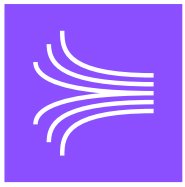


AWS Kinesis

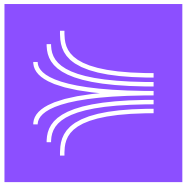
AWS Kinesis is a platform on AWS for real-time data processing. It enables you to collect, process, and analyze large streams of data records in real time. Here's a breakdown of its components and why you might need it:

Components

1. **Kinesis Data Streams**: This service allows you to continuously ingest and process large volumes of data records in real time. You can use it for applications like log and event data collection.
2. **Kinesis Data Firehose**: This is a fully managed service for delivering real-time streaming data to destinations like Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk. It's useful for loading streaming data into data lakes and analytics services.
3. **Kinesis Data Analytics**: This service enables you to process and analyze streaming data using standard SQL queries. It's ideal for creating real-time dashboards and analytics applications.
4. **Kinesis Video Streams**: This service makes it easy to securely stream video data from connected devices to AWS for real-time analytics and storage.

Why You Might Need It

1. **Real-Time Processing**: Kinesis allows you to process and analyze data as it arrives, which is crucial for use cases like fraud detection, real-time analytics, and monitoring.
2. **Scalability**: It can handle massive amounts of data and scale automatically to accommodate increases in data volume, making it suitable for applications with variable or high data throughput.
3. **Flexibility**: It supports various types of data, including logs, metrics, and video, and integrates with other AWS services for further processing and analysis.
4. **Streamlined Data Pipeline**: It simplifies the process of collecting, processing, and storing data, reducing the complexity of building and maintaining custom data pipelines.
5. **Low Latency**: By processing data in real time, you can achieve low-latency insights and actions, which is important for time-sensitive applications.



We saw the basic foundational definitions of the AWS Kinesis ..!!

No, we need to understand few terms and analogies before we learn the Kinesis, those are as follows

Data Streaming:

Data streaming refers to the continuous flow of data generated by various sources, which can be processed, stored, and analysed in real-time. This method is essential for applications that require up-to-date information, such as financial trading platforms, live sports updates, and social media feeds.

Shards on data streaming:

A shard is a fundamental unit of scalability within a data stream. Each shard acts as a partition for the data stream, allowing it to handle a portion of the total data throughput and processing.

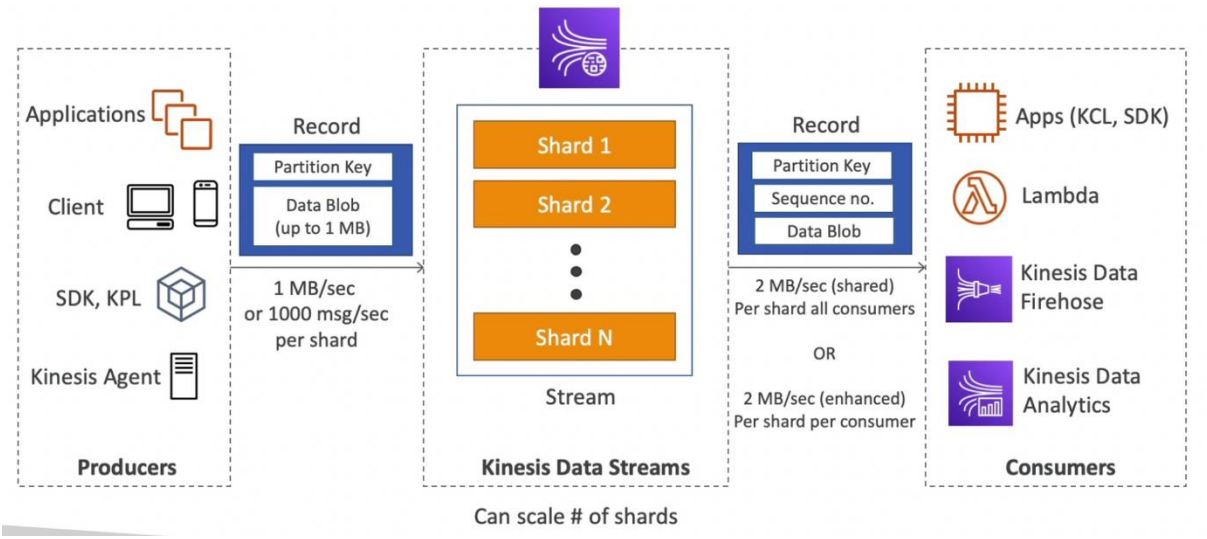
Shards enable parallel processing of the data stream. By splitting the stream into multiple shards, you can process data from different shards simultaneously, improving the overall performance and throughput.

