Kinesis

# Index

Table of Contents

[Index 2](#_Toc28975260)

[Kinesis Data Streams 3](#_Toc28975261)

[Definition 3](#_Toc28975262)

[Kinesis Shards 3](#_Toc28975263)

[Kinesis Records 3](#_Toc28975264)

[Kinesis Producers 4](#_Toc28975265)

[Consumer Models 5](#_Toc28975266)

[Miscellaneous Features 5](#_Toc28975267)

[Kinesis Data Firehose 6](#_Toc28975268)

[Definition 7](#_Toc28975269)

[Key Terms 7](#_Toc28975270)

[Data Transformation 7](#_Toc28975271)

[Cost Model 8](#_Toc28975272)

[Kinesis Analytics 9](#_Toc28975273)

[Definition 9](#_Toc28975274)

[Benefits 9](#_Toc28975275)

# Kinesis Data Streams

## Definition

* It is a platform to send streaming data to.
* It makes it easy to load and analyze streaming data i.e. run real-time data analytics & reporting, while also providing the ability for one to build one’s own custom applications as per business requirements.
* One can create complex stream processing wherein one or more Kinesis data streams feed data into another data stream.
* Streaming data is data generated continuously by thousands of data sources, which typically send the data simultaneously, and in small sizes (order of Kilobytes).
  + Purchase from online stores
  + Stock prices
  + Game data
  + Social network data
  + Geospatial data
  + IOT sensor data

## Kinesis Shards

* Shards are the building blocks of a Kinesis Data Stream & determine its capacity.
* Each shard spreads over multiple AZs.
* Each shard is allowed a maximum of:
  + 1 MB per second in
  + 1000 data records per second in
  + 2 MB per second out
  + 5 transactions per second out (1 transaction contains 1 or more records)
* Shards are designed to support multiple consumers with high read consumption per shard
* The process of scaling throughput up or down by adding or removing shards is called resharding.
  + There are two separate options for resharding:
    - Merging a shard: Combining data of two shards into a single shard.
    - Splitting a shard: Creating a new shard and splitting the data between the new and existing shards. The old shard will be closed once it’s data expires.
  + Resharding is an expensive process as although it takes less than a minute, it can not be done in parallel. (i.e. going from a single shard to 40 shards can take ~40 minutes). Hence, it is important to get the initial shards provisioning as accurate as possible.

## Kinesis Records

* Records are a unit of data in the Kinesis Data Stream.
* Each record contains a:
  + Sequence number
    - Numbers given to each data record that are unique within that data record’s shard.
    - Assigned to the data record when it is written to the stream.
    - It is used to order the records per shard.
  + Partition Key
    - Required when a producer puts data into a stream.
    - Used to group data by shard within a stream i.e. records with the same partition key go to the same shard.
    - Generally randomized values of Unicode strings with a limit of 256 bytes hashed with MD5 to map the keys to shards.
    - Able to keep related data together on the same shard. (This can lead to performance issues wherein skewed data create ‘hot’ shards).
  + Data Blob
    - Are an immutable sequence of bytes
    - Can be up to 1MB.
    - Are not inspected, interpreted, or changed by Kinesis Data Streams in any way.
* The record retention period defaults to 24 hours and can be configured to be retained for up to 168 hours (7 days).

