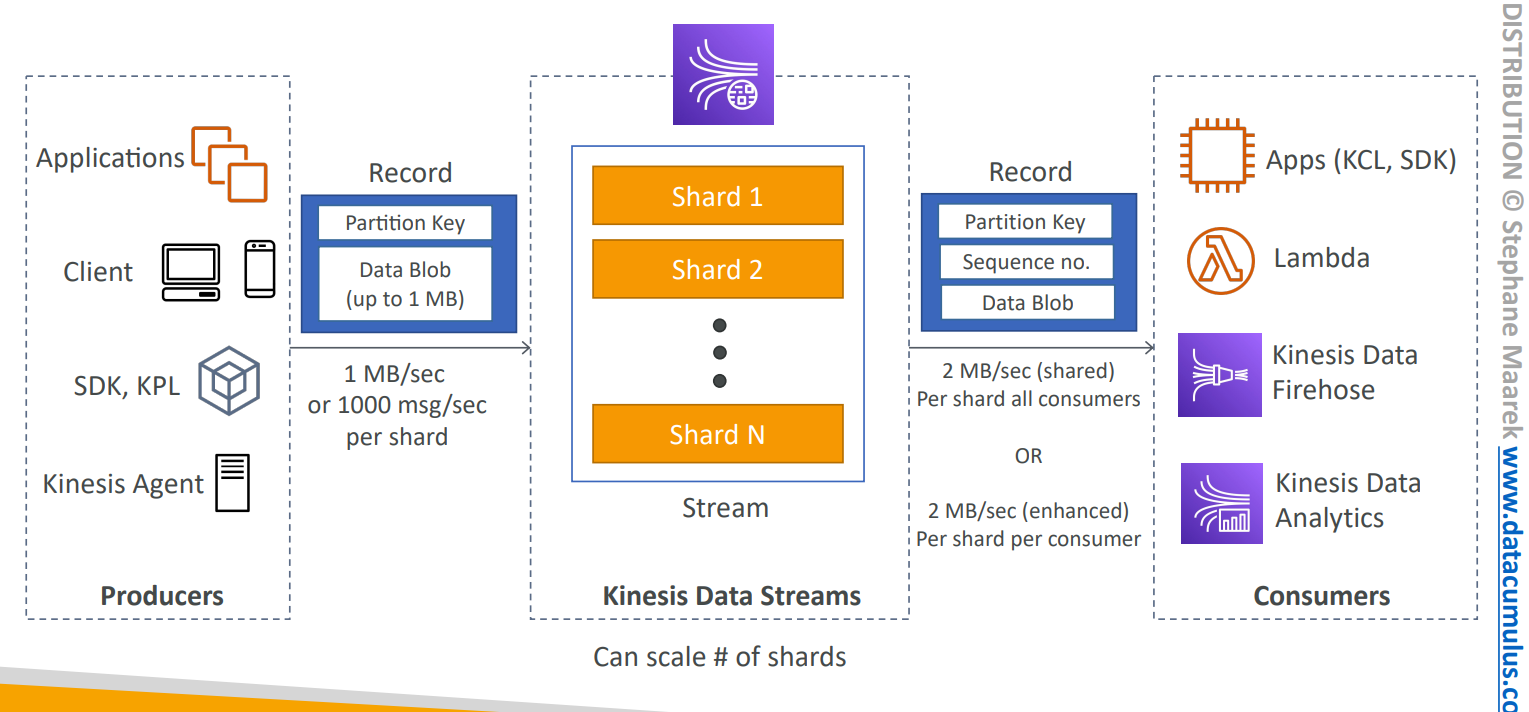
**Kinesis**

After deploying a web application in the server, clients will start hitting the web application then logs will be generated like from which IP address client hit the web application, browser status code etc. Then users want to do analytics on the top of the logs for that AWS provides a service Kinesis to capture, process and analyze the real stream data.