Data stream is nothing but a set of Shards which are working together ..!!

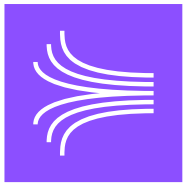
Now we will look into each service – first one would be KDS

Kinesis Data Stream: KDS

Simple basic diagram



- Real-time data streaming service
- **Used to ingest data in real time directly from source**
- **Not Serverless**
- **Data Retention: 1 day (default) to 365 days**
- A record consists of a **partition key** and data blob (**max 1MB**)
- **Once data is inserted in KDS, it can't be modified or deleted (immutability)**
- Records will be ordered in each shard
- Producers use SDK, Kinesis Producer Library (KPL) or **Kinesis Agent** to publish records
- Consumers use SDK or Kinesis Client Library (KCL) to consume the records
- Ability to re-process (**replay**) data



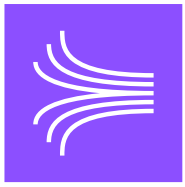
AWS Kinesis

Capacity Modes

- **Provisioned**
  - Publishing: 1MB/sec per shard or 1000 msg/sec per shard
  - Consuming:
    - **Shared:** 2MB/sec per shard (throughput shared between all consumers)
    - **Enhanced Fanout:** 2MB/sec per shard per consumer (dedicated throughput for each consumer)
  - Throughput scales with shards (**manual scaling**)
  - Pay per shard provisioned per hour
- **On-demand**
  - No need to provision or manage the capacity (shards)
  - Default capacity provisioned - **4 MB/sec or 4000 records/sec**
  - Scales automatically based on observed throughput peak during the last 30 days
  - Pay per stream per hour & data in/out per GB

Security

- **KDS is present outside the VPC.** VPC endpoints can be used to access Kinesis from within the VPC.
- Access control for producing/consuming using IAM
- In-flight encryption using HTTPS
- **Server-side at-rest encryption** using KMS or client-side encryption



Now we will deep dive into who are producers and consumers and what data stream is doing … !!

KDS Basic Process

Step 01 → Producers:

- **Data Generation:** Producers generate data that needs to be processed and stored in Kinesis Data Streams. Producers can be applications, services, IoT devices, etc.
- **Putting Data into the Stream:** Producers use the PutRecord or PutRecords API to send data to a Kinesis stream. **Each piece of data is called a record, and it consists of a partition key, and the actual data blob (payload).**

Partition Key: A partition key is provided by the producer. This key determines which shard the data will be stored in.

Data Blob: The actual data payload that you want to store in the stream.

Step 2 → Kinesis Data Stream:

- **Shards:** The stream is divided into shards. Each shard can handle a specific amount of data input and output. The data is distributed **across shards based on the partition key.**
- **Sequence Numbers:** When a record is added to a shard, **KDS assigns a unique sequence number to each record within that shard.** This sequence number helps maintain the order of records within the shard and is used for tracking and retrieval.
- **Retention Period:** The data is stored in the stream for a configurable retention period **(24 hours to 365 days)**, after which it is automatically deleted.

Sequence Numbers

- Sequence numbers are unique identifiers assigned to each record within a shard when it is ingested into KDS. They help maintain the order of records.
- Consumers use sequence numbers to read records in the correct order and to keep track of their progress (checkpointing).
- Sequence numbers are crucial for ensuring the integrity and order of the data being processed.

Shard Assignment:

Kinesis uses the partition key to determine which shard will store the data. The partition key is hashed, and the hash value is used to assign the record to a specific shard.

