**10**

**Big Data and Streaming Data Processing in AWS**

Traditionally, a business’s most important resources are its human and financial capital. However, in the last few decades, more and more businesses have realized that another resource may be just as, if not more, vital: its data capital.

Data has taken a special place at the center of some of today’s most successful enterprises. For this reason, business leaders have concluded that to survive in today’s business climate, they must collect, process, transform, distill, and safeguard their data like their other traditional business capital.

In this chapter, you will dive deep into AWS’s analytics services. First, you will learn about Amazon EMR, which is Hadoop in the cloud, and about AWS data cataloging offering, AWS Glue. Finally, you will look at how to handle streaming data using AWS. In this chapter, you will cover the following topics:

* Why use the cloud for big data analytics?
* Amazon **Elastic Map Reduce** (**EMR**)
* Introduction to AWS Glue
* Choosing between AWS Glue and Amazon EMR
* Handling streaming data in AWS
* Choosing between Amazon Kinesis and Amazon MSK

By the end of this chapter, you will know about the various AWS services available to build an analytics pipeline and perform **Extract, Transform, and Load** (**ETL**) operations on your significant data workload. Let’s roll up our sleeves and get to it.

**Why use the cloud for big data analytics?**

The definition of big data has changed drastically in the last 2 decades. Now, there is a massive amount of data coming from various sources. This data can be structured, unstructured, or semi-structured. We see large technology organizations, such as Amazon, Google, Meta, and so on, flourishing as they can get insight from user data and utilize it for customer benefits, thus growing their business multifold. IDC says, *“The Global DataSphere is expected to more than double in size from 2022 to 2026,”* (Source – <https://www.idc.com/getdoc.jsp?containerId=US49018922>).

Now, it’s normal for organizations to have multi-terabytes or petabytes of data, and you want to gain new insights to use the power of this collected data. You must easily access and analyze all data types, such as log files, clickstream data, voice, and video. But your team may require diverse skills and tools. You need to enable your team and applications to access the data but ensure security, privacy, and compliance regulations are met for the data. Also, it’s important to clearly decide, watch, and handle who can see certain information.

It would become time-consuming, complex, and expensive if you were to collect, store, secure, and process all that data on-premises. If you were to pick up an open-source project such as a Spark, Hive, or Pig and set up the Hadoop environments, it would require a lot of expertise to ensure the open-source projects integrated well and could work with one another. Hadoop is a free software created by Apache for storing and working with big datasets. With Hadoop, you can link many computers together to simultaneously analyze huge datasets faster, rather than relying on a single large computer to store and analyze the data. You can learn more about Hadoop by visiting the AWS page – <https://aws.amazon.com/emr/details/hadoop/what-is-hadoop/>.

Big data administration skills are hard to find. Even if you have taken care of that, the infrastructure likely does not scale as needed. Innovation and development are always limited by how many resources you have. Costs add up very quickly. The tight coupling of storage and compute pushes the costs further. You are forced to buy more computing even if you just need more storage, as you will not process all the data in one go.

To address all these concerns, AWS built **Amazon Elastic Map Reduce** (**EMR**), a managed big data platform supporting the open-source project part of the Hadoop framework.

EMR is a kind of Hadoop solution for the cloud, meaning you don’t need to reskill your workforce to move on to the cloud. EMR scales based on processing needs while keeping costs low by requiring you to pay only for what you use and keeping compute and storage separate. Let’s learn more about EMR in detail.

**Amazon Elastic Map Reduce (EMR)**

Back in 2009, AWS introduced EMR, a tool that can handle extremely large amounts of data (terabytes and petabytes) using the latest open-source big data tools like Spark, Hive, Presto, HBase, Flink, and Hudi in the cloud. Amazon EMR is a managed cluster platform that makes it easier to run big data tools, such as Apache Hadoop and Apache Spark, on the AWS cloud for processing and analyzing massive datasets. It is a wrapper around distributed open-source computing frameworks. This wrapper abstracts the effort required to set up infrastructure, security, network communication, disaster recovery, and scalability. Additionally, EMR offers 100% compliance with open-source APIs. So, there is no need to change your application code when you move to EMR from the on-premises Hadoop system.

EMR runs directly against the data stored in your S3 data lake, so you don’t need to move that data or transform your data. You can store data in the data lake in its raw and processed forms, as well in a variety of formats, including log files, images, and so on. S3 data lakes are scalable, secure, and cost-effective, making them a popular choice. You will learn more about AWS data lakes in *Chapter 15*, *Data Lake Patterns – Integrating Your Data across the Enterprise*.

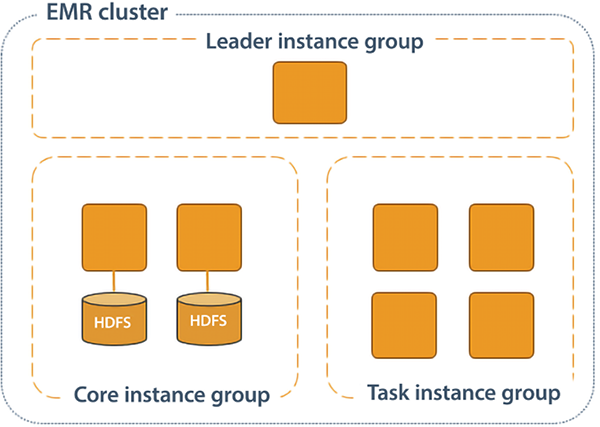
EMR makes it simple to produce clusters and set up one, hundreds, or thousands of computing units to manage data of any magnitude. EMR also automatically scales cluster sizes based on usage, and you only pay for the resources you consume.

Because you’re running against S3, you can have multiple clusters operating on the same data. Using Amazon S3 for storage also provides strong business continuity. Rather than depending on one cluster that will go down in the case of a DC failure, EMR clusters in multiple **Availability Zones** (**AZs**) have equal access to Amazon S3. In the event of a failure, you can quickly switch traffic to the other cluster or spin up a new cluster in another AZ.

EMR decouples compute and storage to optimize costs, allowing you to scale each independently. For storage, you can take advantage of the tiered storage of Amazon S3, and for computing, you can take advantage of **Elastic Compute Cloud** (**EC2**) Spot Instances to save up to 80% off the cost of using on-demand instances. EMR also enables data analysts’ and scientists’ interactive analytics by integrating with other AWS machine learning services. Let’s dive deeper to learn more about EMR clusters.

**Understanding EMR clusters and nodes**

Amazon EMR is a service that is centered around clusters, which are groups of Amazon EC2 instances. These instances, known as nodes, each have a specific function within the cluster and are equipped with different software tools depending on their role. In an Amazon EMR cluster, each node serves a unique purpose; as you can see in the following diagrams, the node groups are Leader nodes, Core nodes, and Task nodes.



*Figure 10.1: Amazon EMR node types*

As shown in the preceding diagram, the following is the role of each node type:

* **Leader node:** The master node is the central point of control for the cluster. It manages the job flow and coordinates the work across the other nodes in the cluster.
* **Core nodes:** Core nodes are worker nodes that store data and process tasks. They run tasks as directed by the master node and store intermediate data in memory or on a local disk.
* **Task nodes:** Task nodes are similar to Core nodes, but they are only used to run tasks and do not store data. They are typically used when a large amount of compute resources are needed for a short period of time.

In addition to these, you have additional nodes to support EMR jobs – for example, Gateway nodes to provide a connection to external data sources and to stage data in and out of the cluster. Client nodes are used to submit jobs to the cluster and view the status of those jobs. They do not store data or run tasks, but they do have access to the data and resources of the cluster.

An Amazon EMR cluster can be configured with three Leader nodes to provide high availability. In the event that the primary Leader node fails, Amazon EMR will automatically switch to a standby Leader node to ensure that the cluster remains operational. If a Leader node does fail, Amazon EMR will also automatically replace it with a new Leader node that is configured in the same way and has the same bootstrap actions as the failed node. This helps to ensure that the cluster remains available and can continue to process data even in the event of a failure.

EMR provides a wide variety of EC2 instances to choose from. It allows you to choose the instance families suitable for your workload. The processing ability of your Core nodes and the size of your data determine how much information you can manage. While processing, the input, intermediate, and output datasets are stored on the cluster.

The cluster type could be **a persistent cluster**, where you always want to keep it on to run interactive queries, or a transient cluster, which you need for a few hours to process batch jobs for data processing. A **transient cluster** can have the same lifetime as the workload running on it. Once the application or workload completes, the results are stored inside S3, and the cluster can be terminated. This provides the benefit of cost saving while running EMR.

Amazon EMR offers a feature called **EMR Managed Scaling**, which allows you to automatically adjust the size of your cluster based on your workload. This can help you to optimize the cost and speed of your cluster by scaling it up or down as needed to meet the demands of your workload. EMR Managed Scaling continuously monitors cluster metrics to make decisions about scaling and ensures that your cluster is always sized appropriately to meet your needs. AWS manages all the configuration, with no policies for you to define except for the minimum and maximum number of instances.

AWS also provides the ability to deploy **Amazon EMR on EKS**. This is a new deployment mode that allows you to run EMR on EKS-managed Kubernetes clusters. EMR on EKS brings the power of both EMR and EKS services into a consolidated offering so you can run Spark applications on Kubernetes easily and securely. With EMR on EKS, there are no more clusters to build specifically for your analytics workload. AWS containerizes the Spark runtime, so applications can quickly run on an existing EKS cluster. You’ll also be able to consolidate resources into a single cluster and improve their overall utilization, which drives cost savings. This allows you to run multiple versions of Spark on the same cluster and also allows you to build a security model that is job centric.

AWS also launched **Amazon EMR Serverless** at re:Invent 2021. EMR Serverless automatically determines and provisions the compute and memory resources to run the application and scales the resources up and down as needed based on changing requirements.