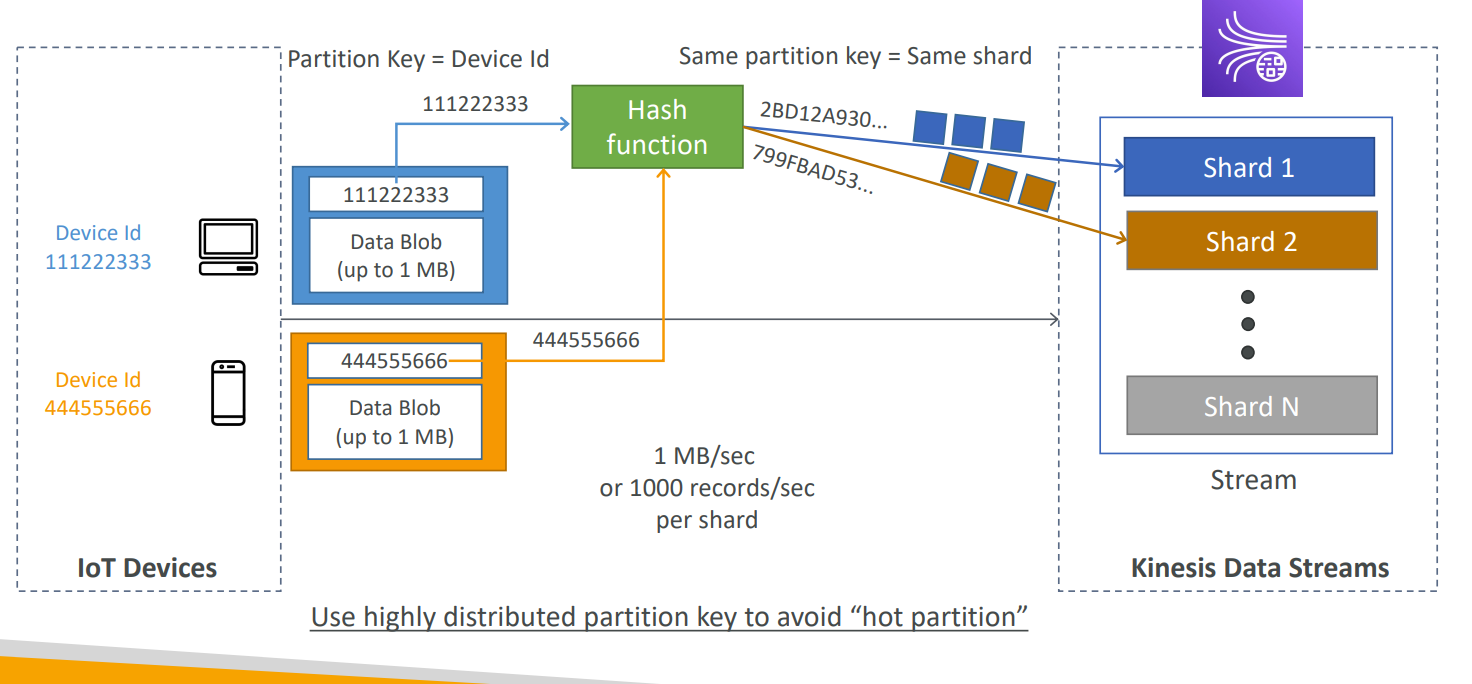
In the Kinesis there are four types of sub service they are:

* **Kinesis Data Streams:** capture, process, and store data streams
* **Kinesis Data Firehose:** load data streams into AWS data stores
* **Kinesis Data Analytics:** analyze data streams with SQL or Apache Flink
* **Kinesis Video Streams:** capture, process, and store video streams

##### **Kinesis Architecture**



###### **Producers**

Producers will put the records using the PutRecord API in the form of partition key i.e which uniquely represent they will be stored in the respective shard and data with the size up to 1 MB in the Kinesis data streams. As no of shards are scale then throughput will increase(1 MB/Sec or 1000 msg/sec per shard) and as well as cost also. The data can be generated from the applications, clients, Software Development Kit, Kinesis Producer Library, Kinesis Agent etc. 

In the figure there are two different IOT devices each having a different device id while putting the data with the partition key. The same partition key will store in the same shard of the Kinesis data stream.

###### **Kinesis Data Stream**

The data which is put by the producers will be stored in the kinesis shard.