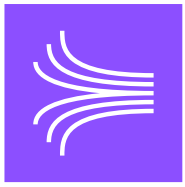
Sequence Number Assignment:

Once the record is assigned to a shard, Kinesis assigns a unique sequence number to the record within that shard. This sequence number is used to order records within the shard and to track records for processing.

Step 03 → Consumers:

- **Reading Data:** Consumers are applications or services that read and process data from the stream. They use the GetRecords API to fetch records from shards.
- **Checkpointing:** **Consumers track the sequence number** of the last record they successfully processed (checkpointing). This allows them to resume reading from where they left off in case of a failure or restart.
- **Processing Data:** Consumers can process the data in real-time, perform transformations, store it in other systems (e.g., databases, data lakes), or trigger other actions based on the data.

Consumers Can read from any shard, regardless of which producer or partition key the records came from.
Flexibility Consumers can be configured to read from multiple shards.
Processing You can dynamically adjust which shards a consumer reads from based on your processing needs.

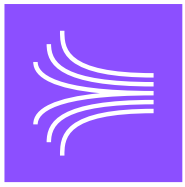


Producers to Shards Mapping

- **One-to-One Mapping Based on Partition Key:** When a producer sends data to a Kinesis Data Stream, it includes a partition key with each record. Kinesis uses this partition key to determine which shard will store the record. This mapping is deterministic, based on a hash of the partition key.
  - Partition Key: A string used to distribute records across shards.
  - Hash Function: Kinesis applies a hash function to the partition key to determine the shard.
  - Shard Assignment: Each record is assigned to one shard based on the result of the hash function.

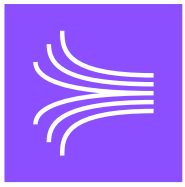
Shards to Consumers Mapping

- **Flexible Mapping: Developers have the flexibility to configure how consumers read from shards.** A single consumer can read from one or multiple shards, and multiple consumers can read from the same shard.
  - Single Shard to Consumer: A consumer can be configured to read from a single shard.
  - Multiple Shards to Consumer: A consumer can read from multiple shards, allowing it to process a wider range of data.
  - Multiple Consumers per Shard: Multiple consumers can read from the same shard to allow for parallel processing of the data.



Producers

- Producers can be:
    - **AWS SDK** - simple producer
    - **Kinesis Producer Library (KPL)** - handles batching, compression and retries
    - **Kinesis Agent** - uses KPL to stream log files
- PutRecord API is used to publish a record in KDS
- **Batching** in PutRecord API **reduces cost and increases throughput** (automatically done by KPL)
- Partition key is passed through a hashing function to determine the shard. Use a well distributed field in the data as the partition key to avoid **hot partition** where most of the data is sent to a single shard.
- Going above the throughput limit (1 MB/s or 1000 records/sec) for any shard will cause **ProvisionedThroughputExceeded** exception. Possible solutions:
    - Retries with exponential backoff
    - Ensure the partition key is well distributed
    - Split the shards (increase number of shards)
- **Use KPL to achieve high write throughput in KDS**
- Embed a primary key within the record to handle duplicate records on the consumer side



There are 2 types of Producers

Direct Producers:

- These are applications or services that send data directly to Kinesis Data Streams using the PutRecord or PutRecords API.
- Examples: Custom applications, AWS Lambda functions, AWS SDKs.

Indirect Producers:

- These producers send data indirectly through intermediaries like Kinesis Data Firehose.
- Examples: Amazon CloudWatch, AWS IoT, AWS Lambda through Kinesis Data Firehose.

Producing Member types

AWS Services:

AWS Lambda: Can act as a producer by pushing data into Kinesis streams.

Amazon CloudWatch: Can send logs and metrics to Kinesis for real-time monitoring and analysis.

Amazon Kinesis Data Firehose: Can deliver streaming data to Kinesis Data Streams.

AWS SDKs and Kinesis Agent:

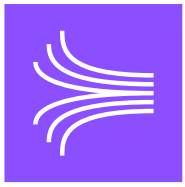
AWS SDKs: Various language-specific SDKs (like Java, Python, Node.js) can be used to create applications that send data to Kinesis streams.