For example, Amazon EMR Serverless automatically provisions and adjusts resources required to run Spark applications as the data volumes being processed change. You can check the status of running jobs in **EMR Studio** or the AWS Console, review job history, and use familiar open-source tools to debug jobs. EMR Studio is a fully managed, web-based notebook environment that you can use to interactively explore, visualize, and analyze data using Apache Spark and other popular open-source libraries. It is integrated with Amazon EMR, so you can easily run and debug your code on a live cluster, and it includes collaboration features such as version control and the ability to share notebooks with other users.

In this section, you’ve learned about compute in EMR; let’s look into storage in EMR with the supported file system.

**Understanding the EMR File System (EMFRS)**

To run a large data workload, you need scalable storage and a file system to support that storage. One major differentiation for EMR is its support for S3, for which AWS built a propriety file system called EMRFS, which continues to support other traditional file systems. Let’s look into file systems supported by EMR:

* **Hadoop Distributed File System (HDFS)**: HDFS is a type of file system that is meant to operate on low-cost, commodity hardware in a distributed computing environment. EMR mainly utilizes HDFS as its primary storage system, and it is well suited for storing large amounts of data that need to be processed by MapReduce jobs.
* **Elastic Map Reduce File System (EMRFS)** : EMRFS is a Hadoop-compliant file system that is designed to work seamlessly with Amazon S3. It allows you to store data in S3 while still being able to access it through the HDFS interface. EMRFS provides consistent, low-latency data access while still maintaining the durability and cost-effectiveness of S3.
* **The local file system**: EMR can also use the local file system of the instances in the cluster as the file system. This can be useful for storing intermediate data that is generated by MapReduce jobs, or for storing small amounts of data that are used by the jobs.

Using EMRFS and utilizing the power of S3 until you need sub-millisecond latency helps you reduce cost by decoupling compute and storage.

You don’t want to limit your data pipeline to just seeing historical data. The actual value for data comes when you can predict the future using machine learning and play with your data using a developed, friendly interface. AWS has launched an EMR Studio offering to make your data analysis future-looking. Let’s look into more details.

**Amazon EMR Studio**

EMR Studio is an Integrated Development Environment that reduces the time that data scientists and engineers take to build and deploy code. The following are the key benefits of EMR Studio:

* **Use corporate identity to log into notebooks instead of the AWS Console**: You can log into Amazon EMR Studio through a secure URL to manage notebooks using corporate identities via AWS **Single Sign-On** (**SSO**). There is no need to log into the AWS Console. It is a multi-tenant studio where multiple users and groups can log into the same studio.
* **Develop, visualize, debug and optimize analytics and machine learning applications**: EMR Studio provides fully managed notebooks based on JupyterLab. It gives you the flexibility to create notebooks independent of clusters. You can attach and detach notebooks to and from clusters using a single click. You can also provision EMR clusters using cluster templates pre-configured by your administrator or create a new cluster from scratch. You can diagnose jobs on active and terminated clusters using the Spark UI, Tez UI, and **Yet Another Resource Negotiator** (**YARN**) Timeline Service. EMR Studio allows you to do this easily by browsing through all your clusters in one place and narrowing down clusters or jobs for investigation using filters.
* **Collaborate with others by sharing notebooks via GitHub:** You can collaborate with your team and analyze data from your data lakes using PySpark, Spark SQL, Spark R, and Scala. The studio integrates with AWS CodeCommit, GitHub, and Bitbucket for easy collaboration, and you can import custom Python libraries like Pandas or NumPy and install custom kernels directly from notebooks.
* **Build pipelines using orchestration services like Apache Airflow**: You can run notebooks as pipelines via Amazon’s**Managed Workflows for Apache Airflow** (**MWAA**), self-managed Apache Airflow, or AWS Step functions. You can also parameterize and chain notebooks that can be run as pipelines.

Data security is an essential aspect of any analytics workload. Let’s look into data security in EMR.

**Securing data in Amazon EMR**

Amazon EMR allows you to specify security configurations to ensure the encryption of data at rest, in transit, or both. You can use these configurations to encrypt data stored in Amazon S3 or on the local disks of your cluster instances. The security configurations are stored separately from the cluster configuration, so they can easily be reused whenever you create a new cluster. In addition, in-transit encryption can be enabled to secure data as it is transmitted between various components of the cluster.

There are several ways to secure data in Amazon EMR:

* **Encrypting data at rest:** Data at rest refers to data that is stored on disk, such as data stored in Amazon S3 or on the local disks of your EMR cluster instances. To safeguard data at rest in EMR, you can create a security configuration that establishes the necessary parameters for encrypting data stored in Amazon S3 or on the local disks of your cluster instances. This helps ensure that your data is secure and unreadable by unauthorized users.
* **Encrypting data in transit:**Data in transit refers to data that is transmitted between components of your EMR cluster, such as data transmitted between Amazon EC2 instances or between Amazon EC2 instances and Amazon S3. To encrypt data in transit in EMR, you can use **Secure Sockets Layer** (**SSL**)/TLS to secure data transmitted over the network.
* **Using secure access to Amazon S3:** When accessing data stored in Amazon S3 from your EMR cluster, you can use SSL to encrypt the data transmitted over the network. You can also use AWS **Identity and Access Management** (**IAM**) to control access to your data in Amazon S3 and to make sure that only authorized users and applications have access to your data.
* **Using security groups:** Using security groups, you can manage the flow of incoming and outgoing traffic to and from your EMR cluster instances. This helps to limit access to only authorized users and resources, and enhances the overall security of your cluster. This also allows you to restrict access to your cluster instances and to specify which IP addresses and protocols are allowed to access your cluster.
* **Using network isolation:** You can use Amazon **Virtual Private Cloud** (**VPC**) to create a virtual network that is isolated from the rest of the internet and to launch your EMR cluster in this virtual network. This allows you to further secure your cluster by creating a private network that is isolated from the public internet.

For authentication, use can use IAM and Kerberos. You can also use AWS SSO through the corporate active directory to verify the user’s validity. Furthermore, EMR provides the ability to perform audits through logs and AWS CloudTrail.

Here is an example of using EMR to process data stored in Amazon S3 using a MapReduce job written in Python:

1. First, you will need to create an Amazon S3 bucket to store your input data and output data.
2. Next, you will need to upload your input data to Amazon S3. This can be done using the AWS Management Console, the AWS CLI, or the Amazon S3 API.
3. Then, you will need to create an EMR cluster using the AWS Management Console, the AWS CLI, or the Amazon EMR API. When creating the cluster, you will need to specify the number of instances you want in the cluster, the instance type, and the EC2 key pair that you want to use to access the cluster instances.
4. Once the cluster is up and running, you can submit a MapReduce job to the cluster using the AWS Management Console, the AWS CLI, or the Amazon EMR API. The MapReduce job should specify the location of the input data in Amazon S3 and the location where you want the output data to be stored.
5. The MapReduce job will then be executed on the EMR cluster, and the output data will be stored in the specified location in Amazon S3.

Here is an example of the AWS CLI command to submit a MapReduce job to an EMR cluster:

aws emr add-steps --cluster-id j-123456789EXAMPLE --steps Type=CUSTOM\_JAR,Name=SABookCustomJar,ActionOnFailure=CONTINUE,Jar=s3://sa-book-bucket/book-jar.jar,Args=["s3://sa-book-bucket/input-data","s3://sa-book-bucket/output-data"]

This command will submit a MapReduce job that runs the book-jar.jar JAR file on the input data located in s3://sa-book-bucket/input-data and stores the output data in s3://sa-book-bucket/output-data.

EMR is a vast topic that warrants a book in itself. You can learn more about EMR by visiting the AWS page here: <https://aws.amazon.com/emr/>.

Spark is one of the most popular big data processing frameworks due to its high performance and speed. Recently, most organizations have moved to using Spark. Another key attribute of big data analytics is data cataloging. Data catalogs help you to understand the details of your data, such as its type, volume, and structure, which help to build efficient data processing jobs. AWS provides the AWS Glue service to handle Spark workloads and data cataloging. Let’s learn about it in more detail.

**Introduction to AWS Glue**

Data-driven businesses can increase their profitability and efficiency, reduce costs, deliver new products and services, better serve their customers, comply with regulatory requirements, and ultimately thrive. Unfortunately, as we have seen many examples of in recent years, companies that don’t make this transition will not be able to survive. An important part of a data-driven enterprise is the ability to ingest, process, transform, and analyze this data.

AWS Glue is a foundational service at the heart of the AWS offering.

With the introduction of Apache Spark, enterprises can process petabytes’ worth of data daily. Processing this amount of data opens the door to making data an enterprise’s most valuable asset. Processing this data at this scale allows enterprises to create new industries and markets. Some examples of business activities that have significantly benefited from this massive data processing are as follows:

* Personalized marketing
* Drug discovery
* Anomaly detection (such as fraud detection)
* Real-time log and clickstream processing

AWS has created a service that leverages Apache Spark and takes it to the next level. The name of that service is AWS Glue.

What is AWS Glue? AWS Glue is a fully managed service used to extract data from data sources, ingest the data into other AWS services, such as Amazon S3, and transform this data to be used by consuming services or users. It is not meant to be used for small batches and files. Under the hood, AWS Glue uses Apache Spark, running it in a serverless environment.

Another important feature of AWS Glue is that it can handle disparate sources such as SQL and NoSQL databases – not just Amazon S3 files.

As we mentioned in the introduction, it is hard to overestimate the value of data in the current environment. Regardless of the industry, properly harnessing and leveraging data is critical to compete. AWS Glue and its underlying Apache Spark engine function as cornerstone technologies in many enterprises to process the vast amounts of data generated.

As with quite a few other AWS services, AWS Glue leverages a popular open-source technology (in this case, Apache Spark) and places wrappers around it to supercharge the technology even further.