**Retention period**

The minimum retention period for the kinesis stream is 1 day to 365 days. Once the data is stored in the Kinesis data stream it won’t be deleted and the kinesis stream will maintain the immutability nature.

**Capacity modes**

In the Kinesis Data Stream there are two types of capacity modes: Provisioned mode & capacity mode. Users can switch between the two modes only two times a day.

**Provisioned mode**

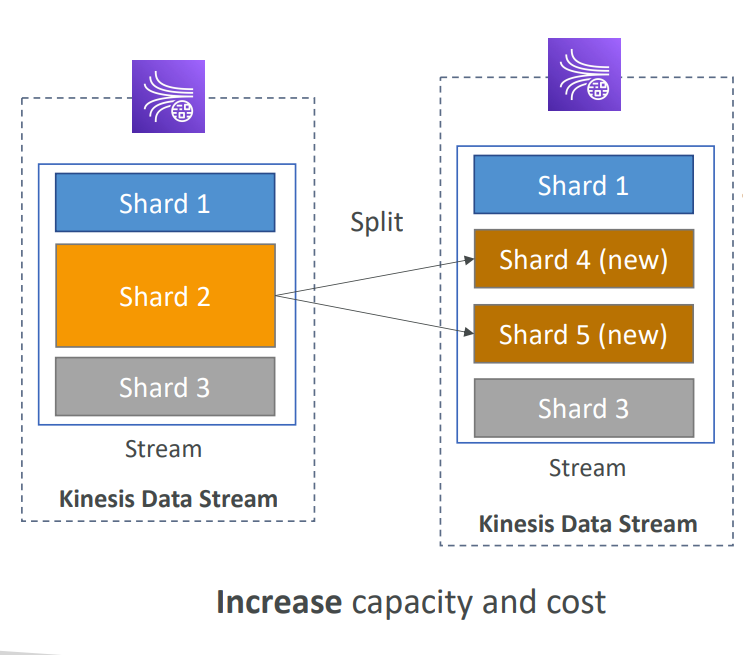
In the Provisioned capacity mode the maximum write capacity is 1 MB/sec & read capacity is 2 MB/sec for each shard. The throughput of a Kinesis data stream in provisioned mode is designed to scale without limits by increasing the number of shards within a data stream.

**On demand mode**

In on-demand mode the stream has a quota of 4 MB/second and 4,000 records per second for writes. By default, these streams automatically scale up to 200 MB/second and 200,000 records per second for writes. We can create a stream with this mode when your data stream's throughput requirements are unpredictable and variable. With on-demand mode, your data stream's capacity scales automatically.

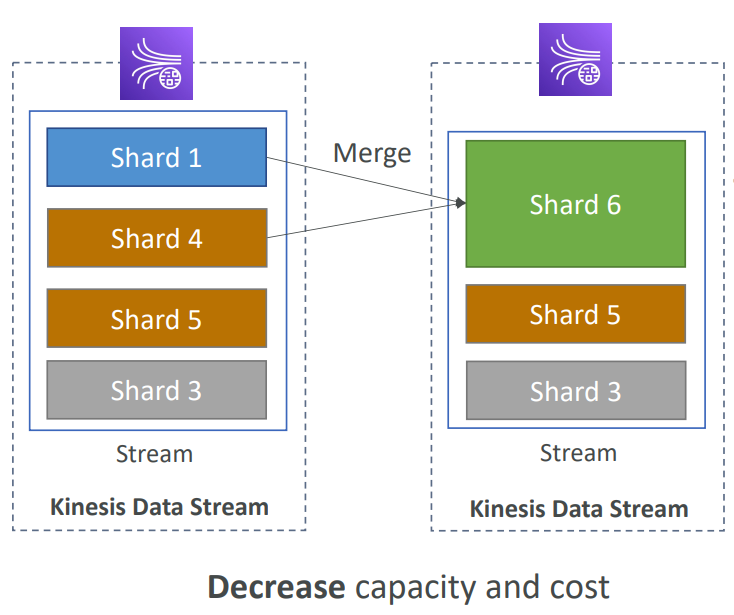
**Scaling operation Merging and Splitting a shard**

**Splitting a shard**



You can scale up a Kinesis Data Stream capacity in provisioned mode by splitting existing shards using the SplitShard API then throughput of the stream and cost will be increased. In a single operation can’t split a shard more than two shards. Splitting is used to divide the hot shard. A hot shard is a shard that is experiencing a higher volume of data throughput than other shards in the same stream.