Kinesis Agent: A pre-built Java application that offers an easy way to collect and send data from log files to Kinesis streams.

Custom Applications:

Web or Mobile Applications: Custom applications can use AWS SDKs to send data directly to Kinesis streams.

Server Applications: Backend services and applications can act as producers by sending data to Kinesis streams using the AWS SDK.



Consumers

- Consumers can be:
    - AWS Lambda
    - Kinesis Data Analytics
    - Kinesis Data Firehose
    - Custom Consumer (AWS SDK) – consume at a low level (need to manage complexities)
    - Kinesis Client Library (KCL) - consume at a high level (complexities already managed)

Consumers are 2 types

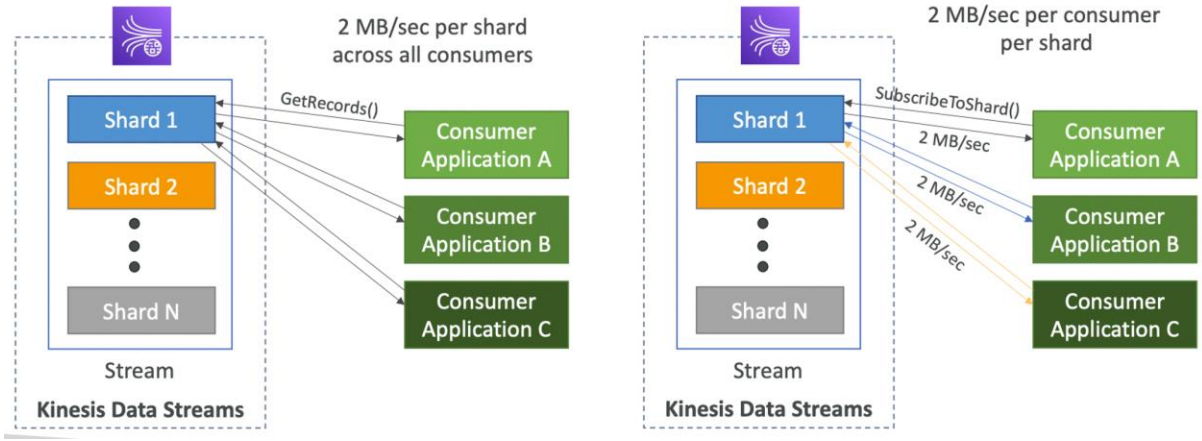
Classic (Shared) Consumer

- Read throughput: 2 MB/sec per shard shared across all consumers
- **Consumers poll data from KDS using GetRecords API call (pull-based)**
- Good for small number of consumers
- Limit of **5 GetRecords API calls/sec per shard**
- Latency **~200 ms (polling)**
- Low cost
- Returns up to 10 MB or up to 10,000 records then throttle for 5 seconds

Enhanced Fan-Out Consumer

- Read throughput: 2 MB/sec per shard dedicated for each consumer
- **Consumers subscribe to a shard using SubscribeToShard API. KDS pushes data to consumers (push-based)**
- Good for large number of consumers
- Latency **~ 70 ms (event driven)**
- High cost
- Default soft limit of **5 consumers** per data stream

Simple diagram for the consumer types



Consumer Member Types

AWS Services:

- **AWS Lambda:** Can be triggered by Kinesis Data Streams to process records in real-time.
- **Amazon Kinesis Data Firehose:** Can consume data from Kinesis streams and deliver it to destinations such as Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk.

Amazon Kinesis Data Analytics:

- Can process data in real-time from Kinesis Data Streams to perform analytics using SQL.

Custom Applications:

- **KCL (Kinesis Client Library) Applications:** Applications written in languages such as Java, Python, Ruby, and Node.js can use the KCL to simplify consuming and processing data from Kinesis streams.

  Kinesis Client Library (KCL) – we need to read a bit more.!!