Let’s look at a simple but powerful example of this. The traditional way to set up Apache Spark is to set up a cluster of powerful machines. Depending on how much data will be ingested and how many transformations need to be performed, it is not uncommon to have Spark clusters with dozens and sometimes even hundreds of nodes. As you can imagine, a dedicated infrastructure setup like this can be costly.

Using AWS Glue, you can create a similarly powerful cluster of machines that are spun up when demand requires it, but importantly, the cluster can be spun down when demand wanes.

If no work is processed, the cluster can be completely shut down, and the compute costs go down to zero. By making costs variable and not having to pay for idle machines, the amount of use cases that can be handled by AWS Glue versus a traditional Apache Spark cluster increases exponentially. Projects that would have been prohibitively expensive before now become economically feasible.

Here is a list of common use cases that leverage AWS Glue:

* The population of data lakes, data warehouses, and lake houses
* Event-driven ETL pipelines
* The creation and cleansing of datasets for machine learning

AWS Glue has a series of components to achieve its intended purpose as an **Extract, Transform, and Load** (**ETL**) service. These are as follows:

* The AWS Glue console
* The AWS Glue Data Catalog
* AWS Glue classifiers
* AWS Glue crawlers
* AWS Glue code generators

Let’s check out the major ones.

**Operating the AWS Glue console**

The AWS Glue console creates, configures, orchestrates, and develops ingestion workflows. The AWS Glue console interacts with other components in AWS Glue by calling APIs to update the AWS Glue Data Catalog and to run AWS Glue jobs. These jobs can be run to accomplish the following kind of actions:

* **Definition of AWS Glue objects such as connections, jobs, crawlers, and tables** – The console can be used to create various AWS Glue objects. We will learn more about crawlers in an upcoming section. A table in AWS Glue is simply a file after it is processed. Once processed, the file can act as a SQL table, and SQL commands can be run against it. As with any kind of SQL database, we need to create a connection to connect to the table. All these objects can be created in the AWS Glue console.
* **Crawler scheduling** – AWS Glue crawlers, which we will learn about shortly, must be scheduled. This scheduling is another action that can be performed in the AWS Console.
* **Scheduling of job triggers** – The scheduling of job triggers that will perform ETL code can also be implemented in AWS Glue.
* **Filtering of AWS Glue objects** – In the simplest AWS Glue implementations, it may be easy to locate different objects by simply browsing through the objects listed. Once things start getting a little complicated, we will need to have the ability to filter these objects by name, date created, and so on. This filtering can be performed in the AWS Glue console.
* **Transformation script editing** – Lastly, one more task that can be accomplished in the AWS Glue console is the creation and maintenance of ETL scripts.

**Cataloging with the AWS Glue Data Catalog**

The AWS Glue Data Catalog is a persistent metadata repository. It is another service managed by AWS that enables the storage, annotation, and publishing of metadata. Under the hood, the AWS Glue Data Catalog operates similarly to an Apache Hive metastore.

There is only one AWS Glue Data Catalog per AWS Region. Different services can persist and access the extracted metadata by having only one repository per Region. It is essential to clearly distinguish between the metadata and the data in each dataset. Often, the data is sensitive and cannot be shared unless the authority to do so is granted directly. But in many cases, sharing the metadata for a given dataset is okay. For example, if a file contains social security numbers, names, addresses, and phone numbers, only a select group of individuals may need access to the actual data. But it is okay to disseminate to a wider audience that this file contains social security numbers, names, addresses, and more.

AWS IAM policies can be created to give access permission to AWS Glue Data Catalog datasets. These policies can be used to manage access for a variety of groups. Certain groups may be allowed to publish data, others may be allowed to access the metadata, and others may be given access to the actual data. Using IAM policies, it is possible to give detailed and granular access to the appropriate parties at the required levels.

Another feature of the AWS Glue Data Catalog is its schema version history. A **schema** is the structure and format of a data record, and it is used to define the fields, types, and constraints of the data. Version history will be kept about each ingestion and each Data Catalog update so that these changes can be monitored and audited over time.

The Data Catalog also offers audit and governance functionality. It can track when schema changes are performed and when data is accessed. It allows auditors to determine when and by whom changes are performed.

The catalog can be changed using **Data Definition Language** (**DDL**) statements or the AWS Management Console. Any schemas that are created are automatically persisted until they are explicitly removed. These files can be accessed using Amazon Athena. Amazon Athena leverages a schema-on-read methodology that enables table definitions to be applied to files in Amazon S3 during query execution instead of during file ingestion. This minimizes additional file writes and data transformation. Additionally, these table definitions created by AWS Glue can be deleted, and the underlying files will persist in Amazon S3.

**Crawling with AWS Glue crawlers**

AWS Glue crawlers are used to discover data stored in a data store and create a table schema for the data in the AWS Glue Data Catalog. The data store can be a database, a flat file, or a collection of files stored in a directory. AWS Glue crawlers scan files, extract the file metadata, and populate the AWS Glue Data Catalog with this metadata information. An AWS Glue crawler can scan multiple file locations simultaneously. Once these files have been scanned and the tables are available, they can then be used in other jobs, and the data contained can be transformed and ingested into other downstream processes. Once the files have been crawled, users will be able to access the contents of the files while treating these files as SQL tables. The table definitions can be used to read and write data stored in the data store.

AWS Glue crawlers can be configured to access data stored in a variety of data stores, including Amazon S3, Amazon RDS, Amazon Redshift, and JDBC data stores. You can also create custom connectors to access data in other data stores. To use an AWS Glue crawler, you create a crawler definition that specifies the data store to be crawled and the classifiers to be used to identify the data. You can then schedule the crawler to run on a regular basis, or you can run it on demand.

When the crawler runs, it connects to the data store and extracts metadata about the data. It then creates table definitions for the data in the AWS Glue Data Catalog and stores the metadata in the Data Catalog. AWS Glue crawlers are a useful tool for discovering and cataloging data stored in a variety of data stores. They can save you time and effort by automating the process of creating table definitions and by providing a central location for storing and accessing metadata about your data.

AWS Glue crawlers are extremely scalable and can crawl multiple data sources simultaneously. Once the AWS Glue Data Catalog is populated, it can be fed into other processes to perform ETL tasks. In addition to discovering the schema information for new data sources, it can also discover changes in the schema that have occurred in previously ingested data sources. It will update the Data Catalog with the new metadata.

IMPORTANT NOTE

One important limitation is that while AWS Glue crawlers can discover schema information and extract metadata from individual files and tables, it does not discover relationships between the data sources.

There is one important consideration when it comes to security with this architecture. Regardless of which AWS or third-party services are used to access tables from the AWS Glue Data Catalog, any restrictions placed on the underlying S3 files will persist. If a user tries to access a resource using SQL commands from one of the tables in the AWS Data Catalog and they don’t have access to the underlying table, they will not be able to view the contents of that table.

**Categorizing with AWS Glue classifiers**

As data is crawled from the various data sources, its metadata will be extracted, and the Data Catalog will be populated. Additionally, it can also be classified by AWS Glue classifiers. Classifiers recognize the format of the data and persist this information. Classifiers also assign a certainty score depending on how sure they are about the data format.

AWS Glue comes with a set of out-of-the-box classifiers and also provides the ability to create custom classifiers. AWS Glue classifiers are used to identify the schema of data stored in a data store. Classifiers are used by AWS Glue crawlers to determine the data format, data type, and other characteristics of the data. For example, suppose you have a collection of CSV files stored in an Amazon S3 bucket, and you want to use an AWS Glue crawler to discover the data and create table definitions for the data in the AWS Glue Data Catalog. You can use a CSV classifier to specify the format of the data and identify the delimiter used in the files.

AWS Glue will first run the custom classifiers during the classifier invocation, using the sequence specified in the crawler configuration. If the custom classifier cannot determine the format of a given data source, AWS Glue will then run the built-in classifiers to see if they can determine the format. You can then use this classifier when creating a crawler to crawl the CSV files in the Amazon S3 bucket. The crawler will use the classifier to determine the schema of the data and create table definitions in the AWS Glue Data Catalog.

Lastly, it is important to note that, in addition to providing a format, a classifier will return a certainty score with a value between 0.0 and 1.0. A score of 1.0 indicates that the classifier has 100% certainty about the data format.

As you can imagine, some of the built-in classifiers can recognize some of the most common file formats, including the following:

* Apache Avro
* Apache ORC
* Apache Parquet
* JSON
* Binary JSON
* XML
* **Comma-Separated Values** (**CSV**)
* **Tab-Separated Values** (**TSV**)

You can use these classifiers to identify the schema of the data and to create table definitions in the Data Catalog.

**Generating code with AWS Glue code generators**

AWS Glue automatically generates highly scalable Python or Scala ETL code to ingest and transform data. The resulting code is optimized for distributed computing. The choice of language between Python and Scala is up to you, based on your coding preferences. Once the code is generated, it can be customized and edited using AWS Glue Studio. AWS Glue Studio is an editor that allows developers to create some functionality using a drag-and-drop job editor. However, for proficient developers, editing the code directly is also possible.

If you want to customize the generated code, AWS Glue has development endpoints that enable developers to make changes to the code. It can then be tested and debugged. AWS Glue provides the ability to create custom readers, writers, and transformations. Code created with these endpoints can be incorporated into code pipelines, then versioned and managed in code repositories like any other code.