When a shard becomes hot, it means that it is processing more data records than other shards, and this can lead to increased latency and lower throughput for the stream as a whole. This can happen for a variety of reasons, such as uneven data distribution or a sudden spike in data volume.

**Merging a shard** 

You can scale down capacity by merging two shards using the MergeShard API.

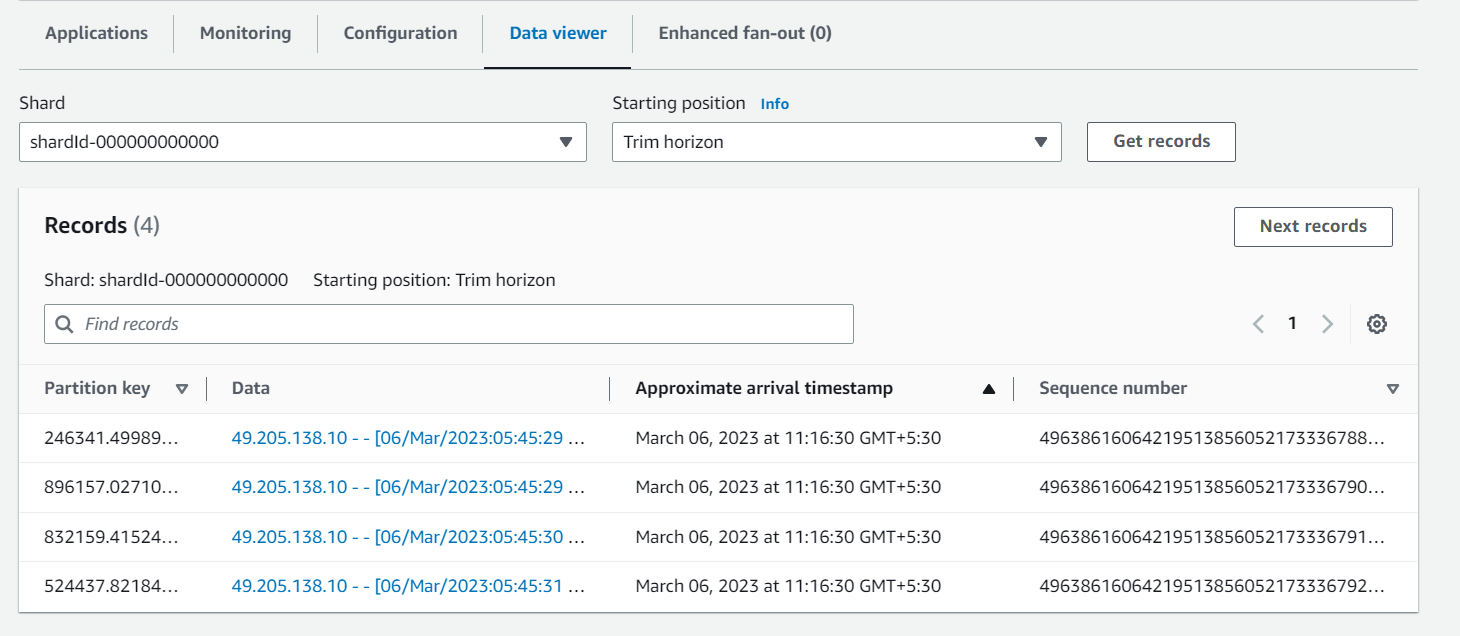
It will decrease the capacity and cost. If the shard is not receiving any new data then it is referred to as cold shard . In a single operation can’t merge more than two shards into one shard. Cold shards can be consolidated with other cold shards to reduce the total number of shards in the stream and reduce costs. This process is known as shard consolidation.