  - A **Java** library that helps read record from KDS with **distributed applications** sharing the read workload
  - **Maximum number of KCL instances = number of shards**

  - **KCL checkpoints the read progress into DynamoDB** (the application running KCL needs the IAM permissions)
  - By checkpointing in DynamoDB, KCL instances of an application track each other to divide the shards among themselves.
  - KCL can run on any compute resource (cloud or **on-premise**)
  - Records are read in order at the shard level
  - Versions:
  - KCL 1.x (supports shared consumer)
  - KCL 2.x (supports shared & enhanced fan-out consumer)

- **AWS SDKs**: Applications can use AWS SDKs to directly read and process data from Kinesis streams.

Amazon DynamoDB:

- Can act as a consumer by using AWS Lambda to write processed data from Kinesis streams into DynamoDB tables.

Kinesis Service Starting Points

TRIM_HORIZON: This starting point type indicates that the application should start reading from the oldest available data record in the stream. This is useful if you want to process all the data available in the stream, regardless of when it was added.

LATEST: This starting point type indicates that the application should start reading from the most recent record in the stream at the time the application starts. This is useful if you only want to process new data that arrives after the application starts running.

AT_TIMESTAMP: This starting point type allows you to specify a specific timestamp from which to start reading records. The application will start reading from the first record with a timestamp equal to or greater than the specified timestamp. This is useful for replaying data from a particular point in time.

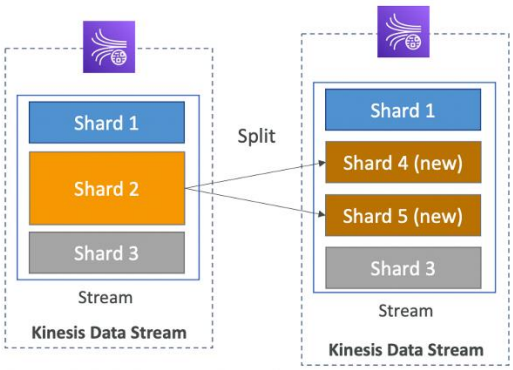
AT_SEQUENCE_NUMBER: This starting point indicates that the application should start reading from the record with the specified sequence number. This is useful if you know the exact sequence number of the record from which you want to start processing.

AFTER_SEQUENCE_NUMBER: This starting point indicates that the application should start reading immediately after the record with the specified sequence number. This is useful if you want to skip a specific record and start processing from the next one.

We will see 2 more terms ..!!

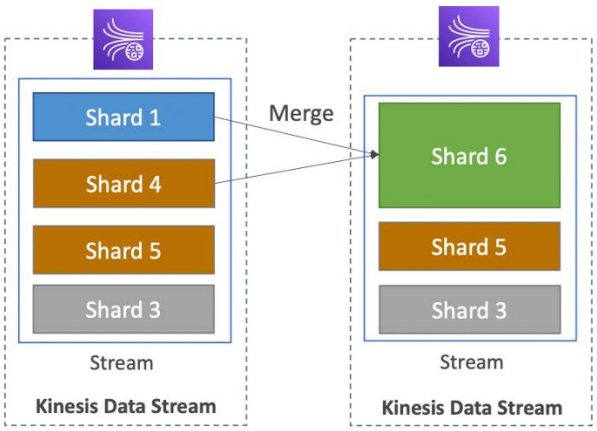
Shard Splitting

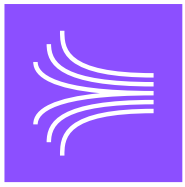
- Split a **hot shard** (high traffic) into 2 shards
- Increases the stream capacity by equivalent of adding 1 shard
- The old shard is closed and will be deleted when the data in it expires
- Can't split more than 2 shards in a single operation



Shard Merging

- Merge two **cold shards** (low traffic) into a single shard
- Decreases the stream capacity by equivalent of removing 1 shard
- Old shards are closed and will be deleted when the data in them expires
- Can't merge more than 2 shards in a single operation





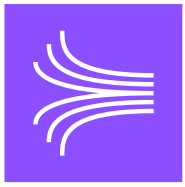
Kinesis Data Firehose: KDF

Amazon Kinesis Data Firehose is a fully managed service designed to simplify the process of loading streaming data into data lakes, data stores, and analytics services.

