**Chapter 9. Big Data, Analytics and Machine Learning Services**

**A NOTE FOR EARLY RELEASE READERS**

With Early Release ebooks, you get books in their earliest form—the authors’ raw and unedited content as they write—so you can take advantage of these technologies long before the official release of these titles.

This will be the 13th chapter of the final book. Please note that the GitHub repo will be made active later on.

If you have comments about how we might improve the content and/or examples in this book, or if you notice missing material within this chapter, please reach out to the editor at *mpotter@oreilly.com*.

In the world of information technology, data is generated at a huge volume. This data can be just the user information of all registered users on online food ordering applications or capturing real-time user actions on the application. The data generated at large volume is referred to as Big Data. If you have a use case to store this data, you can utilize the storage solutions we discussed in Chapter 10 based on your requirements. This chapter focuses on how we can process the data at high volume. How can we generate insights out of data already present in storage or live streaming data by running data analytics or machine learning models on top of it? For example, determining the most ordered food item based on location or restaurants with highest rating in particular locality.

The first part of this chapter introduces you to AWS big data, live streaming and analytics services such as Amazon Elastic MapReduce ([EMR](https://aws.amazon.com/emr/)), [AWS Glue](https://aws.amazon.com/glue/), [Amazon Athena](https://aws.amazon.com/athena/), [Amazon Kinesis](https://aws.amazon.com/kinesis/), [Amazon Redshift](https://aws.amazon.com/redshift/) and [Amazon Quicksight](https://aws.amazon.com/quicksight/). The second section explores how you can run Machine Learning ([ML](https://aws.amazon.com/free/machine-learning/)) workloads on AWS Cloud and different services supporting it.

**AWS Big Data and Analytics**

Information is vital to make business decisions or serve our customers better, but the volume of data is increasing, ranging from terabytes to petabytes (and more) along with the variety of data—the data can be in any form. We require specific tools for storage and processing of big data. The traditional tools can become a bottleneck as we can no longer operate on a single machine for data processing. Another challenge is the velocity at which this data is produced—we also require tools to consume the data and produce insights from it in near real time to gain maximum output. In this section, we’ll talk about data processing tools such as Amazon EMR and AWS Glue; analytics tools such as Amazon Athena and Amazon Redshift; live streaming ingestion and analytics via Amazon Kinesis offerings; and business intelligence service called Amazon Quicksight.

**Amazon Elastic MapReduce**

We introduced you to Hadoop in Chapter 8, an open-source software popular for big data processing. Amazon EMR is a managed service useful for execution of data processing frameworks and tools such as [Map Reduce](https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html), [Apache Spark](https://spark.apache.org/), [Apache Hive](https://hive.apache.org/), [Apache HBase](https://hbase.apache.org/), and [Presto](https://prestodb.io/). The set up and management of large clusters requires time and expertise but Amazon EMR makes it easy by offering a managed service where we can launch clusters in minutes and run large data processing workloads.

Open source Hadoop cluster defaults to [Hadoop Distributed File System](https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html) (HDFS) for data storage, and we can also explore using different tools available in the community to extend the storage to Amazon S3. HDFS is tied to the local disks which disappears once the cluster terminates. Now, as our use-case is processing big data, our storage will grow over time and since HDFS is attached to compute instances, the number of instances will also increase. There is a possibility that our use-case has high storage requirements but the compute requirement is relatively less, so we’re paying for the compute but not using it at full capacity. This is where the [EMR File System](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-fs.html) (EMRFS) appears as a better solution.

**EMRFS**

EMRFS uses S3 as a file system for our data processing instead of local HDFS, so essentially it’s a connector that links EMR clusters to S3. EMRFS ensures streaming of data directly to S3 and uses HDFS as intermediate storage.

Before we dive deep into EMRFS, please understand that EMRFS is not a solution to every problem and you should not see it as a replacement to HDFS. HDFS can still be a better option for jobs with iterative reads on the same dataset or I/O intensive workloads (meaning workloads that require frequent disk access and retrieving data over network, as these scenarios could prove to be latency intensive operations). You can use the combination of Amazon S3 and HDFS to address the temporary data storage (data is lost on cluster termination) limitation of HDFS by leveraging tools such as [s3-dist-cp](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/UsingEMR_s3distcp.html) tool to copy data from S3 to HDFS or from HDFS to S3.

Let’s look into some of the benefits that EMRFS offers over traditional HDFS:

* In scenarios when you require the same data to be accessed by multiple clusters, we need to copy data from one cluster to another since HDFS is associated with a single cluster and not shared among all clusters. We can save this cost by using the EMRFS as a storage option. EMRFS allows multiple clusters to access the data from the same place, decoupling compute and storage completely, and also offers flexibility to scale them independently.
* In addition to the above point, we might not always require our EMR cluster to be running. So we can terminate the cluster and storage is maintained in S3 saving compute costs.
* To ensure data availability, HDFS replicates the data at multiple nodes as per the replicator factor. For example, with a replication factor as 3, we’ll be paying 3x the storage cost. EMRFS is backed by S3 so we don’t have to worry about [data durability](https://aws.amazon.com/s3/faqs/) as it is already handled by S3.

The above data points give a clear picture of how EMRFS is a useful utility to consider for storing data. We can make best out of any tool if we follow certain best practices associated with it—let’s discuss some tips around EMRFS:

* The data should be partitioned so that the EMR cluster should fetch only the data that is required for processing. This ensures faster data retrieval along with reduced costs.
* The file size should be optimized in a way that we don’t have too many files or too few files. As we start off EMRFS usage, try to avoid files with size less than 128 MB because it will ensure less calls to S3 along with HDFS requests.
* We also recommend considering data compression, ensuring less storage space usage on S3. This helps in reducing storage costs at two fronts:storage cost and network costs for data retrieval.
* The data access patterns can vary based on business use-case. It could be possible that you’re only interested in a subset of columns across the columns or a subset of rows across the rows. [Apache Parquet](https://parquet.apache.org/) and [Apache ORC](https://orc.apache.org/) are columnar file formats that give increased read performance for prior use-case and [Apache Avro](https://avro.apache.org/docs/1.7.6/spec.html), a row optimized file format is useful for latter use-case.

Now you should have a fair grasp around the concept of EMRFS as a storage option. Now let’s explore more on other EMR features and specifications.

**EMR Cluster Considerations**

The data processing jobs run on a set of servers, and instance types can be chosen based on workload requirements. As a compute option, we can choose to run an EMR cluster on [EC2 instances](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-ec2-instances.html), [EKS](https://docs.aws.amazon.com/emr/latest/EMR-on-EKS-DevelopmentGuide/emr-eks.html) or go completely [Serverless](https://docs.aws.amazon.com/emr/latest/EMR-Serverless-UserGuide/emr-serverless.html" \t "_blank).

The EMR cluster architecture can have three types of nodes, namely master node, core node and task node as shown in [Figure 9-1](https://learning.oreilly.com/library/view/system-design-on/9781098146887/ch09.html#fig_1_emr_cluster_nodes).

*Master Node*

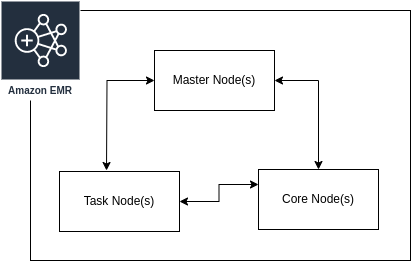
Master node is the primary node in the EMR cluster. It uses YARN resource manager service for application resource management and runs HDFS NameNode service to track status of jobs submitted on cluster along with instance groups health monitoring.

*Core Node*

The core nodes are responsible for running the Data daemon for HDFS data storage and Task Tracker daemon to perform computation tasks on data. The EMR cluster can have a maximum of one core [instance group or instance fleet](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-instance-group-configuration.html).

*Task Node*

Task nodes don’t provide data storage—they should be used as extra computation power to the EMR cluster. EMR by default ensures that the application master of any job runs on the core node to ensure the job doesn’t terminate if any of the task nodes is terminated (as in case of EC2 spot instances). We can configure up to 48 task instance groups depending on the job requirements.