Alternatively, you can use UpdateShardCount API to scale up (or down) a stream capacity to a specific shard count.

###### **Consumer**

Consumers will retrieve the records using the GetRecords from the stream and process them. There are different ways to retrieve the records directly from the DataViewer from the console, AWS Lambda, Kinesis Data Firehose, Kinesis Data Analytics and Kinesis Client library. Data with the size up to 2 MB per shard.

**Retrieving the records from DataViewer kinesis console**



In the producers discussed the way of ordering the data in the shard to retrieve the data from the console. We have selected the shard ID and Iteration types.

While storing the record in the shard it will store based on the **Partition key ID** which uniquely represents which shard has to store.

Approximate **arrival time stamp** will represent the time of arrival for the record in the shard.

The **sequence number** is a unique identifier of the record in a shard of the kinesis stream. If two concurrent clients make a request using the same partition key and if the hash value calculated based on the partition key lies with the hash key range of the same shard in the kinesis stream. they both will try to put to the same shard and will get a unique sequence number in return. However, if hash values for the two clients using the same partition key are different causing the record to go to a different shard, it's possible they get the same sequence number. But the records would belong to different shards in this scenario. If you want the records to be put to the same shard every time. use ExplicitHashKey which falls within the hash key range of the shard.

**Shard Iterators**

The shard iterators are five types they are:

**At Sequence number :** It will retrieve the records from the mentioned sequence number value with the StartingSequenceNumber variable.

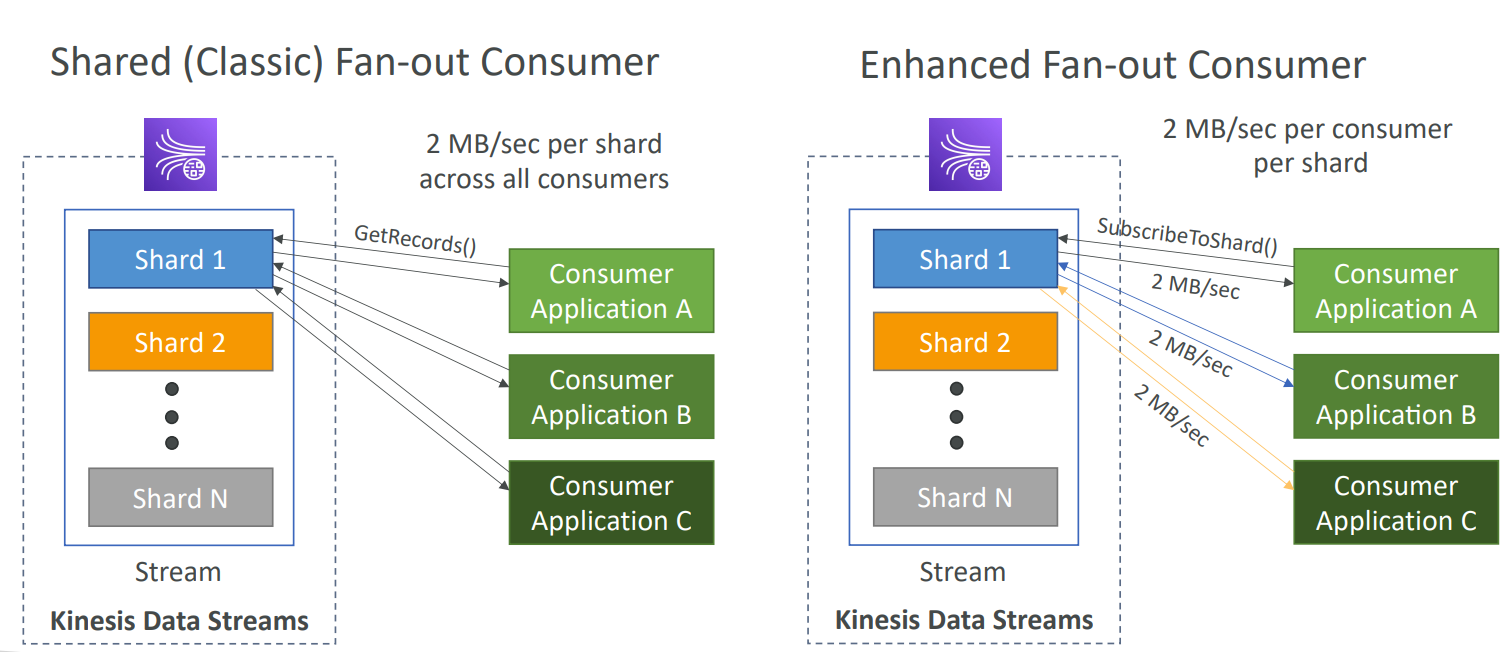
**After Sequence number :** It will retrieve the records after the mentioned sequence number value with the StartingSequenceNumber variable.