Here is a sample ETL script in Python that demonstrates how to use AWS Glue to transform data stored in S3 and write the transformed data back to S3:

import sys

from awsglue.transforms import \*

from awsglue.utils import getResolvedOptions

from pyspark.context import SparkContext

from awsglue.context import GlueContext

from awsglue.job import Job

*## @params: [JOB\_NAME]*

args = getResolvedOptions(sys.argv, ['JOB\_NAME'])

sc = SparkContext()

glueContext = GlueContext(sc)

spark = glueContext.spark\_session

job = Job(glueContext)

job.init(args['JOB\_NAME'], args)

*## YOUR CODE HERE*

datasource0 = glueContext.create\_dynamic\_frame.from\_catalog(database = "mydatabase", table\_name = "mytable", transformation\_ctx = "datasource0")

applymapping1 = ApplyMapping.apply(frame = datasource0, mappings = [("col1", "string", "col1", "string"), ("col2", "string", "col2", "string"), ("col3", "string", "col3", "string")], transformation\_ctx = "applymapping1")

resolvechoice2 = ResolveChoice.apply(frame = applymapping1, choice = "make\_struct", transformation\_ctx = "resolvechoice2")

dropnullfields3 = DropNullFields.apply(frame = resolvechoice2, transformation\_ctx = "dropnullfields3")

datasink4 = glueContext.write\_dynamic\_frame.from\_options(frame = dropnullfields3, connection\_type = "s3", connection\_options = {"path": "s3://mybucket/output"}, format = "parquet", transformation\_ctx = "datasink4")

job.commit()

This ETL script performs the following operations:

* Reads data from a table in the Glue Data Catalog called mytable in a database called mydatabase
* Applies a mapping to the input data, specifying the data types for each column
* Resolves any data type conflicts in the input data
* Drops any null fields from the input data
* Writes the transformed data to an S3 bucket in the Parquet format

Once the code is developed, it can be triggered as an AWS Glue job. These jobs can be triggered in a variety of ways:

* **On a schedule** – For example, every day of the week at 9 a.m. or once a week on Tuesday at 3 p.m.
* **By manual intervention** – When the user kicks off the job via either the Console or the AWS CLI
* **Event triggers** – Based on a triggering event such as a file loading or a row in a database being inserted

Often, these jobs will have dependencies on each other. AWS Glue provides orchestration capabilities to cobble together the ETL dependencies and handles retry logic in case a job fails. The execution of all these jobs is tracked by Amazon CloudWatch, which will enable users to monitor these jobs. Alerts can be created and received for various actions, including job completion or job failure.

**IMPORTANT NOTE**

Integrating AWS Glue jobs with other orchestration tools such as Apache Airflow and Control-M is possible. You can learn more about **Apache Airflow** at <https://airflow.apache.org/>.

More information about **Control-M** can be found at <https://www.bmc.com/it-solutions/control-m.html>.

Now AWS provides **MWAA**,which is a fully managed service that makes it easy to create, maintain, and monitor workflows using Apache Airflow. MWAA removes the need to set up, operate, and scale infrastructure, allowing you to focus on creating, testing, and running workflows to move and transform data. With MWAA, you can define workflows as **Directed Acyclic Graphs** (**DAGs**) using Python, and then use the MWAA web console or the MWAA API to deploy and manage your workflows. MWAA provides a fully managed execution environment, with built-in support for scheduling, monitoring, and retries. It also integrates with other AWS services, such as Amazon **Simple Queue Service** (**SQS**) and Amazon **Simple Notification Service** (**SNS**), to enable you to build complex data pipelines.

**AWS Glue serverless streaming ETL**

You may be too young to remember, but back in the day, if you went to a store and tried paying with a credit card, they had these massive books where the checkout person had to look up your card number to determine if your card was fraudulent. Not only did this method slow down the checkout process but, as you can imagine, by the time fraudulent cards were added to the book, they could have been used many times to purchase thousands of dollars worth of merchandise.

As technology improved, this process got quicker to the point where these records became automated and were updated on a nightly basis. But fraud has always been a cat-and-mouse game. Criminals got smart and made sure to use stolen cards faster to be still able to profit from their crimes.

The ideal solution is to be able to report and disseminate the theft of the card in real time. This presents some formidable technological challenges, but technologies like AWS Glue support streaming processing.

Streaming ETL jobs can be created in AWS Glue to continuously process files from streaming services such as Amazon Kinesis and Amazon MSK. These jobs can process, clean, and transform data almost instantaneously. This data then becomes available to data analysts and data scientists so that they can run their models and analysis on the data. Other common use cases for AWS Glue streaming ETL jobs are as follows:

* Processing IoT event streams
* Consuming clickstreams and other web activities
* Analyzing application logs

AWS Glue offers streaming ETL jobs that can be integrated into data pipelines to enhance and consolidate data, merge batch and streaming data sources, and execute analytics and machine learning models. This helps to streamline the process of managing and analyzing data, and can lead to more valuable insights and predictions. All this can be done at scale, enabling the processing of hundreds of thousands of transactions per second.

**AWS Glue DataBrew**

AWS Glue DataBrew is a user-friendly visual data preparation tool that simplifies the task of cleaning, profiling, and converting data stored in AWS data stores. It provides an intuitive interface for users to easily perform data transformations and makes data more accessible for analysis, reporting, and machine learning.

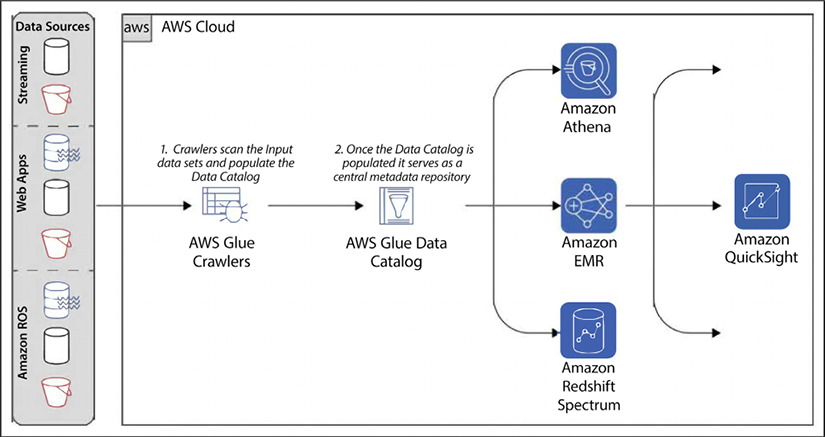
It is fully integrated with Amazon Glue and can be accessed from the Glue console or via the Glue API. With DataBrew, you can create data transformation recipes using a visual interface, without having to write any code. You can use DataBrew to perform a wide range of data preparation tasks, including filtering and sorting data, renaming and dropping columns, pivoting and unpivoting data, and aggregating and summarizing data.

DataBrew also provides data profiling and visualization capabilities, allowing you to understand the structure and content of your data, identify data quality issues, and preview the results of your transformations. Once you have created and tested your data transformation recipes, you can use DataBrew to schedule them to run at a specific frequency or trigger them to run on demand. You can also export your recipes as Glue ETL jobs or Glue Python code, and use them as part of your larger data processing workflows.

So far, in this section, we have learned about the basics of AWS Glue. We also learned about the components that make up AWS Glue and make it a juggernaut. In the next section, we will put it together and explain how the various components work together to provide a powerful combination and deliver one of the most popular services in the AWS ecosystem.

**Putting AWS Glue components together**

Now that we have learned about all the major components in AWS Glue, let’s look at how all the pieces fit together. The following diagram illustrates this:



*Figure 10.2: AWS Glue typical workflow steps*

In the preceding diagram, we see can the various steps that can take place when AWS Glue runs. The steps are explained in the following points:

1. The first step is for the crawlers to scan sources and extract metadata from them.
2. This metadata can then be used to seed the AWS Glue Data Catalog.
3. This metadata can be used by other AWS services, such as Amazon Athena, an AWS-provided query service, Redshift Spectrum, an AWS-provided cloud data warehouse service, and Amazon EMR. These services can be used to write queries against the ingested data using the metadata from the AWS Glue Data Catalog to build these queries.
4. Finally, the results of these queries can be used for visualizations in other AWS services, including Amazon QuickSight (an AWS-provided business intelligence service).

You will learn about Amazon Redshift, Athena, and QuickSight in *Chapter 11*, *Data Warehouses, Data Queries, and Visualization in AWS*.

A wide variety of data sources can be ingested with AWS Glue, such as Amazon S3 objects, Amazon RDS records, or web application data via APIs.

Hopefully, the discussion in these sections has given you a good taste of the basics of AWS Glue and its importance in the AWS ecosystem. Hopefully, you are also convinced that data and the insights derived from processing and distilling it are critical for today’s enterprises.

We will now spend some time learning the best ways to implement AWS Glue in your environment.

**AWS Glue best practices**

As we have done with many of the other services covered in the book, we will now provide some recommendations on how to best architect the configuration of your AWS Glue jobs.

Amazon Athena, under the hood, uses the open-source software Presto to process **Data Manipulation Language** (**DML**) statements and Apache Hive to process DDL statements. An example of a DML statement is a select statement, and an example of a DDL statement is a create table statement.

Similarly, under the hood, AWS Glue runs its ETL jobs using Apache Spark.

Knowing that these are the underlying technologies used by these AWS services will enable you to better leverage and optimize your use of Amazon Athena and AWS Glue.

**Choosing the right worker type**

AWS Glue can execute with one of three different worker types. Worker types are also known as **Data Processing Units** (**DPUs**). Each type has different advantages and disadvantages, and they should be chosen based on the use case that we have on hand.

Worker types, or DPUs, come in these configurations:

* **Standard** – This worker type has 16 GB of memory, 4 vCPUs for computing power, 50 GB of attached EBS storage, and includes two Spark executors.
* **G.1X** – G.1X worker types have 16 GB of memory, use 4 vCPUs, and come with 64 GB of attached EBS storage and only 1 Spark executor.
* **G.2X** – The G.2X worker types have 32 GB of memory, use 8 vCPUs, and come with 128 GB of attached EBS storage and only 1 Spark executor.

AWS Glue’s serverless architecture takes advantage of Spark to compute parallelism and can scale horizontally regardless of the type of worker.

If a workload is more memory intensive, AWS Glue jobs that could benefit from vertical scaling should use the G1.X or G2.X worker types.

**Optimizing file splitting**

AWS Glue automatically splits files when processing common traditional file formats such as CSV and JSON, as well as some of the more modern formats, including AVRO and Parquet, by using something called AWS Glue DynamicFrame classes. To learn more about DynamicFrame, you can visit <https://aws.amazon.com/blogs/big-data/work-with-partitioned-data-in-aws-glue/>.

A “file split” is a section in a file that an AWS Glue worker can process independently. Out of the box, file splitting can be performed on line-delimited native formats. This enables the parallel execution of Apache Spark jobs on AWS Glue, spanning multiple nodes. AWS Glue jobs can be optimized to handle files that have a file size between hundreds of megabytes and a couple of gigabytes by leveraging horizontal scaling and attaching more AWS Glue workers.

Splitting files into smaller blocks can also improve performance when using block-based compression formats. This is because the compressed data can be processed in parallel across multiple nodes, allowing for faster data processing and analysis. Each compression block can be read on a split boundary file and processed independently and simultaneously with other files. Compression formats that don’t support splitting, such as gzip, bzip2, Zstandard, LZMA, and so on, do not achieve performance gains from file splitting.

The inputs should use several medium-sized files to achieve horizontal scalability with compression formats or files that can’t be split.

After files have been split, they can be stored in Amazon S3 and accessed as individual objects. These objects can then be deserialized into an AWS Glue DynamicFrame partition, which is a logical container for organizing and processing data. Once the data is in a DynamicFrame, it can be processed using Apache Spark tasks to perform various operations such as filtering, aggregating, and transforming the data. The size of a deserialized partition can be much larger than a disk block file split size of 64 MB, such as for highly compressed formats that can be split like Parquet or larger files that use compression formats such as gzip and cannot be split. When data is deserialized into an AWS Glue DynamicFrame partition, it is not loaded into memory unless required for processing. Instead, Apache Spark’s lazy transformation evaluation is used, which means that transformations on the data are only performed when necessary. This approach helps to avoid memory pressure on AWS Glue tasks, as only the required data is loaded into memory at any given time.

However, when a partition is explicitly cached in memory or spills onto a disk, it can return **Out-of-Memory** (**OOM**) or out-of-disk exceptions. AWS Glue can handle these use cases with AWS Glue worker types that use DPU instances that can be vertically scaled.

**Exceeding YARN’s memory overhead allocation**

Apache YARN is the resource manager used under the hood by Apache Spark and AWS Glue. **YARN** stands for **Yet Another Resource Negotiator**. YARN oversees allocating resources when Spark is running and handling applications workloads. On top of handling memory allocation for each executor running jobs, YARN also oversees the allocation of additional memory assigned to the JVM and metadata that needs to be loaded for the JVM to run correctly. This overhead is allocated 10% of the total executor memory by default. Operations that require a lot of memory, such as table joins or dataset processing with skewed distributions, may require additional overhead and throw an error. If you know that your application will be memory-intensive, it is recommended to allocate additional overheard memory from the start to avoid these types of errors.

Another way to avoid these OOM issues is to use AWS Glue’s vertical scaling feature. Using workers that have been assigned more memory and disk space can also help to avoid this problem.

Lastly, using the dashboard provided by AWS Glue with job metrics can assist in debugging and resolving these OOM issues. For memory-intensive jobs, such as on large datasets with significant skew, use the G1.X and G2.X worker types.

For more information about how to debug these types of issues, visit <https://docs.aws.amazon.com/glue/latest/dg/monitor-profile-debug-oom-abnormalities.html>.

**Leveraging the Apache Spark UI**

Another helpful tool in the Spark arsenal is the Apache Spark UI. The Spark UI can inspect, monitor, and optimize AWS Glue ETL jobs. It allows you to visualize the jobs by providing **Directed Acyclic Graphs** (**DAGs**) of the job’s execution. It can also be used to identify demanding stages and large shuffles and to analyze query plans. The UI will enable you to quickly identify the bottlenecks in your Spark jobs and make adjustments to increase performance and remove those bottlenecks.

More information about the Spark UI can be found at <https://docs.aws.amazon.com/glue/latest/dg/monitor-spark-ui.html>.

**Processing many small files**

It is not uncommon for AWS Glue to routinely handle thousands and even millions of files. This would be more the norm than the exception for use cases involving Amazon Kinesis Data Firehose. In these situations, the Apache Spark driver can run out of memory while reading these files.

Apache Spark version 2.2 can handle 600,000 files on a standard worker type. To increase the number of files that a worker can handle, AWS provides an option to process these files in larger batches. One way to do this is to use a G1.X worker to read the files instead of a standard worker.

Another way is to reduce the number of files processed at one time. This can be achieved by taking advantage of AWS Glue file groupings. Doing so reduces the chance of getting OOM exceptions. The groupFiles and groupSize parameters need to be configured to enable file grouping. Here is a sample call that sets those parameters:

dyf = glueContext.create\_dynamic\_frame\_from\_options("s3",

{'paths': ["s3://path-to-files/"],

'recurse':True,

'groupFiles': 'inPartition',

'groupSize': '2084236'},

format="json")

The purpose of this command is to create a dynamic framework that can then be fed into other ETL jobs for processing. The paths parameter determines the S3 folder that contains the files used to create the frame. The recurse parameter indicates whether subdirectories in this folder should be included for processing.

The groupFiles parameter can be configured to group files in an S3 partition within a folder or across Amazon S3 partitions. In many instances, grouping in each partition is enough to bring down the number of parallel Spark tasks and reduce the amount of memory used by the Spark tasks.

In a battery of tests, ETL jobs configured using this grouping parameter proved to be about 7 times faster than other jobs without this configuration when handling over 300,000 files spanning over 100 Amazon S3 partitions. A significant portion of time is spent with Apache Spark building in-memory indices while listing Amazon S3 files and scheduling many short-running tasks to handle these files. When grouping is enabled, AWS Glue should be able to handle over one million files simultaneously using the standard AWS Glue worker type.

Tuning the groupSize parameter can significantly impact the number of files that can be processed at any one time, which translates into how many files can be produced. Properly tuning this parameter can achieve significant task parallelism, while not correctly configuring it can result in the cluster being underutilized and many of the workers sitting idle.

By default, when the number of files exceeds about 50,000, AWS Glue will automatically enable grouping. AWS Glue can figure out an appropriate value for the groupSize parameter and set it accordingly to minimize the amount of excessive parallelism.

**Data partitioning and predicate pushdown**

Partitioning files is an important technique to split datasets so that these files can be accessed quickly and efficiently. Picking an excellent key to split files is critical to gaining these efficiencies. For example, a dataset may be divided into folders using the ingestion date as the key. In this case, you may have a series of subfolders organized by year, month, and day.

Here is an example of what a directory using this naming scheme could look like:

s3://employees/year=2020/month=01/day=01/

You can use the INSERT INTO statement in HiveQL (Hive’s SQL-like query language) with the HiveOutputFormat and the appropriate file format. For example, you can use the following HiveQL statement to write the results of a SELECT query to the specified S3 location as ORC files:

INSERT INTO TABLE employees

PARTITION (year=2020, month=01, day=01)

SELECT \* FROM source\_table

WHERE year=2020 AND month=01 AND day=01;

Partitioning the data in such a way enables predicate pushdown. Predicate pushdown is a fancy way of saying that by partitioning the data, we don’t need to read all the directories and files to get the results we need when we have a query with a filter.

Predicate pushdown uses filter criteria using partition columns. With predicate pushdown, the data is not read into memory first and then filtered. Instead, because the data is pre-sorted, we know what files meet the criteria being sought, and only those files are brought into memory while the rest of the files are simply skipped.

For example, imagine you have a query like this:

Select \* from employees where year = 2019

Here, the year is the partition, and 2019 is the filter criteria. In this case, the file we had as an example previously would be skipped.

Using pruning can deliver massive performance boosts and greatly reduced response times. Performance can be improved by providing even more filters in the selection criteria, which will eliminate additional partitions.

**Partitioning data while writing to Amazon S3**

The last task during processing is to persist the transformed output in Amazon S3. Once this is done, other services, such as Amazon Athena, can be used for their retrieval. By default, when a DynamicFrame is persisted, it is not partitioned. The results are persisted in a single output path. Until recently, it was only possible to partition a DynamicFrame by converting it into a Spark SQL DataFrame before it persisted. However, nowadays, native partitioning using a key sequence can be used to write out DynamicFrame.

This can be accomplished by setting the partition keys parameter during sink creation. As an example, the following code can be used to output a dataset:

%spark

glueContext.getSinkWithFormat(

connectionType = "s3",

options = JsonOptions(Map("path" -> "$output\_path", "partitionKeys" -> Seq("process\_year"))),

format = "parquet").writeDynamicFrame(employees)

This method creates a DataSink that persists data to an output destination. This destination can be repositories on Amazon S3, Amazon RDS, and so on.

This method allows you to set the data format to be used when persisting the data.

In this case, $output\_path is the output directory in Amazon S3. The partitionKeys parameter specifies the column used as a partition when writing the data to Amazon S3.

When data is written out, the process\_year column is removed from the dataset and is instead used to help form the directory structure. Here is how the directory might look if we listed it out:

PRE year=2020

PRE year=2019

PRE year=2018

PRE year=2017

PRE year=2016

So, what are some good columns to use when selecting partition keys? There are two criteria that should drive this selection:

* Use columns that have a low (but not extremely low) cardinality. For example, a person’s name, phone number, or email would not be a good candidate. Conversely, a column with only one or two values is also not a good candidate.
* Use columns that are expected to be used often and will be used as filters.

For example, if your dataset contains log data, using dates and partitioning them by year, month, and day is often a good strategy. The cardinality should be just right. We should have plenty of results for each day of the logs, and using predicate pushdown would result in only retrieving files for individual days.

There is another benefit to correctly partitioning files, in addition to improving query performance. A proper partition also minimizes costly Apache Spark shuffle transformations for downstream ETL jobs.

Repartitioning a dataset by frequently calling the repartition() or coalesce() functions leads workers to shuffle data. This can have a negative impact on the time it takes to run ETL jobs and will most likely require more memory. By contrast, writing data into Amazon S3 from the start using Apache Hive partitions does not require the data to be shuffled, and it can be sorted locally within a worker node. In Apache Hive, a partition is a way to divide a table into smaller and more manageable pieces, based on the values of certain columns. For example, you can partition a table by date, so that each partition corresponds to a specific day, month, or year. Partitions can improve query performance by enabling the Hive query optimizer to skip over irrelevant partitions, and by allowing data to be stored in a more efficient way.

They can also make it easier to manage and organize large datasets by allowing you to delete or exchange individual partitions as needed.

This concludes the best practices we recommend using to deploy AWS Glue. This list is by no means exhaustive and only scratches the surface. Deploying AWS Glue at scale is not trivial, and architects can build a career purely by mastering this powerful and fundamental service.

**Choosing between AWS Glue and Amazon EMR**

Having learned about Glue and EMR, you must be wondering, to some extent, whether these offerings do a similar job in data processing, and when to choose one over the other. Yes, AWS has a similar offering and that can be confusing sometimes, but both have a specific purpose. Amazon always works backward from the customer, so all these offerings are available because customers have asked for them.

There is a no-brainer for your data cataloging needs; you should always use AWS Glue, and these data catalogs can be utilized when you are processing a job in EMR. However, Glue only supports the Spark framework, and if you are interested in using any other open-source software such as Hive, Ping, or Presto, then you need to choose EMR.

When running data transformation using the Spark platform, you must choose between EMR and Glue. Suppose you are migrating your ETL job from an on-premises Hadoop environment. In that case, you can go with EMR, as it will require minimal code changes, but if you are starting fresh, it’s better to start with Glue, as it is serverless with in-built job orchestration, which will make your work much more manageable. In addition, if you want more control over your environment at the OS level, then you can choose EMR, as Glue abstracts those details. Again, if you want to benefit from DynamicFrame to streamline your data at runtime, then you can choose Glue over EMR. Here are some examples of use cases where you might choose AWS Glue over Amazon EMR, or vice versa:

* **ETL and data preparation:** If you need to extract data from various sources, transform and clean it, and load it into a target data store, AWS Glue is a good choice. It provides a fully managed ETL service with a visual interface and built-in data transformation capabilities. Amazon EMR is more geared toward distributed data processing and analysis, and may not be the best choice for ETL and data preparation tasks.
* **Distributed data processing:** If you have a large dataset that needs to be processed using a distributed computing framework, such as Apache Hadoop or Apache Spark, Amazon EMR is a good choice.

It provides a managed big data platform that can run distributed applications written in various languages and can scale up or down as needed. AWS Glue also supports Apache Spark, but it is more geared towards ETL and data preparation tasks, and may not be the best choice for distributed data processing.

* **Machine learning:** If you need to build and train machine learning models on large datasets, Amazon EMR is a good choice. It integrates with Apache Spark and other machine learning libraries, and provides a range of tools and frameworks for building, training, and deploying machine learning models at scale. AWS Glue also provides some basic machine learning capabilities, such as data preparation and feature engineering, but it is not as comprehensive as the machine learning capabilities provided by Amazon EMR.
* **Data visualization and analysis:** If you need to create interactive data visualizations and perform ad hoc data analysis using SQL, Python, R, or other languages, Amazon EMR is a good choice. It provides a range of tools and frameworks for data visualization and analysis, including Apache Zeppelin, Jupyter, and RStudio. AWS Glue provides some basic data visualization capabilities, such as data profiling and previewing the results of transformations, but it is not as comprehensive as the data visualization capabilities provided by Amazon EMR.

Ultimately, whatever service you choose to run your spark workload, AWS provides multiple features and options. Before November 2021, only Glue was serverless, so if you didn’t want to manage infrastructure, it was always a clear choice to go for Glue, but now after the launch of EMR serverless, you have more choices available to run any Hadoop open-source framework on the cloud, bringing the benefits of serverless capabilities and offloading all the infrastructure admin work to AWS.

Streaming data is becoming more important as customers expect real-time responses. Let’s learn more about taking care of streaming data in the cloud.

**Handling streaming data in AWS**

In today’s world, businesses aim to gain a competitive edge by providing timely tailored experiences to consumers. Consumers expect personalized experiences that meet their specific needs and reject those that don’t, such as when applying for a loan, investing, shopping online, tracking health alerts, or monitoring home security systems. As a result, speed has become a critical characteristic that businesses strive to achieve. Insights from data are perishable and can lose value quickly. Streaming data processing allows analytical insights to be gathered and acted upon instantly to deliver the desired customer experience.

Batch processing data doesn’t allow for real-time risk mitigation or customer authentication, and the customer experience can be ruined – and is hard to recover – if action isn’t taken in real time. Acting on real-time data can help prevent fraud and increase customer loyalty. Untimely data, on the other hand, can inhibit your firm’s ability to grow. The following are some use cases for streaming data:

* **Anomaly and fraud detection in real-time**: Streaming data is used to detect fraudulent activities in real time, by analyzing patterns and anomalies in financial transactions or other types of data.
* **Tailoring customer experience in real time**: Online businesses continuously push features on their websites to respond to ever-changing competition, drive higher sales, and enhance customer satisfaction. With the help of real-time streaming analytics, businesses can understand a given feature’s adoption and tailor the pushed feature to drive more traffic in real time to ensure success.
* **Empowering IoT analytics**: Industrial and commercial IoT sectors have many use cases that can take advantage of real-time streaming analytics. In the IoT space, streaming data is used to track and monitor the real-time status of devices and systems, and to trigger actions based on specific events or thresholds.
* **Financial trading:** In the financial industry, streaming data is used to track market trends, analyze trading patterns, and make real-time decisions.
* **Social media:** Streaming data is used to analyze social media activity in real time and identify trends, sentiments, and patterns.
* **Supply chain management:** Streaming data is used to track and monitor the real-time status of logistics and supply chain operations, and to optimize the flow of goods and materials.
* **Telecommunications:** Streaming data is used to monitor and analyze network traffic and performance in real time, optimize network usage, and detect issues.
* **Traffic management:**Streaming data is used to track and monitor traffic patterns and conditions in real time, optimize traffic flow, and reduce congestion.

To enable real-time analytics, you need to ingest, process, and analyze large volumes of high-velocity data from various sources in real time. Devices and/or applications produce real-time data at high velocity. In the case of IoT devices, tens of thousands of data sources need to be collected and ingested in real time. After that, you store data in the order it was received for a set duration and provide the ability for it to be replayed indefinitely during this time. You need to enable real-time analytics or streaming ETL and store the final data in a data lake or data warehouse.

AWS provides a streaming data processing ability through its serverless offering, Amazon Kinesis, and Kafka offers it through Amazon **Managed Streaming for Apache Kafka** (**MSK**). Let’s begin by learning about Kinesis in more detail.

**Streaming data processing with Amazon Kinesis**

Amazon Kinesis is a cloud-based service that provides businesses with the ability to gather, process, and analyze real-time streaming data, allowing them to gain valuable insights and respond quickly to new information. It offers powerful and cost-effective capabilities for processing large volumes of streaming data at scale, and allows you to select the best tools for your application’s specific requirements. With Amazon Kinesis, you can collect different types of real-time data, including video, audio, logs, clickstreams, and IoT telemetry, and use it for analytics, machine learning, and other purposes.

Amazon Kinesis enables real-time processing of data as it arrives, providing instantaneous responses without having to wait for all the data to be collected. The service provides four fully managed options for collecting, processing, and analyzing streaming data in real time. Let’s explore the different services offered by Kinesis.

**Amazon Kinesis Data Streams (KDS)**

KDS captures data in real time and allows you to build custom, real-time applications for data processing through popular stream processing frameworks. Amazon Kinesis stores incoming data in the order it was received, for a specified duration, and allows for indefinite replays during this time period. This provides the flexibility to process and analyze data multiple times or use it for testing and debugging. By default, KDS retain data for 24 hours, but extended retention is available up to 7 days, and data can be retained for up to 365 days.

KDS has a data shard to ingest and hold data. To simplify this, you can imagine each shard as a single water pipe and data as water stored in this pipe. If you have a large volume of water, you need multiple pipes to consume that, and in the same manner, you need to plan your shards if there is a large volume of data. Each shard can ingest 1 MB of data per second, or 1,000 records per second, and consumers can read data from the shard at the rate of 2 MB of data per second, or 5 reads per second. With the standard consumer model, the lowest latency is 200 ms, and AWS provides an **Enhanced Fan-Out** (**EFO**) data consumer, with which you can achieve 70 ms of latency.

In 2021, AWS launched long-term retention, where you can store data for up to 1 year in KDS. AWS also provides an automatically scaled capacity in response to changing data volumes, which helps to achieve built-in availability and fault tolerance by default.

**Amazon Kinesis Data Firehose (KDF)**

**Kinesis Data Firehose** (**KDF**) is a fully managed service that captures, transforms, and loads data streams into Amazon S3, Amazon Redshift, Amazon Elasticsearch, and Splunk for near real-time analytics through existing business intelligence tools. KDF is serverless, which means no administration and seamless elasticity. It involves direct-to-data-store integration without any coding requirements. You can ingest data in near real time and perform data format conversion to Parquet/ORC on the fly. KDF now provides data delivery to any HTTP endpoint. It supports the AWS API gateway, which helps load data directly to RDS, Amazon SNS, and other popular platforms such as Datadog, Sumo Logic, New Relic, and MongoDB.

For example, KDF can be used in a scenario where a company wants to monitor customer interactions on their e-commerce website in real-time to improve the customer experience. The website generates clickstream data, which includes user clicks, navigation, and other interactions. KDF can be used to capture this clickstream data in real-time, transform it into a structured format, and then load it into Amazon Redshift or Amazon OpenSearch Service for further analysis. The company can then use business intelligence tools such as Amazon QuickSight to visualize the data and gain insights into user behavior, enabling them to make data-driven decisions to improve the customer experience.