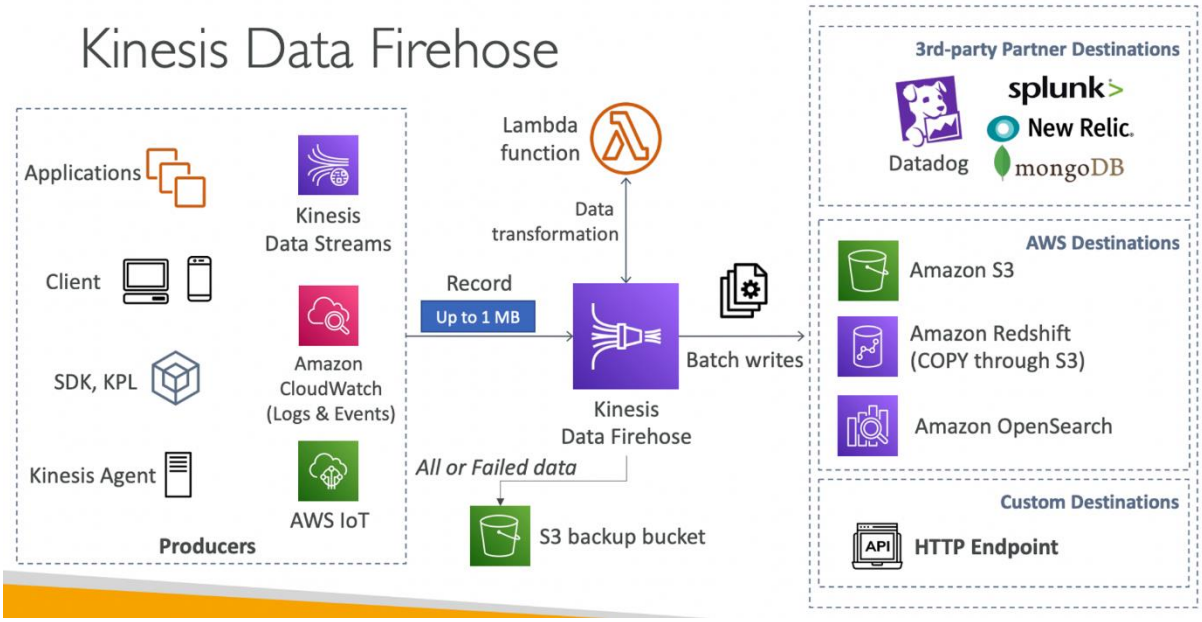
- Used to load streaming data into a target location
- Serverless
- Writes data in batches efficiently (near real time)
  - Buffer size (size of the batch) - 1 MB to 128MB (default 5MB)
  - Buffer interval (how long to wait for buffer to fill up) - 60s to 900s (default 300s)
  - Greater the buffer size, higher the write efficiency, longer it will take to fill the buffer
- Can ingest data in real time directly from source
- Auto-scaling
- Destinations:
  - AWS: Redshift, S3, OpenSearch
  - 3rd party: Splunk, MongoDB, DataDog, NewRelic, etc.
  - Custom HTTP endpoint
- Pay for data going through Firehose (no provisioning)
- Custom data transformation using Lambda functions (not supported in KDS)
- No replay capability (does not store data like KDS)

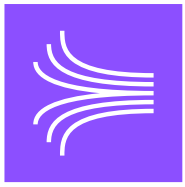
Use Cases

- **Log and Event Data Aggregation**: Aggregate and deliver logs and events from various sources to S3 for storage or to Elasticsearch for search and analytics.
- **Streaming Data Analytics:** Send streaming data to Redshift or S3 for real-time analytics and reporting.
- **Data Lake Ingestion:** Ingest raw or processed data into S3 as part of a data lake architecture.



AWS Kinesis


Kinesis Data Firehose
Applications
Client
SDK, KPL
Kinesis Agent
Producers
Kinesis Data Streams
Amazon CloudWatch (Logs & