS for database migration and replication, and when to use AWS DataSync for transferring large volumes of data between on-premises and AWS storage services. Know how to handle schema changes and data transformations during batch ingestion.

Be able to implement large-scale data transfer solutions Understand when to use the AWS Snow Family devices versus AWS Direct Connect for large data transfers. Snow devices are ideal for offline transfer of massive datasets with limited connectivity, while Direct Connect provides dedicated network connections for consistent, secure data transfer. Know the capacity limits and security features of each solution.

Be able to design fault-tolerant data ingestion architectures Know how to implement replication, partitioning, and error handling in streaming and batch ingestion pipelines. Understand how to use enhanced fan-out in Kinesis, configure MSK for high availability, and

implement proper retry mechanisms in AWS Glue jobs. Master the configuration of monitoring and alerting for ingestion pipelines.

Know how to optimize performance and costs in data ingestion Understand the throughput limits, scaling options, and cost implications of different ingestion services. Know how to choose between on-demand and provisioned capacity modes, implement proper partitioning strategies, and optimize buffer sizes and intervals for optimal performance and cost efficiency.

Review Questions

1. A data engineering team is designing a real-time data processing system for a social media analytics company. The system needs to ingest and process millions of social media posts per minute, with the ability to scale automatically based on incoming data volume. The processed data should be available for real-time dashboards and stored for future analysis. Which combination of Amazon Kinesis services would best meet these requirements? (Choose two.)
   - **A.** Amazon Kinesis Data Streams for data ingestion
   - **B.** Amazon Data Firehose for data delivery to storage
   - **C.** Amazon Kinesis Video Streams for data ingestion
   - **D.** Amazon Kinesis Data Analytics for real-time processing
   - **E.** Amazon SQS for message queuing

2. A data engineering team is building a system to analyze IoT sensor data from thousands of devices in real time. The data needs to be processed as it arrives and then stored for long-term analysis. The team wants to minimize the operational overhead and ensure that no data is lost during processing. Which Amazon Kinesis configuration would best meet these requirements?
   - **A.** Use Kinesis Data Streams with enhanced fan-out consumers and configure the stream to use Provisioned Throughput mode.
   - **B.** Implement Amazon Data Firehose with dynamic partitioning and deliver data directly to Amazon S3.
   - **C.** Set up Kinesis Data Streams with standard consumers and use On-Demand capacity mode.
   - **D.** Deploy Kinesis Data Analytics with tumbling windows and connect it directly to IoT Core.

3. A data engineering team is designing a real-time data streaming solution using Amazon Managed Streaming for Apache Kafka (Amazon MSK). They need to ensure that their Kafka cluster can handle sudden increases in traffic without manual intervention. Which feature should they enable to meet this requirement?
   - **A.** MSK Connect
   - **B.** MSK Serverless
   - **C.** MSK Auto Scaling
   - **D.** MSK Cruise Control

4. A company is using Amazon MSK for its data streaming needs and wants to integrate it with its existing AWS Identity and Access Management (IAM) setup for authentication and authorization. Which of the following methods should they use to achieve this integration?
   - **A.** Enable SASL/SCRAM authentication and create IAM users for each Kafka client.
   - **B.** Use SSL certificates for authentication and IAM roles for authorization.
   - **C.** Enable IAM access control for the MSK cluster and use IAM roles or users for client authentication.
   - **D.** Implement Kerberos authentication and map Kerberos principals to IAM roles.

5. A company is using Amazon Data Firehose to deliver streaming data to Amazon S3. They want to ensure that the data is encrypted at rest in the S3 bucket. Which of the following methods provides the highest level of control over the encryption keys while still allowing Amazon Data Firehose to encrypt the data?
   - **A.** Enable encryption in transit for Amazon Data Firehose.
   - **B.** Use AWS Key Management Service (KMS) customer-managed keys.
   - **C.** Use client-side encryption before sending data to Amazon Data Firehose.
   - **D.** Use S3-managed encryption keys (SSE-S3).

6. A data engineering team needs to build an ETL pipeline to process large volumes of semi-structured JSON data stored in Amazon S3 and load it into an Amazon Redshift data warehouse. The team wants to minimize the amount of custom code they need to write and maintain. Which AWS service should they choose for this task?
   - **A.** Amazon EMR
   - **B.** AWS Glue
   - **C.** AWS Lambda
   - **D.** Amazon Redshift Spectrum

7. A company wants to build a data lake solution that can automatically discover and catalog data from various sources, including on-premises databases and cloud storage. It needs a solution that can handle schema changes automatically and provide a searchable catalog of its data assets. Which AWS service should they use as the core component of this solution?
   - **A.** Amazon Athena
   - **B.** AWS Data Pipeline
   - **C.** AWS Glue
   - **D.** AWS Lake Formation

8. A large e-commerce company is planning to migrate its on-premises Oracle database to Amazon Aurora PostgreSQL. The database is approximately 5 TB in size and experiences high transaction volumes during business hours. The migration needs to be performed with minimal downtime, and the company wants to ensure that any data changes occurring during the migration process are captured and applied to the target database. Additionally, the company needs to transform some of the data during the migration process, including encrypting sensitive customer information and restructuring some tables to optimize for the new database engine. The migration also needs to handle large BLOB data stored in the Oracle database. Which AWS service or combination of services would be most suitable for this migration task?
   - **A.** Use AWS Glue to extract data from Oracle, transform it, and load it into Aurora PostgreSQL.
   - **B.** Employ AWS DataSync to transfer the data and use AWS Lambda for data transformation.
   - **C.** Utilize Amazon EMR with custom scripts for data extraction, transformation, and loading.
   - **D.** Implement AWS Database Migration Service (DMS) with custom transformations.

9. A genomics research institute has accumulated 80 TB of raw sequencing data stored in their on-premises data center. They want to transfer this data to AWS for analysis using various AWS analytics services. The institute is located in a remote area with limited Internet connectivity, with a maximum bandwidth of 100 Mbps. They need to complete the data transfer within two weeks and ensure the security of the sensitive genetic data during transit. Additionally, they want to minimize the impact on their daily operations and internet usage. Which AWS service or solution should they use for this data transfer?

   A. Set up an AWS Direct Connect connection and transfer the data over a private network.

   B. Use AWS DataSync with a DataSync agent installed on-premises to transfer the data to Amazon S3.

   C. Implement AWS Snowball devices for offline data transfer to AWS.

   D. Utilize Amazon S3 Transfer Acceleration with multipart uploads for faster data transfer.

10. A retail company wants to ingest large volumes of daily sales data (approximately 500 GB per day) from its operational database into its Amazon Redshift data warehouse for analytics. The data is currently exported as CSV files and stored in Amazon S3. The company needs a solution that can quickly load this data into Redshift tables, perform basic transformations like data type conversions, and handle occasional schema changes. They also want to ensure optimal performance and cost-efficiency. Which method should it use for this data ingestion process?

    A. Use AWS Glue to extract data from S3, transform it, and load it into Redshift.

    B. Implement DistCp with Amazon EMR to copy data from S3 to Redshift.

    C. Utilize Sqoop with Amazon EMR to transfer data from S3 to Redshift.

    D. Employ the Amazon Redshift `COPY` command to load data directly from S3.