Another example could be in the context of IoT data. KDF can be used to capture and process sensor data generated by IoT devices and store it in Amazon S3. This data can then be analyzed to identify patterns or anomalies in device behavior, enabling predictive maintenance or other use cases that improve operational efficiency.

In both use cases, KDF allows for the processing and analysis of streaming data in near real time, enabling businesses to make quick decisions based on current data, and thus improving their operations and customer experience.

**Amazon Kinesis Data Analytics (KDA)**

KDA processes real-time data streams for analytics through SQL and Apache Flink applications. KDA enables advanced analytics on streaming data through built-in functions that can filter, aggregate, and transform data in real time. With sub-second latencies, it can quickly process streaming data, allowing you to analyze and respond to incoming data and events as they occur.

You can run KDA Studio in interactive mode to inspect streaming data, run ad hoc queries, and visualize data using basic charts (bar charts, pie charts, trend lines, and pivot charts) that can be used for a one-click visualization of any output data within a notebook.

**Kinesis Data Analytics Studio** is a web-based interface that allows you to visually build, test, and debug KDA applications. The studio provides a range of features, such as an SQL editor, a code editor, a data preview tool, and a debugger, to help you build and test your analytics applications.

KDA also has in-built support for **Apache Flink** for more sophisticated applications or if you prefer Java. The following are the key benefits of KDA for Apache Flink:

* **Simple programming** – Easy-to-use and flexible APIs make building apps fast
* **High performance** – In-memory computing provides low latency and high throughput
* **Stateful processing** – Durable application state saves strong data integrity with exactly-once processing

KDA offers native integration with Amazon KDS, Amazon KDF, and Amazon MSK, enabling you to process and analyze streaming data in real time. With KDA, you can use built-in functions to filter, aggregate, and transform data streams for advanced analytics. KDA is built on Apache Flink, which provides low-latency processing and high-throughput ingestion for real time data streams. Additionally, KDA supports other data sources that can produce data directly to Apache Flink, enabling you to bring in data from other sources and process it in real-time alongside your Kinesis data.

**Amazon Kinesis Video Streams** (**KVS**) is another offering that supports secure video streaming from connected devices to AWS for analytics, machine learning, and other processing. It can be beneficial for media streaming and processing data from video streams, such as CCTV.

Real-time data processing using Kinesis can be life-saving for healthcare and emergency services. Real-time analytics can be used in clinical healthcare to monitor patient safety, personalize patient results, assess clinical risks, and reduce patient readmission. By processing and analyzing data in real time, healthcare organizations can gain insights and take actions that can improve patient care and outcomes. For example, real-time analytics can be used to identify potential risks and intervene before they lead to adverse events, or to tailor treatment plans based on the latest data and evidence. They can also help to reduce the number of readmissions by identifying and addressing the root causes of readmission. Overall, real-time analytics can help healthcare organizations to deliver better, more personalized care to their patients. Wearable health device data, combined with geospatial data, can proactively notify relatives, caregivers, or incident commanders for a timely response. Kinesis has many use cases; you can learn more about them by referring to the AWS whitepaper *Streaming Data Solutions on AWS with Amazon Kinesis* – <https://aws.amazon.com/kinesis/whitepaper/>.

Apache Kafka is one of the most popular technologies to address streaming use cases and build an organization’s **Enterprise Service Bus** (**ESB**) for communication between different applications. AWS provides a managed Kafka offering to address those use cases. Let’s learn more about it.

**Amazon Managed Streaming for Apache Kafka (MSK)**

Apache Kafka is one of the most popular open-source platforms for building real-time streaming data pipelines and applications. Managing Apache Kafka clusters in production can be difficult, as it requires careful planning and ongoing maintenance. Setting up and configuring Apache Kafka requires provisioning servers and performing manual tasks. Additionally, you must continuously monitor and maintain the servers to ensure their reliability, security, and performance. Tasks involved in managing a cluster encompass several activities such as replacing failed servers, orchestrating patches and upgrades, designing the cluster for high availability, storing data durably, setting up monitoring and alarms, and planning scaling events to support changing workloads. Overall, managing Apache Kafka clusters in production can be a complex and time-consuming process.

Amazon MSK is a fully managed service provided by AWS that allows users to build and run production applications on Apache Kafka without requiring in-house Apache Kafka infrastructure management expertise. With Amazon MSK, AWS handles the infrastructure management tasks such as server replacement, patching and upgrades, and high availability design. This enables users to focus on building their applications and leveraging the features of Apache Kafka, such as its ability to handle real-time, high-throughput, fault-tolerant data streaming. MSK allows you to launch clusters across multiple AZs with customizable configurations to achieve high-performance and high-availability environments. You can scale out compute by adding brokers to existing clusters, and scale up storage without downtime.

Amazon MSK is a fully managed service that allows you to capture, process, and derive insights from log and event streams in real time. Amazon MSK enables users to ingest events and analyze data streams in real-time using Apache Zeppelin notebooks, providing near-instant insights. With MSK, you can build centralized data buses using the Apache Kafka log structure, enabling the creation of real time, secure, and centralized data buses. Additionally, you can create event-driven systems that can react to digital changes within your applications and business infrastructure in real time. By simplifying the process of building, running, and maintaining streaming data pipelines and event-driven systems, MSK streamlines the data processing workflow.

Amazon MSK provides native Apache Kafka APIs to enable you to work with data lakes, stream data to and from databases, and power machine learning and analytics applications. If you’re already running Apache Kafka on-premises, on EC2, or with another managed service provider, you can migrate your existing Apache Kafka applications to AWS and run them without any changes to the application code. This ensures open-source compatibility, and you can continue using custom and community-built tools like MirrorMaker and Apache Flink. Apache Kafka MirrorMaker is a tool that enables you to replicate data between Apache Kafka clusters. It works by reading data from a source Kafka cluster, and then writing the data to a target Kafka cluster. MirrorMaker can be used for various purposes, such as disaster recovery, data backup, data migration, and cross-region replication.

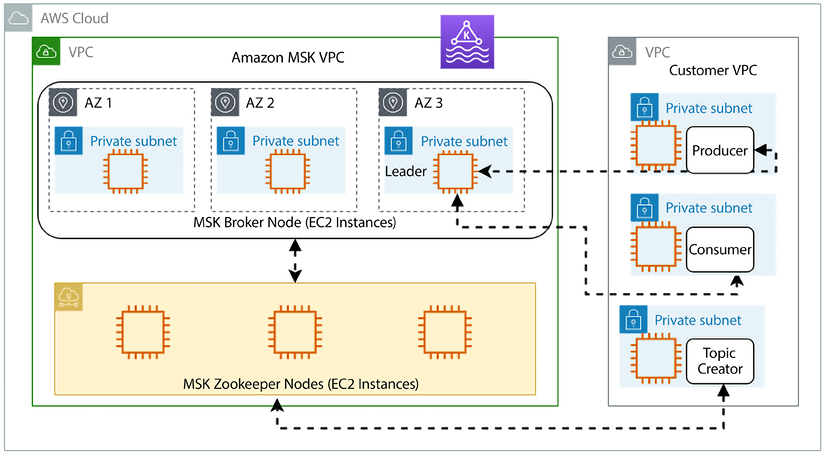
MirrorMaker provides various configuration options, such as the ability to filter data based on topic, partition, or offset, and to customize the consumer and producer settings. It also provides features such as message transformation, error handling, and monitoring, to help you manage the data replication process.

Let’s look into Amazon MSK cluster architecture in detail.

**Amazon MSK cluster architecture**

With MSK, you can focus on your applications and use cases, rather than worrying about the underlying infrastructure and operational tasks. MSK handles the provisioning and maintenance of the Kafka clusters and provides features such as automatic scaling, self-healing, and monitoring to ensure high availability and performance. You can see the complete list of operations available in the Amazon MSK API reference guide – <https://docs.aws.amazon.com/msk/1.0/apireference/resources.html>.

The following diagram provides insight into an Amazon MSK cluster. Amazon MSK provides the capability to create clusters that utilize Apache Kafka versions up to 3.2.0. AWS keeps on adding new versions as they become available, which allows users to take advantage of the latest features and improvements of Apache Kafka without worrying about infrastructure management. Please refer to the Amazon MSK developer guide for the latest information on the Amazon MSK-supported version of Kafka – <https://docs.aws.amazon.com/msk/latest/developerguide/supported-kafka-versions.html>:



*Figure 10.3: Amazon MSK architecture*

The architecture of Amazon MSK consists of the following components:

* **Kafka clusters**: An MSK cluster is a group of Amazon EC2 instances that run the Apache Kafka software. An MSK cluster can have one or more Kafka brokers and can be scaled up or down as needed to support the processing and storage requirements of your applications.
* **Zookeeper**: Apache Kafka uses Apache Zookeeper to store metadata and coordinate the Kafka brokers. In an MSK cluster, Zookeeper is automatically deployed and configured as part of the cluster.
* **Producers and consumers**: Producers are applications or systems that produce data and send it to a Kafka cluster, while consumers are applications or systems that receive data from a Kafka cluster and process it. In an MSK cluster, you can use the Apache Kafka producer and consumer APIs to produce and consume data, or use higher-level libraries and tools such as Kafka Connect or KDA to simplify the process.
* **Data storage**: Apache Kafka stores data in topics, which are partitioned and replicated across the Kafka brokers in the cluster. In an MSK cluster, data is stored in Amazon EBS volumes or Amazon S3 buckets, depending on the storage type you choose.
* **Networking**:Amazon MSK uses Amazon VPC to provide a secure and isolated network environment for your Kafka clusters. You can specify the VPC, subnets, and security groups to use when you create an MSK cluster.