## Kinesis Producers

* Kinesis Agent
  + A Java-based client built on top of KPL that streams and tails files to Kinesis Data Streams.
  + The client can:
    - Stream/ tail files from multiple directories to multiple streams.
    - Handle file rotation/ checkpointing/ retries on failure
    - Preprocess data before sending it. For example, convert multi-line records to one line, CSV to JSON, LOG to JSON, etc.
    - Send metrics to CloudWatch such as *BytesSent*, *RecordSendAttempts*, etc.
* Kinesis Producer Library (KPL)
  + An easy to use Java/C++ library one can include as part of custom applications to stream data to Kinesis Data Streams or Data Firehose.
  + It’s best suited for long running, high-rate producers or those that need record aggregation.
  + The producer can:
    - Write to multiple Kinesis streams.
    - Handle errors with built-in retry mechanisms.
    - Do both synchronous as well as asynchronous writing of data.
    - Support multithreading.
    - Integrate with CloudWatch to show metrics such as Records In, Records Out, etc.
  + The KPL helps improve performance and reduce network request overhead in two ways:
    - Aggregation: Multiple smaller pieces of information /records can be merged into a single record to avoid the 1MB or 1000 record per second per shard limit (reduces the number of records).
    - Collection: Using ‘*PutRecords*’ API, multiple records can all be aggregated into a single HTTP request to the Kinesis API to save on request overhead (reduces the number of HTTP requests).
  + KPL records must be de-coded with KCL or special helper library.
* AWS SDKs
  + A set of tools that can be used to send data to Kinesis Data Streams or Data Firehose.
    - Uses ‘*PutRecord(s)*’ API, which in turn uses batching and increases throughput while simultaneously making fewer HTTP requests.
  + One can build their own producers for Kinesis.
  + Usually preferred for low-rate producers such as mobile apps, IoT devices, web clients, etc. where low throughput & high latency is acceptable.

## Kinesis Consumers

* Kinesis SDK
  + Records are polled by consumers from a shard using the ‘*GetRecords*’ API.
  + Each shard has a total of 2 MB aggregate throughput.
  + Each ‘*GetRecords*’ call returns up to 10 MB of data, or 10,000 records. Then it throttles for 5 seconds.
  + There is a maximum of 5 ‘*GetRecords’* API calls per second = 200ms latency.
  + If 5 consumer applications consume from the same shard, then that means each consumer can poll once a second and receive less than 400 KB/s.
* Kinesis Client Library (KCL)
  + Java-first library, but exists for other languages such as Golang, Python, Ruby, etc.
  + Used to read records from Kinesis produced by KPL using de-aggregation.
  + One can shard multiple shards with multiple consumers in one ‘group’, also known as shard discovery.
  + It has a checkpointing feature to resume progress.
    - It leverages a DynamoDB table for checkpointing purposes. (One row per shard)
    - If DynamoDB capacity units are not configured properly, this mechanism can slow down KCL.
  + It has record processors to process the data.
* Kinesis Connector Library
  + An older, deprecated library based off of KCL library.
  + Can be used to write data to S3, DynamoDB, Redshift, ElasticSearch.
  + Replaced by Kinesis Data Firehose for most targets & AWS Lambda for others.
* Third-party libraries like Spark, Flume, Kafka, etc.
* Kinesis Data Firehose
* AWS Lambda
  + Lambda has a library to de-aggregate record from the KPL.
  + It can be used to trigger notifications/ send emails in real time.
  + It has a configurable batch size to determine how much data to process in one go.

## Consumer Models

* Standard Consumer Model
  + Each consumer shares the same streaming pipe and thus it’s limit of 5 transactions per second and 2MB data transfer per second.
  + This model is pull-based, as each consumer can poll the stream every 200 ms at best (One transaction per 200 ms = 5 transactions per second).
  + This model should be used when:
    - The number of consumers is low.
    - The consumers are not latency-sensitive.
    - When costs must be kept low.
* Fan-Out Consumer Model
  + Each consumer has its own pipe and hence does not share service limits with other consumers.
  + This model is push-based, i.e. consumers can subscribe to shards. The messages are pushed to consumers by the service over a long running HTTP2 request made by the consumer of a subscribe to shard API call. This leads to sub 70 ms latency.
  + This model should be used when:
    - The number of consumers is large (Default soft limit is 5).
    - The latency needs to be kept minimum.
  + Works with KCL 2.0 and AWS Lambda.

## Kinesis vs SQS

* When to use Kinesis
  + Since records with the same partition key are sent to the same shard/ record processor (as in streaming MapReduce), Kinesis is beneficial.
    - For example, for counts and aggregations.
  + When ordering of records is important.
  + When multiple applications consume the same stream concurrently and independently. In SQS, applications must wait for the visibility timeout to expire.
    - NOTE: By using SNS-SQS Fan-out architecture, this disadvantage is overcome.
  + When consumer applications need to process the data after a lengthy duration. (Kinesis can store data for up to 7 days).
  + Popular use cases include fast log and event data collection, real time metrics and reports, gaming data feed, IoT, etc.
* When to use SQS:
  + When messaging semantics such as message level acknowledgement and failure is important.
  + When individual message delay is required. One can configure messages to appear in the queue with a delay of up to 15 minutes.
  + Dynamically increasing concurrency/ throughput without external provisioning in case of usage spikes. In Kinesis, autoscaling shards is a slow process.
  + Popular use cases include order/ image processing, buffer and batch messages for future processing, request offloading, etc.