**At Timestamp :** It will retrieve the records from the mentioned timestamp value with the Timestamp variable.

**Trim Horizon :** It will retrieve the records from the beginning(Oldest records to new records) in the shard.

**Latest:** it will retrieve the records from the latest(Newest records to old records) in the shard.

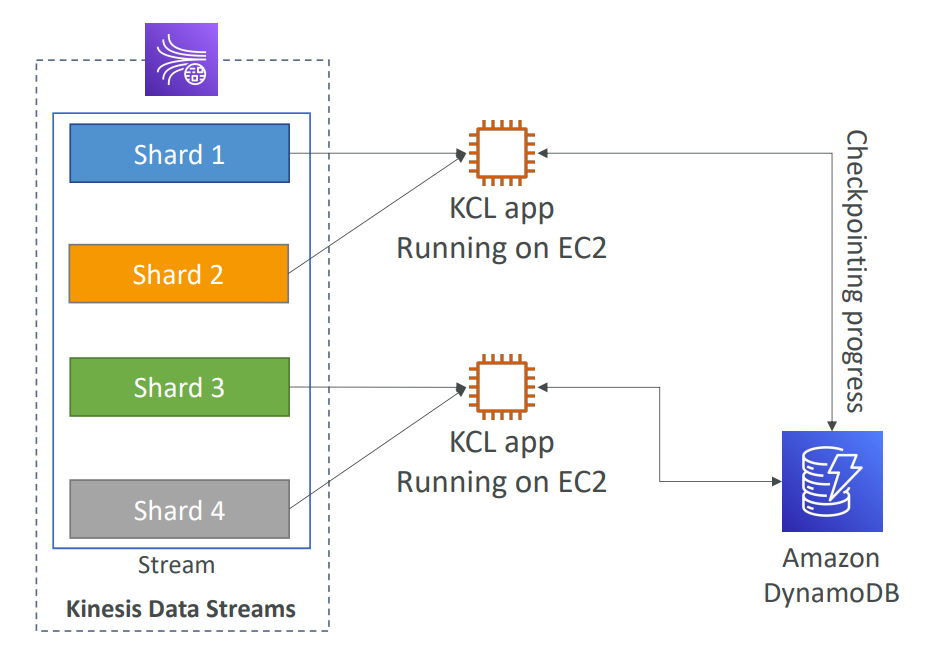
**Consumers Types**



There are two types of consumers. They are Sharded Fan-out consumer and Enhanced Fan-out consumer.

**Sharded Fan-out consumer**: It will use GetRecords() api call to retrieve the records from the shard. It will use the pull method to retrieve the elements, It will retrieve a max of 5 records with the read throughput of 2MB/sec per shard for all consumers. In case if multiple consumers are retrieving the records from the same shard it the throughput will equally distributed to each consumer of max 2 MB/sec.

**Enhanced Fan-out consumer:** It will use SubscribeToShard() api call to retrieve the records from the shard. It will use the pull method to retrieve the elements, It will retrieve the records with the read throughput of 2MB/sec. In case if multiple consumers are retrieving the records from the same shard it the throughput of every consumer has a max of 2 MB/sec.

**KCL**

**Kinesis Client library** is builded using the java library that helps read record from a Kinesis Data Stream with distributed applications sharing the read workload and progress is checkpointed into DynamoDB which helps to track other workers and share the work amongst shards using DynamoDB. In the distributed applications each shard is to be read by only one KCL instance.