By default, MSK creates a new VPC and subnets for a cluster and configures the security groups to allow traffic between the Kafka brokers and the producers and consumers. However, you can also choose to use an existing VPC and subnets, or create a custom VPC and subnets using AWS CloudFormation templates. Once the MSK cluster has been created, the Kafka brokers and the Zookeeper nodes are launched in the specified subnets and are automatically registered with the security groups. The Kafka brokers and the Zookeeper nodes communicate with each other and with the producers and consumers using private IP addresses within the VPC.

You can also use VPC peering or a VPN to connect your MSK cluster to other VPCs or on-premises networks and enable communication between the Kafka brokers and the producers and consumers using public or private IP addresses. Overall, the networking configuration of an MSK cluster determines how the Kafka brokers, the Zookeeper nodes, and the producers and consumers communicate with each other and with the rest of your network. It is important to carefully plan and configure the networking for your MSK cluster to ensure the high availability and performance of your Kafka-based applications.

Amazon MSK provides a range of tools and features to help you monitor and manage your Kafka clusters and applications. This includes integration with Amazon CloudWatch for monitoring, Amazon CloudFormation for infrastructure as code, and the AWS Management Console for visual management.

At re:Invent 2021, AWS launched a new offering called MSK Serverless, which helps you to easily run Apache Kafka clusters without needing to adjust the size of the cluster capacity or worry about overprovisioning. With Amazon MSK Serverless, you can scale input and output (I/O) instantly without the need to manually manage capacity or partition reassignments. This service is designed to provide a simplified and cost-effective way to stream and retain data with pricing based on throughput. Additionally, you only pay for the data volume that you stream and retain, making it a cost-efficient option for managing streaming data workloads. If you are planning to migrate your existing Apache Kafka workload to MSK, AWS has published an MSK migration guide to help you, which you can find by visiting the link here – <https://docs.aws.amazon.com/whitepapers/latest/amazon-msk-migration-guide>.

Handling data cataloging for streaming data can be complicated. Let’s learn about AWS’s new offering for streaming data cataloging.

**Data cataloging for streaming data using AWS Glue Schema Registry (GSR)**

Data streaming technologies like Amazon MSK, Apache Kafka, and Amazon Kinesis are widely used across various industries to capture and distribute data generated by applications, websites, or machines. They serve as a highly available data transport layer, which separates the data-producing applications from the real-time routing, analytics, and machine learning data processors. This decoupling allows for greater scalability and flexibility, as well as the ability to quickly and easily respond to new information and insights in real time. Industries such as finance, healthcare, manufacturing, retail, and more use data streaming technologies to improve operational efficiency, enhance customer experiences, and gain competitive advantages through data-driven decision making.

However, it’s hard to coordinate data formats and structures (schemas) across so many systems owned by different teams. Teams have difficulty using each other’s data, and downstream teams have to adapt when formats change. They often build complex tools or custom code to ensure data continuity when schemas evolve.

Cataloging streaming data is challenging due to the continuous ingestion of data and various formats. AWS introduced the **Glue Schema Registry** (**GSR**) to manage data schemas for streaming data catalogs. It enables users to discover, control, and evolve data schemas easily.

AWS GSR allows you to centrally manage and store schemas for your data, and provides tools and APIs to help you create, modify, and version schemas as your data evolves. It also provides a schema discovery feature that allows you to discover and extract schemas from data sources, and a schema validation feature that allows you to validate data records against the stored schemas.

AWS GSR is designed to be used with other AWS Glue services, such as AWS Glue ETL jobs and AWS Glue DataBrew, to help you build and operate data pipelines that are based on well-defined and standardized data schemas. It can also be used with other AWS services that support data integration, such as Amazon KDS and Amazon S3, to help you ensure the integrity and quality of the data being ingested and processed.

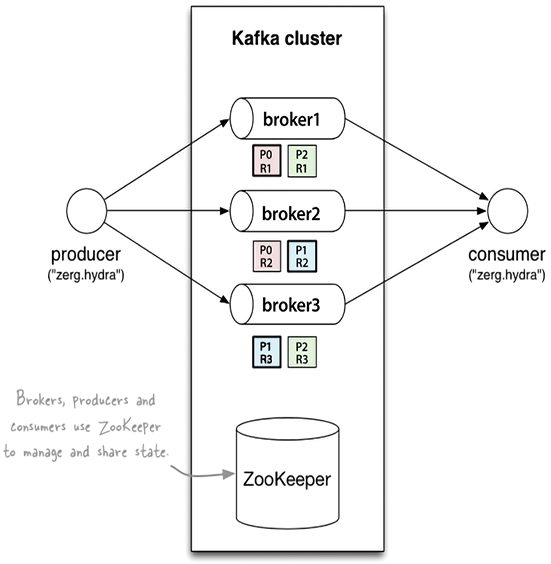
AWS GSR prevents downstream failures caused by schema changes and boosts productivity for developers by enforcing schemas where data is produced before the data gets sent downstream. The customer can customize schema enforcement using one of eight compatibility modes (Backward, Backward\_All, Forward, Forward\_All, Full, Full\_All, Disabled, or None). The schema registry’s open source serializers allow for the efficient serialization of data into a compact binary and compressed format.

This results in reduced data transfer and storage costs for customers compared to using uncompressed JSON.

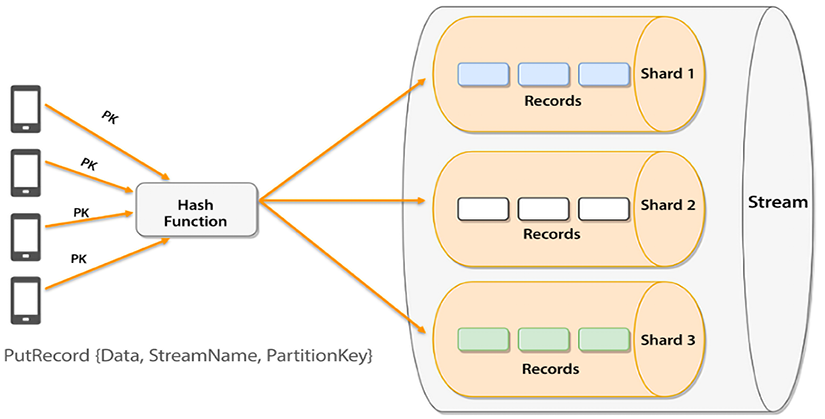
Now that you have learned about Kinesis and MSK, let’s see the significant differences and when to choose one versus the other.

**Choosing between Amazon Kinesis and Amazon MSK**

AWS launched Kinesis in 2013, and it was the only streaming data offering until 2018 when AWS launched MSK, in response to the high demand from their customers for managed Apache Kafka clusters. Now, both are similar offerings, so you must be wondering when to choose one versus the other. If you already have an existing Kafka workload on-premises or are running Kafka in EC2, it’s better to migrate to MSK, as you don’t need to make any changes in the code. You can take the help of the existing MirrorMaker tool to migrate. The following diagrams show key architectural differences between MSK and Kinesis:



*Figure 10.4: Amazon MSK architecture*



*Figure 10.5: Amazon Kinesis architecture*

As shown in the preceding diagrams, there are similarities between the MSK and Kinesis architectures. In the MSK cluster, you have brokers to store and ingest data, while in Kinesis, you have shards. In MSK, you need Zookeeper to manage configuration while AWS takes care of this admin overhead in Kinesis. The following table shows some more differentiating attributes:

|  |  |
| --- | --- |
| **Amazon MSK** | **Amazon Kinesis** |
| You need to decide the number of clusters, brokers per cluster, topics per broker, and portions per topic to operate the MSK cluster. | You need to decide on the number of data streams and shards per stream to operate the Kinesis data stream. |
| MSK operates under the cluster provision model, which has a higher cost than Kinesis. | Kinesis needs a throughput provision model which has a lower cost as you only pay for the data you use. |
| You can only increase the number of partitions; decreasing partitions is not possible. | You can increase or decrease the number of shards. |
| MSK integrates with a few AWS services, such as KDA for Apache Flink. | Kinesis fully integrates with many AWS Services such as Lambda, KDA, etc. |
| MSK has no limit on throughput. | Kinesis throughput scales with shards support for up to 1 MB payloads. |
| MSK is open source. | Kinesis is not open source |

*Table 10.1: Comparison between MSK and Kinesis*

In a nutshell, if you are starting a brand new streaming data ingestion and processing pipeline, it’s better to go for Kinesis due to its ease of use and cost. Often, organizations use Kafka as an event bus to establish communication between applications, and in that case, you should continue with Kafka.

**Summary**

In this chapter, you started by understanding why to choose the cloud for big data analytics. You learned about the details of Amazon EMR, which is the AWS Hadoop offering on the cloud, and that in 2021, AWS also launched the server offering of EMR. You learned about EMR clusters, file systems, and security.

Later in this chapter, you were introduced to one of the most important services in the AWS stack – AWS Glue. You learned about the high-level components that comprise AWS Glue, such as the AWS Glue console, the AWS Glue Data Catalog, AWS Glue crawlers, and AWS Glue code generators. You then learned how everything is connected and how it can be used. Finally, you learned about the recommended best practices when architecting and implementing AWS Glue. You also learned when to choose Glue over EMR, and vice versa.

Real-time insights are becoming essential to the modern customer experience, and you learned about handling streaming data in the cloud. You learned about the AWS streaming data offering Amazon Kinesis and the different services in the Kinesis portfolio, including KDS for data ingestion and storage, KDA for data processing and query, and KDF for direct data loading without any code. You also learned about an AWS-managed Kafka offering called Amazon MSK, along with the MSK cluster architecture. You learned about AWS GSR for cataloging streaming data. Finally, you saw a comparison between MSK and Kinesis and discovered when to choose one versus the other.

In the next chapter, you will learn how to store and consume your data in AWS using its data warehouse, data query, and data visualization services.