**Figure 9-1. EMR Cluster nodes**

EMR cluster allows customers to choose from the available applications and it will auto-install all the software without any effort from our end. There may also be a requirement to install any custom software or modify a configuration for nodes on EMR nodes. To support this, we can use [EMR bootstrap actions](https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-plan-bootstrap.html) which executes the action after the cluster instances are up.

The EMR cluster can consist of a large number of nodes depending on the requirement and this large number will definitely add to the resources cost we incur from AWS. We can introduce some cost saving mechanisms as described below:

* Amazon EMR provides configuration to auto terminate clusters. This ensures the cluster is auto terminated once the submitted jobs are completed.
* You may also have requirements for long running clusters as well. If you’re sure of capacity requirements, you can explore [reserved instances](https://aws.amazon.com/ec2/pricing/reserved-instances/) and [AWS saving plans](https://aws.amazon.com/savingsplans/compute-pricing/).
* If the EMR workloads are non-critical, you can use EC2 spot instances (discussed in Chapter 11). The spot instances are offered at up to 90% discount as compared to on-demand instances and can help substantially reduce the costs. You can also set up clusters with a mix of spot and on-demand instances to ensure the cluster is always up and running but also cluster pricing in mind.
* For the variable workloads on long running clusters, you can use automatic scaling policies on instance groups to add or remove instances.

Before we close our discussion on Amazon EMR, let’s ponder upon few of the considerations to ensure high reliability of workloads running on EMR clusters:

* The instances in clusters should be spread across AZs to avoid any AZ downtimes. By default, the [Hive](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-hive.html) metastore is stored on the primary node’s file system. We recommend storing the metastore [outside of the EMR cluster](https://docs.aws.amazon.com/emr/latest/ReleaseGuide/emr-metastore-external-hive.html) such as inside multi-AZ RDS cluster or Amazon Aurora to avoid data loss on cluster termination.
* We recommend using EMR multi-master instead of a single master node. The master node is a single point of failure and if it goes down, the entire cluster goes down. Multi-master node setup ensures to provide this safety net.
* The critical data should be kept in S3 instead of local HDFS to avoid any data dependency issues.
* We recommend EC2 spot instances only for task nodes and not for core nodes. As HDFS is managed by core nodes for data storage, termination of these nodes might lead to data loss.

To address the requirement of spawning EMR clusters at regular intervals, we can leverage Amazon EventBridge and automate the workflow via AWS Step Function like so: create an EMR cluster with specific configuration, submit jobs to the cluster and once the jobs are complete, terminate the cluster. In some scenarios, we just care about executing the ETL jobs without any worry of instance or storage management. AWS Glue is a completely serverless data integration application offered by AWS to perform analytical tasks on big data.

**AWS Glue**

AWS Glue is a serverless data integration service helping to make sense of data with different features. Before we dig deeper into these features, let’s quickly understand some of the terminology associated with AWS Glue:

*Classifier*

A [Classifier](https://docs.aws.amazon.com/glue/latest/webapi/API_Classifier.html) validates whether it can handle the data available in specific format and if it can, then classifies into a StructType object. We can use Classifiers offered by AWS Glue or define a custom Classifier as [needed](https://docs.aws.amazon.com/glue/latest/dg/add-classifier.html#classifier-built-in) such as data file is not in format being supported by AWS Glue.

*Metadata*

The metadata is the inferred schema by the Classifier from the available data in any of our datastores.

*Database*

A database is a place where you keep metadata tables. A single table can be associated to a single database only and in case the database is not specified, AWS Glue uses the default database.

*Data Catalog*

A Data Catalog maintains databases which then consist of one or more metadata tables. These tables can be useful as source and target

*Data Crawler*

The responsibility of Data Crawler is to crawl the data from data sources such as Amazon S3, figure out the schema via Classifier and then create the required tables or partitions in Glue Data Catalog. For use cases where the data schema frequently changes, we can run [Crawler on schedule](https://docs.aws.amazon.com/glue/latest/dg/incremental-crawls.html) and it automatically figures out modifications since the last run and creates new tables or partitions as per requirement.

*Table Partitioning*

Partitioning is a way to improve the query performance on data. Let’s understand it with the help of an example as shown in [Figure 9-2](https://learning.oreilly.com/library/view/system-design-on/9781098146887/ch09.html#fig_2_data_partitioning_folder_wise_in_amazon_s3). Our application processes sales data and is [stored in Amazon S3](https://docs.aws.amazon.com/glue/latest/dg/crawler-s3-folder-table-partition.html) divided into folders with parent folder as year and child folder as month and days. It can have further sub folders such as hours or minutes depending on type and amount of data. Now if the data is partitioned in such a fashion, it becomes very easy to query for it. Extracting data for February 2022 means going to the folder for Year 2022 and then to sub folder Month 02 making the query much faster. There can be multiple partition keys for a table and we can create a [partition index](https://docs.aws.amazon.com/glue/latest/dg/partition-indexes.html), a subset of partitions to avoid loading all the partitions.

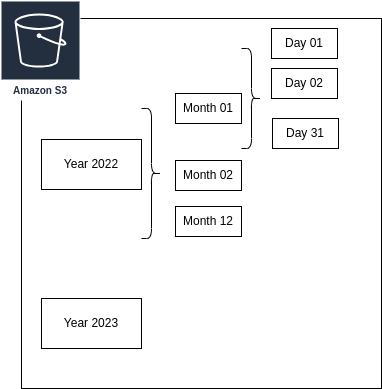
*Data Engine*

Data Engine is helpful in running data processing jobs on top of the data. AWS Glue supports three data engines at the time of this writing, namely [AWS Glue for Apache Spark](https://docs.aws.amazon.com/glue/latest/dg/etl-jobs-section.html), [AWS Glue for Ray](https://docs.aws.amazon.com/glue/latest/dg/ray-jobs-section.html) and [AWS Glue for Python Shell](https://docs.aws.amazon.com/glue/latest/dg/add-job-python.html) and we can choose one based on our workload requirements.

* With AWS Glue for Apache Spark, we can write ETL code in Python and Scala languages and the code executes in a distributed environment. The Spark environment on AWS Glue can be assumed as an Serverless EMR cluster being operated by AWS. The Spark engine is supported for both batch processing and live streaming data. AWS Glue also addresses a very general problem of disk spilling due to data shuffling (data redistribution within the cluster) by providing [Cloud Shuffle plugin](https://docs.aws.amazon.com/glue/latest/dg/cloud-shuffle-storage-plugin.html). This plugin allows the use of Amazon S3 or any other object storage in data spilling situations. Further, this plugin is available as open source software so can be used in custom Spark jobs as well.
* With AWS Glue for Python, a single node Python engine is offered to run workloads written in Python. It offers default integration with open source libraries such as numpy, pandas, etc and we can add our custom libraries as well. The one problem that customers might face is workload scaling as data grows, which can be resolved by using [AWS Glue for Ray](https://docs.aws.amazon.com/glue/latest/dg/how-it-works-engines.html).
* AWS Glue for Ray allows you to execute Python code in a distributed environment set up and is scalable to hundreds of nodes. This processing engine is based on open source compute framework [Ray](https://www.ray.io/) which is extensively used to run Python workloads at large scale such as deep learning models.

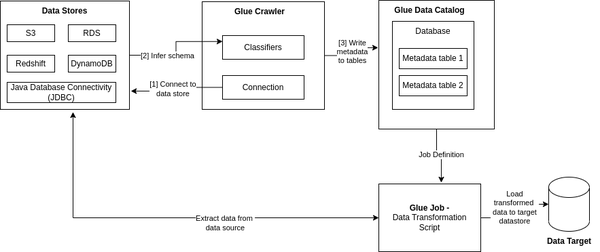
*Data Processing Units*

AWS Glue allows customers to configure types of workers to run the data processing jobs. These workers are referred to as Data Processing Units (DPUs). We can [analyze our workload requirements](https://docs.aws.amazon.com/glue/latest/dg/monitor-debug-capacity.html) for best suited configuration of DPUs.



**Figure 9-2. Data partitioning folder wise in Amazon S3**

Let’s combine our understanding of different concepts into a simple architecture around how AWS Glue operates in [Figure 9-3](https://learning.oreilly.com/library/view/system-design-on/9781098146887/ch09.html#fig_3_aws_glue_architecture_overview). Once the structure of data is created, we can create and run AWS Glue ETL jobs with any of the data engines of our choice for further visualization and analysis.



**Figure 9-3. AWS Glue architecture overview**

The above diagram illustrates a high level picture of how AWS Glue manages and runs ETL jobs. We’ll now discuss some additional features and offerings of AWS Glue that you can utilize on need basis:

* AWS Glue offers Glue Studio with support for visual workflows, built-in transformations, and Glue Studio notebook. This helps to run tests on the data and once we’re happy with testing, the code from the notebook can be imported to an ETL job.
* AWS Glue offers a [Data Quality](https://aws.amazon.com/glue/features/data-quality/) check feature that automatically figures out conditions or logics on which data quality is validated We can further override them as required. Consider an example of sales data: A sales company started off its operations in the year 2021 but there is some data with the year marked as 2019. Here, we can be sure that this is invalid data and should be scraped out or require alerts put on top of it. This feature is available for both data at-rest and in-transit.
* There is another validation feature of [Personal Identifiable Information](https://docs.aws.amazon.com/glue/latest/ug/detect-PII.html) (PII) data supported by AWS Glue. This can be used to detect sensitive data and take any appropriate action such as masking it.
* To reduce compute cost and configure an optimal number of DPUs to run AWS Glue jobs, we can leverage the [Auto Scaling](https://docs.aws.amazon.com/glue/latest/dg/auto-scaling.html) feature. This helps to automatically add or remove workers as per workload requirement. We can also configure a maximum worker number to ensure AWS Glue doesn’t configure workers above a particular limit.
* In Chapter 11, we discussed EC2 Spot instances and how they can be helpful in reducing overall compute costs. AWS Glue offers similar functionality with [Flex job type](https://aws.amazon.com/blogs/big-data/introducing-aws-glue-flex-jobs-cost-savings-on-etl-workloads/) offering up to 34% cost savings. We can leverage this in scenarios of non-critical workloads.
* There may always be use cases where it would be useful to [schedule](https://docs.aws.amazon.com/glue/latest/dg/monitor-data-warehouse-schedule.html) the crawlers or jobs to run at a particular time or on regular intervals. AWS Glue inherently provides support for this. We can additionally trigger jobs and crawlers on external events via AWS Eventbridge.
* AWS Glue offers easy integration with [GitHub and AWS CodeCommit](https://docs.aws.amazon.com/glue/latest/ug/edit-job-add-source-control-integration.html) to manage source version control for our AWS Glue jobs.
* AWS Glue provides [AWS Glue Schema Registry](https://docs.aws.amazon.com/glue/latest/dg/schema-registry.html) as another useful feature. As the name dictates, it’s a registry where we can publish our schemas and can also enforce on data streaming service integrations such as Amazon MSK, Apache Kafka, Kinesis Data Streams, Kinesis Data Analytics and AWS Lambda.

From the above long list of features, it is evident that AWS Glue is a powerful ETL service and helps a lot to reduce operational overhead and can operate at large scale. After navigating through Amazon EMR and AWS Glue, you might be wondering how the Spark application running on Amazon EMR serverless option differs from AWS Glue Spark. Let’s throw some light on this in Table 13-1 as both of these options look quite similar.

|  |  |  |
| --- | --- | --- |
| Parameter | Amazon EMR Serverless | AWS Glue |
| Feature functionality | Data processing and analytics tool which leverages open-source software. | An end-to-end ETL solution with abstraction over software and it offers AWS customized solutions with focus on data integration, data catalog and running transformations on data sources. |
| Operational overhead | Fully managed service | Customers specify the number and type of DPUs. |
| Supported big data applications | Apache Spark and Apache Hive. | Apache Spark and Python jobs. |
| Table 9-1. Amazon EMR Serverless and AWS Glue comparison | | |

AWS Glue and Amazon EMR vary on their base functionality and we as customers should clearly lay out our requirements to choose one for our workloads. In the next section, we’ll discuss [Amazon Athena](https://aws.amazon.com/athena/), a service helpful to query and analyze data stored in Amazon S3.

**Amazon Athena**

Amazon Athena is a fully managed serverless big data analysis tool which offers SQL query support on top of data stored in Amazon S3. The data analysis doesn’t require any transfer of data to any other data storage, the data can be directly read from S3 objects and can be queried upon. We can compare Amazon Athena with [Presto](https://prestodb.io/) running on an Amazon EMR cluster which provides similar functionality. The main advantage of such solutions is that data analysis can be performed on raw data without any transformation at the customer’s end.

Amazon Athena requires that we as customers define the databases and tables with schema. We can do this task manually or leverage AWS Glue crawler/[Apache Hive metastore](https://docs.aws.amazon.com/athena/latest/ug/connect-to-data-source-hive.html) to perform this task for us. Once the database and table is defined, we can query on this data with simple SQL. The key consideration here is as we execute the SQL query, Amazon Athena internally submits a job that operates in asynchronous fashion and the response to the query is saved to Amazon S3. Additionally, we can integrate Amazon Athena with various [BI tools](https://aws.amazon.com/blogs/big-data/creating-dashboards-quickly-on-microsoft-power-bi-using-amazon-athena/) for interactive analysis such as Amazon Quicksight which we’ll discuss later in this chapter.

Like multiple other services, Amazon Athena’s main advantage is that you don’t have to worry about infrastructure management. Let’s discuss a few other features and considerations for using this service for our data analytics use cases:

* We mentioned that Amazon Athena is used to query on data stored in Amazon S3 but what about data present in other AWS data stores such as DynamoDB or on-premise data sources? For these kinds of use cases, Amazon Athena offers a feature called [Federated Query](https://docs.aws.amazon.com/athena/latest/ug/connect-to-a-data-source.html). This feature allows us to run SQL query on a variety of data stores, be it AWS Cloud or on-premise. Amazon Athena achieves this via a Lambda based data source connector—the connector essentially helps to establish a connection and retrieve data.
* In addition to the above point, Amazon Athena can have multiple Lambda invocations in parallel to fastening data processing tasks. It is also possible that data don’t fit into Lambda memory and to handle this, the data is spilled to Amazon S3 and avoids any data loss situations.
* [User Defined Functions](https://docs.aws.amazon.com/athena/latest/ug/querying-udf.html) (UDFs) is another feature of Amazon Athena supported via Federated query SDK. It allows you to write custom Java code on Lambda and invoke directly from SQL queries to perform any pre-processing or post-processing tasks on data. Consider a scenario of sensitive data handling; to avoid storage of sensitive information to S3 as part of Amazon Athena query response, it is masked via logic executed in UDF.
* Amazon Athena also allows us to use deployed Machine Learning (ML) models on [Amazon Sagemaker](https://aws.amazon.com/sagemaker/) (which we’ll discuss later in the chapter). We can directly invoke these [models in SQL query](https://docs.aws.amazon.com/athena/latest/ug/querying-mlmodel.html) for required data analysis—one such example could be detecting negative reviews for our online food ordering platform.
* To help with regular scheduling of workflows or invocation based on certain events, we can leverage Amazon EventBridge. For use cases of workflow orchestration and integration with other services, we can utilize AWS Step Functions.
* The limitation of using Amazon Athena instead of self managed Amazon EMR clusters or Amazon Redshift (which we’ll discuss shortly) could be time taken for query execution for large datasets, so it might not prove to be the best solution for latency sensitive operations.

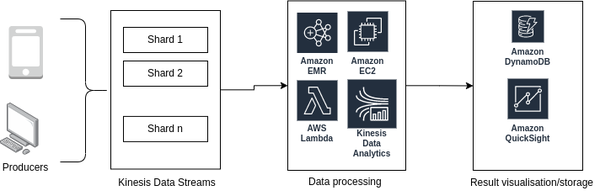
Amazon EMR, AWS Glue and Amazon Athena are powerful services for big data analytics tasks. The discussion we had up until now should help in evaluation of business use cases and how a particular service can be best suited for specific use cases. In the next section, we’ll focus on event streams handling in near real time and run analytics on top of it via Amazon Kinesis.

**Amazon Kinesis**

We started our chapter introduction by taking an example of capturing real time user interactions of users on our online food ordering application. Amazon Kinesis offers capabilities to serve similar kinds of use-cases along with analytics and ETL. There are four sub-services available under the belt of Amazon Kinesis, namely [Kinesis Data Streams](https://aws.amazon.com/kinesis/data-streams/) (KDS), [Kinesis Data Firehose](https://aws.amazon.com/kinesis/data-firehose/) (KDF), [Kinesis Data Analytics](https://aws.amazon.com/kinesis/data-analytics/) (KDA) and [Kinesis Video Streams](https://aws.amazon.com/kinesis/video-streams) (KVS). Let’s start our discussion with Kinesis Data Streams.

**Kinesis Data Streams**

Extending the example of capturing real time user interactions, we’ve multiple devices operated by users referred to as source. This interaction stream is then ingested and stored by Kinesis Data Streams. As the stream is now available in Kinesis, it can be processed based on requirements such as Spark on EMR clusters and finally the results can be dumped to a destination such as DynamoDB or you can analyze the results with any analytical tool as shown in [Figure 9-4](https://learning.oreilly.com/library/view/system-design-on/9781098146887/ch09.html#fig_4_kinesis_data_streams). In short, data streaming on Kinesis enables us to ingest, process and analyze huge volumes of data at large scale without any management overhead on our plate.



**Figure 9-4. Kinesis Data Streams**

A shard is the base throughput unit in KDS and is a uniquely identified sequence of data records in a stream. A shard can ingest up to 1 MB/second supporting up to 1000 TPS and can emit up to 2 MB/second to consumers. We can [configure a number of shards](https://aws.amazon.com/kinesis/data-streams/faqs/#:~:text=The%20throughput%20of%20a%20Kinesis,is%20200%20shards%20per%20stream.) by looking at the supported TPS by a shard and our business use-case production and consumption rate. This configuration is required for Provisioned mode of KDS. We can also explore On-Demand mode where this overhead is handled by AWS itself and it automatically scales as per requirement.

The producers publish data records to data streams which consists of a sequence number, partition key and data blob:

* The sequence number is auto-assigned by Kinesis Data Streams and it is unique per partition key within its shard.
* The partition key is helpful in grouping data at the shard level, meaning to which shard a data record belongs to. The key should be chosen such that it resolves to uniform data distribution across the available shards.
* Data blob is an immutable sequence of bytes which consists of the content or information with a maximum allowed size as 1 MB.

Let’s move forward to discuss key considerations while using KDS:

* To address the limitation of 2 MB/seconds fan-out at a shard level, KDS offers an additional feature referred to as [Enhanced Fan-Out](https://aws.amazon.com/blogs/aws/kds-enhanced-fanout/) (EFO). This can be used to have a dedicated pipe at each consumer level of shard to have 2 MB/second read throughput.
* KDS temporarily stores the data for a time period of 24 hours to 7 days and in the same time, it can also be replayed for any retry strategies.
* We can use [AWS KMS](https://docs.aws.amazon.com/streams/latest/dev/server-side-encryption.html) to encrypt sensitive data as it enters Kinesis Data Streams.
* Kinesis Data Streams provides multiple options on shards such as [update](https://docs.aws.amazon.com/kinesis/latest/APIReference/API_UpdateShardCount.html) number of shards, [split a shard](https://docs.aws.amazon.com/streams/latest/dev/kinesis-using-sdk-java-resharding-split.html) or [merge shards](https://docs.aws.amazon.com/streams/latest/dev/kinesis-using-sdk-java-resharding-merge.html) into one. This helps to ensure full capacity utilization of a shard. The update shard operation can internally merge the split shards to configure the updated number of shards config. Splitting or merging of a shard can be useful if a shard is too “hot” or “cold”, meaning it is receiving too high traffic or too little traffic. If the traffic is too high, it makes sense to split it to balance out the traffic while if the traffic is too low, it makes sense to merge the shards to optimize the number of shards.

We discussed Amazon MSK in Chapter 12 which also provides streaming capabilities similar to Amazon Kinesis Data Streams. As we discussed earlier in the book, the finalization of particular technology depends on the business use-case and no technology is actually better than the other—every tool has their pros and cons. With this in mind, let’s lay out how these two technologies differ from one another in Table 13-2.

|  |  |  |
| --- | --- | --- |
| Parameter | Amazon MSK | Kinesis Data Streams |
| Operations | The base logical entity is partition and we’re required to configure a number of brokers for the cluster launch. The Serverless compute option doesn’t require customers to configure the number and type of brokers and it is fully managed by AWS. | The base logical entity is shard and we’re required to specify the number of shards as per our workload. There is also an option of on-demand capacity which doesn’t require customers to specify shard configuration and is managed by AWS. |
| Latency | The messages are available immediately after it is written to the topic. The latency can be relatively at the lower end as compared to Kinesis Data Streams. | It offers low latency—data to consumers is available to consume within [200 milliseconds](https://docs.aws.amazon.com/streams/latest/dev/kinesis-low-latency.html). With enhanced fan-out the latency is relatively lower than 200 milliseconds. |
| Message Ordering | Messages within a topic partition maintain an order. | Events within a single shard maintain order. |
| Message Delivery | Amazon MSK has support for exactly-once message delivery semantics. This could be helpful for streaming financial transactions with no requirement for explicit handling of duplication in application code. | KDS offers at least-once message delivery semantics. The application should have a [duplicate handling mechanism](https://docs.aws.amazon.com/streams/latest/dev/kinesis-record-processor-duplicates.html) in case the system doesn’t expect duplicate records. |
| Pricing | [Cost](https://aws.amazon.com/msk/pricing/) is based on number and type of broker instances per hour and the amount of storage allocated to the broker. For a serverless option, the cost is based on hourly rate of clusters and each partition being created along with storage required for write and read from the topics. | Customers are [charged](https://aws.amazon.com/kinesis/data-streams/pricing/) by the shard and the number of PUT operations to write data into the stream. No charge for reading data which is less than 7 days old. For on-demand option, the cost is based on per GB data written/read from data streams along with the number of streams. There is additional charge for features like data retention for more than 24 hours and enhanced fan-out |
| Data Retention | Data retention can be configured up to the amount of storage available on the brokers. In case the requirement is to keep data for more than 1 year, Amazon MSK could be the preferred choice though we recommend looking into AWS archival storage solutions if it could be helpful for the business use-case. | Data is accessible by default up to 24 hours and can be increased up to a year. |
| Migration Overhead | Amazon MSK could be a great choice if you’re already managing the Kafka clusters on your own on Amazon EC2 instances or on an on-premise data center. This helps in reduction of any migration workload to a different setup for streaming. | People with no familiarity with any of the services should look for other parameters to decide on a service to be used to meet business requirements. |
| Integration with AWS services | Both KDS and Amazon MSK offer easy integration with other AWS services such as EC2, Kinesis Data analytics, EMR etc. |  |
| Table 9-2. Amazon MSK and Kinesis Data Streams comparison | | |

The above table should be helpful in devising some insights around which solution could be better for any future use-cases. In the last chapter, we also discussed SNS and SQS in relation to the publisher-subscriber design model. If you think about it, KDS offers a similar kind of model where data is published to streams and multiple consumers can consume the data from a stream. The big difference is near real time processing being offered by KDS or Amazon MSK streaming solution. In the case of SNS and SQS model, the data should be ingested first to any specific system for analysis but KDS allows for data analysis as we receive the data. Please refer to Chapter 12 for complete understanding of additional features offered by SQS and SNS. Now let’s move to another Kinesis offering, Kinesis Data Analytics (KDA) which is useful for analytics purposes in near real time as data is processed via Kinesis Data Streams or Amazon MSK.

**Kinesis Data Analytics**

Kinesis Data Analytics is a fully managed service for [Apache Flink](https://flink.apache.org/what-is-flink/flink-architecture/) capabilities that puts no overhead on the customer’s plate and scales automatically for the desired throughput of incoming data. Kinesis Data Analytics helps in processing, querying and analyzing streaming data in near real time and sends it to the configured data destinations such as Kinesis Data Streams, Amazon MSK, S3, and Amazon Opensearch. Let’s try to get some understanding of Apache Flink before we discuss more specific to KDA.

* Apache Flink is an open-source distributed processing engine framework helpful in processing real time data streams.
* It supports both bounded and unbounded streams processing. Bounded streams refers to batch processing while unbounded refers to processing of streaming data in real time.
* It performs stateful computations in-memory ensuring low latency. The stateful nature ensures exactly-once processing of streaming data events. Apache Flink ensures the state is maintained even if the system goes down by asynchronous checkpointing the state to durable storage.
* It supports [SQL, Table API](https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/table/overview/), [DataStream APIs](https://nightlies.apache.org/flink/flink-docs-release-1.17/docs/dev/datastream/overview/) and [stateful functions](https://nightlies.apache.org/flink/flink-statefun-docs-stable/" \t "_blank) for easy access of data. The SQL and Table APIs have the most abstraction while datastreams APIs and stateful functions give more granular control and visibility to customers. It has language support for Java, Scala, Python and SQL.

The deployments and infrastructure management can be a huge pain point for customers and also requires expertise around Apache Flink core concepts. Amazon KDA is a serverless offering, allowing customers to focus on business logic while AWS manages everything end to end. Here are details around some of the features specific to KDA:

* As KDA is available in the AWS environment, it offers easy integration with other AWS services such as Kinesis Data Streams, Amazon MSK, Amazon S3, Amazon Opensearch, Amazon Cloudwatch, Amazon DynamoDB, Kinesis Data Firehose and AWS Glue Schema Registry.
* KDA can also be easily integrated with open source tools via [Apache Flink connectors](https://nightlies.apache.org/flink/flink-docs-release-1.15/docs/connectors/datastream/overview/) and also offers support to build custom connectors for use-cases which are not supported as of now.
* The integration with AWS Glue Data catalog helps in storing and sharing of metadata across multiple applications.
* KDA offers [KDA Studio](https://aws.amazon.com/blogs/aws/introducing-amazon-kinesis-data-analytics-studio-quickly-interact-with-streaming-data-using-sql-python-or-scala/) which is basically a wrapper around Apache Flink SQL and Table APIs to analyze the streaming data in interactive way via [Apache Zeppelin](https://zeppelin.apache.org/) serverless notebooks. KDA Studio supports SQL, Scala and Python language.

**NOTE**

A notebook is a web-based developer interactive environment running in the browser of your choice. The developers can write queries or code in supported languages for different use-cases such as data transformation, visualization and analysis.

Primarily, Kinesis Data Analytics is a fully managed version of Apache Flink and since it is part of AWS Cloud offerings, we can easily integrate with other AWS services. Now let’s discuss one more use case where we just have a requirement to load live streaming data to destinations. This requirement is met via Kinesis Data Firehose so let’s divert our attention to this service now.

**Kinesis Data Firehose**

Kinesis Data Firehose (KDF) is data delivery service for live streaming data to different destinations such as Amazon S3, Amazon Opensearch, [Amazon Redshift](https://aws.amazon.com/redshift/), [Splunk](https://docs.aws.amazon.com/firehose/latest/dev/creating-the-stream-to-splunk.html" \t "_blank) or any custom HTTP/HTTPS endpoints to easily integrate with third-party data storage providers. KDF is a fully managed service and customers don’t have to worry about infrastructure and service maintenance. KDF doesn’t offer any storage of its own so we don’t have the feature of replaying messages like we do with Kinesis Data Streams. We can additionally configure KDF to transform data before delivering it. There are some built-in transformations provided by KDF, such as conversion to Apache Parquet format. We can also add our custom transformations using AWS Lambda functions.

Here are few considerations related to KDF:

* The data record sent by the producer to the KDF delivery stream should not exceed 1000 KB.
* To enhance security and reduce storage space at destination data storage solutions, we can batch, compress and encrypt the data before loading.
* KDF buffers incoming streaming data to a certain size (buffer size) or certain period of time (buffer interval) before being delivered to data destinations. The buffer size could range from 1 to 128 MB for Amazon S3, 1 to 100 MB for Amazon Opensearch and 0.2 MB to 3 MB for AWS Lambda. The buffer interval for all these service integrations can range from 60 to 900 seconds.
* When Amazon Redshift is configured as a data destination, KDF first delivers data to the Amazon S3 bucket and then issues [COPY command](https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html) to load data from S3 bucket to Redshift cluster.
* When Amazon S3 is selected as a delivery destination, we can also enable dynamic partitioning of data. This helps at two fronts: it provides easy access of data in S3 for querying purposes and removes the need of partitioning at source or after the data is stored. We can specify a specific key or an expression evaluated at runtime to identify keys to be used for partitioning.
* There is a limitation on KDF when it comes to data delivery to multiple destinations. A single delivery stream can deliver data only to a single Amazon Redshift cluster or table, single Amazon S3 bucket, and single Amazon Opensearch cluster or index currently. If there is a requirement to deliver at multiple destinations, a separate delivery stream should be created.
* KDF supports at-least once data delivery semantics. There is a possibility that data is duplicated at the delivery destination due to retry mechanisms put in place for failure handling.

In short, KDF is a streaming ETL service fully managed by AWS for the customers. Now let’s move to another Kinesis offering, Kinesis Video Streams service.

**Kinesis Video Streams**

Kinesis Video Streams (KVS) are optimized for delivering live streaming video data to AWS in near real time from millions of sources for any processing, such as running Machine Learning algorithms or applying any custom video processing. KVS can be used for offline batch analytics as well, and supports other data streams such as audio data, images, RADAR data, etc.

KVS can be useful in a lot of [scenarios](https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/examples.html), let’s take an example of how KVS can be useful in building a household security system. People are not always at home and it could be really helpful if they can get a real time continuous feed of what’s happening at their homes. To build this, a security company installs the cameras at home in desired locations. These cameras act as a source of information and have KVS Producer libraries installed to connect with KVS on AWS Cloud securely. The connection helps in constantly streaming the video feed to KVS where we can optionally store it or redirect the data to consumers. The consumers are applications that consume this data such as you accessing the feed on your mobile device. This could really be helpful in cases such as robbery or alarming you if there are unannounced people at home, and all this can be achieved in real time.

A few key considerations about KVS include:

* KVS provides durable data storage for configurable retention periods. It automatically encrypts the data at rest & in-transit and creates an index over this data based on producer-generated or service-side timestamps for easy & quick access.
* KVS is a fully managed Serverless offering so we as customers don’t have to worry about infrastructure management or any expertise to tune any kind of configurations.
* KVS provides a parser[library](https://docs.aws.amazon.com/kinesisvideostreams/latest/dg/parser-library.html) that helps consumer applications to get data from video streams in very low latency. It can be used in Java applications for MKV video streams.

We recommend analyzing your business requirements and then choosing the specific Kinesis offering which best suits your needs. Continuing on our journey to analytics tools at huge scale, now let’s discuss the AWS powered business intelligence tool, Amazon Quicksight.

**Amazon QuickSight**

Looking back at our online food ordering application, it stores data in multiple data stores such as total orders in a particular region, top rated restaurants, most ordered food items, etc. How can we create a single unified dashboard connecting to different datastores without providing direct access to datastores to everyone? This can easily be achieved by using Amazon Quicksight. Amazon Quicksight is a fully managed serverless business intelligence service offering analytics, visualization and reporting. It can connect to different kinds of datastores and include it in a single dashboard, the data can include AWS data, third-party data, spreadsheet data, SaaS data, and more.

Here are key features and considerations about Amazon Quicksight:

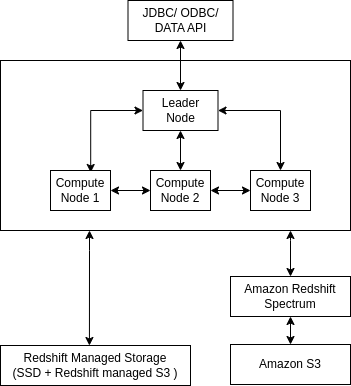
* Amazon Quicksight essentially enables every user to perform analytics and visualize the data for your use case with very less expertise. With Amazon Quicksight, you don’t need to be dependent on business intelligence teams for your data requests.
* Amazon Quicksight is offered in two variants, [Standard and Enterprise edition](https://docs.aws.amazon.com/quicksight/latest/user/editions.html). The Standard edition contains all the features described above. The Enterprise edition includes more advanced capabilities such as automated and customizable data insights powered by Machine Learning, security features including federated user identities, single sign-on, encryption at rest, etc. The Enterprise edition also helps to monetize the dashboards via pay-per-session pricing model. The consumers of the dashboard you’ve created pay for it per their usage.
* As part of Enterprise edition, there is an additional feature available referred to as [Amazon Quicksight Q](https://docs.aws.amazon.com/quicksight/latest/user/quicksight-q-get-started.html). It is a natural language processing tool where instead of writing queries, you can directly ask questions such as “top 5 restaurants in Mumbai”.
* Amazon Quicksight supports fast advanced calculations to serve data via SPICE (Super-fast, Parallel, In-memory Calculation Engine). To make use of this in-memory data store, it should be enabled as part of the database creation or editing process and customers need to configure required [SPICE capacity](https://docs.aws.amazon.com/quicksight/latest/user/managing-spice-capacity.html).

We mentioned that Amazon Quicksight is helpful to draw insights and it can do it for any data source. One such data source is Amazon Redshift which we’ll discuss in the following section.

**Amazon Redshift**

In simple terms, Amazon Redshift is a data warehousing tool which can act as a data store for data from multiple sources and allows us to run SQL queries at a single place for data analytics. Amazon Redshift is an AWS managed service and can scale to petabytes of data with an [elastic scaling](https://docs.aws.amazon.com/redshift/latest/mgmt/managing-cluster-operations.html) option. There is also a [Serverless](https://docs.aws.amazon.com/redshift/latest/mgmt/serverless-whatis.html" \t "_blank) option, which helps to avoid any burden of resource provisioning and workload management.

We’ll start with discussing the architecture of Amazon Redshift and then move towards key features offered by this service. Amazon Redshift is columnar PSQL based storage and the is based on leader follower cluster architecture where leader acts as a query coordinator who parses and complies the query and forwards the query to follower (compute) nodes, which works in parallel to gather results of the query as shown in [Figure 9-5](https://learning.oreilly.com/library/view/system-design-on/9781098146887/ch09.html#fig_5_amazon_redshift_architecture). There can be 2 to 128 nodes in an Amazon Redshift cluster and in the case of the Serverless option, this entire architecture is abstracted out from customers. The query can be invoked from SQL clients via [JDBC](https://docs.aws.amazon.com/redshift/latest/mgmt/jdbc20-install.html)/[ODBC](https://docs.aws.amazon.com/redshift/latest/mgmt/configure-odbc-connection.html) connection or use Data API. The [Data API](https://github.com/aws-samples/getting-started-with-amazon-redshift-data-api) offers query execution in both synchronous and asynchronous fashion and can query for results later within 24 hours with a query Id.



**Figure 9-5. Amazon Redshift architecture**

Here are some additional details to consider about the Amazon Redshift architecture:

* The compute nodes are partitioned into slices and each slice has its own allocated disk and memory to perform parallel processing.
* The data is stored into immutable [blocks](https://docs.aws.amazon.com/redshift/latest/dg/c_columnar_storage_disk_mem_mgmnt.html); A block consists of column data spanning across the multiple rows and a full block can contain millions of values with size of 1 MB.
* To improve query performance over the data blocks, Amazon Redshift maintains in-memory metadata information referred to as [Zone Maps](https://aws.amazon.com/blogs/big-data/amazon-redshift-engineerings-advanced-table-design-playbook-compound-and-interleaved-sort-keys/). It stores the minimum and maximum values for a block and helps in effectively pruning the data blocks that don’t contain data for specific query. The Zone Maps can be optimized by using a [sort key](https://docs.aws.amazon.com/redshift/latest/dg/t_Sorting_data.html) which defines how data is sorted on the physical disk. The sort key works well except for very few scenarios such as: when there is only one block per column per slice, the values within blocks have the same prefix (for strings longer than 8 bytes, Amazon Redshift takes the first 8 bytes as prefix so sorting won’t matter if it’s same), and when the column contains single distinct value (sorting won’t matter as minimum and maximum values are consistent).
* To optimize the query performance, we can optionally choose to provide a [distribution style](https://docs.aws.amazon.com/redshift/latest/dg/c_choosing_dist_sort.html) for data. This ensures data is evenly distributed across the cluster and we can make best out of parallel processing. There are four distribution styles supported by Amazon Redshift; AUTO (default option), EVEN, KEY and ALL.
  + For the AUTO option, Amazon Redshift choses the distribution style based on size of the table data and Amazon Redshift internally switches between the different distribution styles based on the size.
  + For the EVEN option, data is distributed across the compute node slices in round robin fashion by the leader node. This is a recommended option when there are no join operations on the table.
  + For the KEY option, the leader node checks the column value and places it on the compute node slice matching its value.
  + For the ALL option, the entire table data is available on each node. This is only recommended if there are very few insert or update operations on the table.
* Amazon Redshift offers Redshift Spectrum which can use Amazon S3 as storage and the compute nodes are only responsible for gathering the results. This helps customers to directly query the data from Amazon S3 without loading it to Amazon Redshift clusters. One key consideration here is repeated reads against Amazon S3 as storage doesn’t adhere to transactional guarantees.
* Amazon Redshift also offers Redshift Managed Storage (RMS) allowing customers to scale compute and storage independently. It uses S3 as persistent storage and offers high speed SSD backed Tier-1 cache support. RMS allows scaling up to 128 TB per instance and up to 16 PB per cluster storage capacity. For scaling further, we can always utilize Amazon S3 as storage.
* We can choose from [different instance types](https://docs.aws.amazon.com/redshift/latest/mgmt/working-with-clusters.html#rs-about-clusters-and-nodes) available for our Amazon Redshift cluster. The feature is supported by the new Amazon Redshift instance type–[RA3](https://aws.amazon.com/redshift/features/ra3/). The other instance types are DC2 (Dense Compute) with SSD as storage and DS2 (Dense Storage) with magnetic disks as storage. The DS2 instance types are legacy and we don’t recommend using them for your workloads.

Amazon Redshift Spectrum is kind of similar to Amazon Athena where we query on top of the data that sits in Amazon S3. The one big difference is we can configure compute, allowing the queries to be much faster. There are definitely additional feature sets of both of these services which we should evaluate before choosing one for our workloads:

* Amazon Redshift allows us to create [Materialized views](https://docs.aws.amazon.com/redshift/latest/dg/materialized-view-overview.html) from one or more tables to make the queries faster. The Materialized views contain pre-compute result sets based on table joins with all or subset of columns, aggregations and filters. We can also create Materialized views on top of already created Materialized views similar to the base tables.
* Amazon Redshift offers [workload management tools](https://docs.aws.amazon.com/redshift/latest/dg/c_workload_mngmt_classification.html) (WLM) for separation of different query workloads and assigning priority so that important queries are executed and less important queries can be throttled or aborted.
* We can use [COPY](https://docs.aws.amazon.com/redshift/latest/dg/r_COPY.html) command to load data to Amazon Redshift from Amazon S3, Amazon DynamoDB, Amazon EMR or any remote host accessible via Secure Shell Connection (SSH). There is also support available for [auto-copy](https://aws.amazon.com/blogs/big-data/simplify-data-ingestion-from-amazon-s3-to-amazon-redshift-using-auto-copy-preview/) which automatically ingests data from Amazon S3 to Amazon Redshift clusters.
* We can integrate Amazon Redshift with Amazon MSK or Kinesis Data Streams for ingesting live streaming data. We can use [Informatica Data Loader for Amazon Redshift](https://aws.amazon.com/blogs/big-data/simplify-data-loading-on-the-amazon-redshift-console-with-informatica-data-loader/" \t "_blank) to access data from on-premise or third party applications.
* We can use Amazon Redshift [Federated Query](https://docs.aws.amazon.com/redshift/latest/dg/federated-overview.html) feature to access live data from external data stores without loading it to Amazon Redshift clusters such as Amazon RDS PostgreSQL & MySQL and Amazon Aurora PostgreSQL & MySQL.
* Amazon Redshift also allows [data sharing](https://docs.aws.amazon.com/redshift/latest/dg/datashare-overview.html) across the clusters. This helps in isolating the read workloads without copying data on multiple clusters.
* Amazon Redshift provides [direct integration](https://docs.aws.amazon.com/redshift/latest/mgmt/zero-etl-using.html) with Amazon Aurora so as to avoid any pipeline set up by customers and the data is directly available in Amazon Redshift once pushed to Amazon Aurora within seconds.
* [Amazon Redshift Advisor](https://docs.aws.amazon.com/redshift/latest/dg/advisor.html) is a feature that provides recommendations for cluster optimization around query performance and cost savings. For example, query tuning recommendations, deletion of unused clusters, and table data compression.

In summary, Amazon Redshift is a great tool to use for data warehousing and analytics use cases scaling to petabytes of data. In the next section, we’ll look into different Machine Learning services offered by AWS and how they can fit into different business use cases.

**Machine Learning on AWS**

To support Machine Learning (ML) and Artificial Intelligence (AI) use cases, AWS offers a range[of services](https://docs.aws.amazon.com/whitepapers/latest/aws-overview/machine-learning.html) from building and running our own ML models to leveraging fully customized applications for specific purposes. For example; [Amazon Polly](https://aws.amazon.com/polly/) helps in converting text to speech, [Amazon Comprehend](https://aws.amazon.com/comprehend/) helps in figuring out insights and relationships in unstructured data, [Amazon CodeGuru](https://aws.amazon.com/codeguru/) offers intelligent recommendations to improve code quality, etc. The services can be divided into three broad categories: application services, platform services and frameworks & hardware solutions.

Application services are customized solutions for solving a specific use case. Platform services allow us to create our own services such as [Amazon Sagemaker](https://aws.amazon.com/sagemaker/) and [AWS DeepLens](https://aws.amazon.com/deeplens/) by training and deploying ML models. AWS offers different frameworks such as [PyTorch](https://pytorch.org/" \t "_blank), [Keras](https://keras.io/" \t "_blank), [Tensorflow](https://www.tensorflow.org/" \t "_blank), etc and special hardware with customized [CPU](https://docs.aws.amazon.com/dlami/latest/devguide/cpu.html) and [GPU](https://docs.aws.amazon.com/dlami/latest/devguide/gpu.html) to run ML workloads. We’ll start off our discussion with Amazon Sagemaker and then briefly touch upon other offerings.

**NOTE**

Intelligence means information—our ability to ask questions and provide reasons for the answers associated with these questions. We call it AI when this intelligence is shown by machines. ML is a branch of AI representing this intelligence based on previous data and algorithms. This book will not introduce you to ML concepts but only a few of the AWS services offering ML capabilities.

**Amazon SageMaker**

The default benefit we achieve by choosing AWS Cloud is running any service without the worry of infrastructure management. Amazon Sagemaker is an AWS managed service which allows customers to prepare data, build, train and deploy ML models quickly without the worry of managing any infrastructure. Amazon Sagemaker offers features for each of these steps in finally deploying your models with ease and getting the best out of it with minimal effort:

*Data Preparation*

Amazon Sagemaker offers tools for initial data preparation that allow you to build on top of it, such as [Data Wrangler](https://docs.aws.amazon.com/sagemaker/latest/dg/data-wrangler.html). Data Wrangler can be added in ML workflows in [Amazon Sagemaker Studio](https://docs.aws.amazon.com/sagemaker/latest/dg/studio.html) to import, prepare, transform, identify features and analyze data.

*Build*

Customers can use Jupyter notebooks inside Amazon Sagemaker Studio to build their ML models. AWS offers its own custom built algorithms that run efficiently on AWS infrastructure as well as popular open-source frameworks such as TensorFlow, PyTorch, etc. We can also write our custom code if the already available solutions don’t fit our use case.

*Train*

As Amazon Sagemaker manages all the infrastructure, we don’t have to worry about training. We can use all the managed infrastructure from storage to compute.

*Deploy*

Deployment of ML models is also taken care of by Amazon Sagemaker itself. There are multiple ways to deploy a model and a particular way can be selected based on the use case. We recommend using [real-time inference](https://docs.aws.amazon.com/sagemaker/latest/dg/realtime-endpoints.html) for workloads with low latency requirements, [serverless inference](https://docs.aws.amazon.com/sagemaker/latest/dg/serverless-endpoints.html" \t "_blank) as a fully managed solution and for workloads that can tolerate cold start problems, [asynchronous inference](https://docs.aws.amazon.com/sagemaker/latest/dg/async-inference.html) for workloads with large payload sizes (up to 1GB) and near real time processing requirements and [batch transforms](https://docs.aws.amazon.com/sagemaker/latest/dg/batch-transform.html) for processing entire datasets.

Amazon Sagemaker also offers support for A/B testing by enabling multiple ML models behind a single endpoint, A/B testing helps in figuring out how different models are performing and if they are working as per expectations.

Depending on our use case, we can decide to use Amazon Sagemaker for some or all of the above features in building and deploying our entire pipelines. For example, we can bring our already trained model on on-premise infrastructure and deploy it on Amazon Sagemaker.

Next we’ll cover some other key features offered by Amazon Sagemaker:

* Amazon Sagemaker includes multiple features which help improve our ML models, such as [Amazon Sagemaker Clarify](https://docs.aws.amazon.com/sagemaker/latest/dg/clarify-fairness-and-explainability.html). As the name dictates, it detects any potential bias and helps clarify the predictions that ML models make using a feature attribution approach.
* Metrics are an important tool for visualizing performance. To visualize a ML model’s performance, we can use [Amazon Sagemaker Debugger](https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-on-studio.html). It additionally helps to identify system bottlenecks for EC2 instance jobs with metrics such as CPU, GPU, GPU memory, network and Data I/O.
* We mentioned that Amazon Sagemaker helps with the entire cycle from data preparation to model deployments. As we scale our systems, it’s very important that it is automatically managed to reduce operational overhead on our plate. [Amazon Sagemaker Pipelines](https://docs.aws.amazon.com/sagemaker/latest/dg/pipelines.html) helps with this automation and building end-to-end CI/CD pipelines.
* The ML model performance can vary based on hyperparameter values. Amazon Sagemaker Automatic Model Tuning ([AMT](https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html)) runs the training job multiple times and figures out the best version of the model.
* It’s important to train ML models on good quality datasets to get the most out of them. [Amazon Sagemaker Ground Truth](https://docs.aws.amazon.com/sagemaker/latest/dg/sms.html) helps with automation of this process by creating high quality labeled datasets. For data labeling tasks, we can choose the workforce of our choice: from independent contractors, to our own private workforce, to vendor companies on AWS marketplace.

Amazon Sagemaker is a great tool for building ML solutions on AWS Cloud. We recommend going through the [AWS documentation](https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html) for an even deeper dive into its capabilities, as we couldn’t possibly fit everything into this chapter.

There may be use cases where we don’t have expertise to build our own solutions, in these scenarios we can leverage different services offered by AWS built to solve specific use cases. Let’s explore some of these services in the next section.