## Miscellaneous Features

* Once data is inserted in Kinesis, it can’t be deleted until it expires (data is immutable).
* Billing is based on
  + Number of provisioned shared for Kinesis Streams.
    - Batching stream records or compressing data can be done to reduce shard requirement.
  + Amount of data streamed for Kinesis Firehose.
* To avoid a ‘hot’ shard, i.e. when producers send all the data to the same shard due to the dominance of a single partition key, one should choose a more distributed partition key.
* The exception *‘ProvisionedThroughputExceeded*’ can appear when one sends more data then provisioned i.e. either the TPS or MB/s is being exceeded.
  + One should make sure there is no ‘hot’ shard.
  + Solutions include retries with back off, increasing the number of shards or ensuring the partition key distributes incoming data properly.

# Kinesis Data Firehose

## Definition

* Near real-time delivery system, similar to Kinesis Data Streams, except that the data is not persisted temporarily.
* The data is processed as soon as it streams in/if at all. Post the analysis, it is stored into S3/Redshift/ElasticSearch/Splunk using a managed consumer service.
* Before being delivered, it is possible to perform some simple data transformations over the data using Lambda.
* Scales automatically.

## Key Terms

* **Kinesis Data Firehose (KDF) delivery system:** The entity one creates and sends data to.
* **Producer:** Entities that send data to KDF delivery systems. E.g. Kinesis Agents, Kinesis Streams, IoT, etc.
* **Buffer Size:** The size of the buffered data (in MB) that KDF will wait before delivering it.
* **Buffer Interval:** The period (in seconds) that KDF will wait for buffered data before delivering it.
  + The minimum interval is 60 seconds, making Firehose a near real-time application.
* **Destination:** The entity KDF will deliver data to. E.g. Redshift, S3, ElasticSearch, Splunk.

## Data Transformation

* When one enables data transformation, KDF will invoke a pre-configured Lambda function in order to transform the data from one format to another.
  + One can opt for AWS-provided Lambda blueprints for certain transformations like Apache Log to CSV/JSON, CloudWatch Logs ingestion, etc. or one can write their own custom Lambda code.
* Data transformation steps:
  + KDF automatically buffers up to 3 MB of data by default.
  + This is because Lambda has a maximum of 6 MB payload size in the requests and the response.
  + When the buffer size or buffer interval is reached, KDF send the data to Lambda.
  + Lambda attempts to transform the data and assigns each record a result before returning it to KDF for delivery
    - **OK:** Indicates that a record transformation is successful.
    - **Dropped:** Indicates that a record was intentionally dropped with the processing logic.
    - **ProcessingFailed:** Indicate that a record failed to be processed. If this happens, one can:
      * Emit errors to CloudWatch logs.
      * Reprocess it at a later time by sending it to S3 under a ‘failed’ folder.
  + KDF then delivers the data to the appropriate destination.

## Cost Model

* Cost vary per region, but the model has two main components:
  + Data Ingested per GB (discounts for high volumes).
  + Data Format conversion per GB.
* Each record is rounded up in size to the nearest 5 KB.
* For each GB, the cost is approximately ~0.029$.

## Miscellaneous Features

* Compression (GZIP, ZIP, or Snappy) is supported when the target is S3.

# Kinesis Analytics

## Definition

* Works in conjunction with other Kinesis types to analyze the data on-the-fly via SQL and then store the data.
* E.g. One can create an application that aggregates the number of different types of items sold within a time-frame. The output of this is itself a stream.
  + This data can stream into another Kinesis stream or dumped into storage services like S3 using Firehose.

## Benefits

* Scales with requirements (up to a certain limit).
* Allows pre-processing of data with filtering or transformations.
* One can do simple things like counts or aggregations over windows of time.
* Useful to detect anomalies in data or for clickstream analytics.
* Also useful in processing IoT data.