**AWS ML Application Services**

Building ML solutions on our own takes time as well as expertise in a field where we might not want to invest. If you have a general use case for ML, you can use fully managed services by AWS without the worry of building, deploying and maintaining ML models. Some of these services include:

*Amazon CodeWhisperer*

You might be familiar with the concept of pair programming where two people have a discussion and then code together. [Amazon CodeWhisperer](https://aws.amazon.com/codewhisperer/) is your coding companion and helps you with code by generating code suggestions from small code snippets to entire functions. It also helps to flag the security issues in code and offers suggestions to remediate the raised concerns and can be easily integrated in your favorite code editor.

*Amazon Comprehend*

[Amazon Comprehend](https://aws.amazon.com/comprehend/) helps in gathering insights from a document or a collection of documents. Taking our example of an online food ordering application, customers write reviews on the ordered food and we can use Amazon Comprehend for figuring review behavior (positive/negative) or identifying food items being talked about. Amazon Comprehend can identify insights such as entities (person names, places, items, etc), PII information, document language, sentiment (positive, negative, neutral or mixed), and more.

*Amazon Kendra*

[Amazon Kendra](https://aws.amazon.com/pm/kendra/) is a ML powered search engine on top of structured or unstructured data repositories. It uses Natural Language Processing to determine a document most relevant to user’s search queries. We use Amazon Kendra along with [Amazon Lex](https://aws.amazon.com/pm/lex/) to build [AI chatbots](https://aws.amazon.com/blogs/machine-learning/building-ai-chatbots-using-amazon-lex-and-amazon-kendra-for-filtering-query-results-based-on-user-context/) for use cases such as customer support to resolve user queries on our online food ordering application.

*Amazon Forecast*

[Amazon Forecast](https://aws.amazon.com/forecast/) is helpful for accurate time-series forecasts. Consider an example of an online business: how do you predict traffic for an upcoming sale based on historical traffic or what growth looks like if you launch in a new region. Amazon Forecast automatically figures out combinations of ML algorithms suitable for your dataset and helps you with the forecasts.

*Amazon Rekognition*

[Amazon Rekognition](https://aws.amazon.com/rekognition/) is a recognition service helpful in image and video analysis. This service can power use cases such as search over image and video content, face identification and verification, adult content detection, text extraction from images, etc.

*Amazon Transcribe*

[Amazon Transcribe](https://aws.amazon.com/transcribe/) is a speech to text conversion service. Consider an example where you offer a service where customers can submit their feedback via telephone for food items ordered on an online food ordering application. Amazon Transcribe can help derive valuable [insights](https://aws.amazon.com/transcribe/call-analytics/) from such calls. Some other use cases where it can help is taking voice input and converting it to text for processing in your systems, converting audio and video content to be searchable by text, subtitles for any videos, etc.

These customized offerings help in direct integrations within our applications with minimal effort. To take an example, assume that you own a blog hosting service where people can manage their blogs and share material with their followers. Now to add a functionality of text to audio, you can either build your own solution or you can use Amazon Polly for a much faster start. As we’ve mentioned before, we can always improve on the solutions in the future if our use case changes or we see any bottlenecks in existing solutions. The key point is time to market—if you start with a custom solution it might take much more time to launch the service as compared to direct integration with Amazon Polly. In the next section, we’ll discuss special infrastructure support provided by AWS to build our ML workloads.

**AWS ML Infrastructure**

AWS offers EC2 instances specialized to handle ML workloads both for training and inference. These instances include hardware based accelerators, also referred to as co-processors to enhance the computing power and perform tasks such as graphics processing, floating point number calculations, and data pattern matching in a much more efficient manner as compared to software running on general purpose CPUs. Some of the examples of accelerators are Graphical Processing Units ([GPUs](https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/)), Field Programmable Gate Arrays ([FPGAs](https://en.wikipedia.org/wiki/Field-programmable_gate_array)), AWS [Inferentia](https://aws.amazon.com/machine-learning/inferentia/" \t "_blank), and AWS [Trainium](https://aws.amazon.com/machine-learning/trainium/" \t "_blank). Let’s talk about the custom solutions offered by AWS to accelerate our ML workloads:

*AWS Trainium*

The Amazon EC2 [Trainium](https://aws.amazon.com/machine-learning/trainium/" \t "_blank) instances offer up to 50% cost-to-train savings over comparable Amazon EC2 instances. These instances are optimized to run [Deep Learning](https://aws.amazon.com/deep-learning/) (DL) training workloads with native support of different data types such as FP32, TF32, BF16, FP16, UINT8, and configurable FP8. It supports AWS Neuron SDK which is natively integrated with [PyTorch](https://awsdocs-neuron.readthedocs-hosted.com/en/latest/frameworks/torch/index.html" \l "pytorch-neuron-training-tutorials" \t "_blank) and [Tensorflow](https://awsdocs-neuron.readthedocs-hosted.com/en/latest/frameworks/tensorflow/index.html" \t "_blank) so as existing framework applications can be used with minimal code changes.

*AWS Inferentia*

The Amazon EC2 [Inferentia](https://aws.amazon.com/machine-learning/inferentia/" \t "_blank) instances are designed to run ML inference applications with high throughput and low latency while saving costs as compared to general purpose Amazon EC2 instances. These types of instances offer high speed connectivity between the accelerators, enabling us to deploy billions of parameters across multiple accelerators on Amazon Inferentia EC2 instances.

**NOTE**

Inference means reaching a conclusion based on logical reasoning and evidence, so ML inference essentially means running ML models on live data to get a final answer (or make a prediction). The training is a pre-phase to inference where we train ML applications to learn from existing data and then use this intelligence at time of inference.

The EC2 instances are purpose built to optimize compute heavy tasks such as running large scale models, natural language processing, speech recognition, and computer vision. You might consider the cost there— if the accelerator in these EC2 instances makes processing so fast then it would definitely be costly and you should consider if you can afford such an expenditure. As these instances are specifically designed for ML workloads, they are cost effective if you compare them to EC2 instances with similar capabilities. For example, [Amazon EC2 Inf1](https://aws.amazon.com/ec2/instance-types/inf1/) instances provide 2.3x higher throughput and up to 70% lower cost per inference than comparable EC2 instances. The [Amazon EC2 Inf2 instances](https://aws.amazon.com/ec2/instance-types/inf2/) are cheaper than Inf1, 4x higher throughput and 10x lower latency. A very good thing about using AWS Cloud is that AWS innovates on the behalf of customers. We as customers might be happy with the performance and cost benefits of Amazon EC2 Inf1 instances, but AWS launched a new version with more improvement to existing options.

**Conclusion**

This chapter introduced you to AWS services that are helpful in building and running Big Data analytics and Machine Learning workloads at any scale of your business use case. You’ll notice that some services have similar features— such as when we compare Amazon MSK to SQS-SNS or to Kinesis Data Streams. Make sure you always have your requirements in hand when comparing services to make sure you’re making the right selection. Your use case requirements are of utmost importance when selecting a service.

Many times you may want to use a combination of multiple services to serve your use case best, such as Amazon EMR to process the live streaming data published to Amazon MSK or have Amazon Redshift data plotted via Amazon Quicksight. As AWS provides easy integration with other services, it is a seamless experience to use any service and fit it into other service components to meet our requirements.

In the second part of this chapter, we looked into AWS ML offerings which essentially help to run our ML workloads with zero or minimal operational overhead. There are multiple services offered by AWS to solve specific use cases, and these offerings help us to utilize the ML capabilities without having vast knowledge of machine learning concepts. For example, think of building a language translation tool on your own vs directly using [Amazon Translate](https://aws.amazon.com/translate/). Amazon Translate will reduce the time-to-market to launch your own application with translation capability.

We concluded our chapter with discussion around multiple infrastructure options that are specialized to run ML workloads at scale with higher throughput and lower cost as compared to general purpose comparable hardware.This is the final chapter in this section of the book and we’re confident that now we can use our understanding of the material covered in Parts 1 and 2 to build real time large scale systems. In Part 3, we’ll walk through use cases for building systems on AWS Cloud, starting with a URL shortening service.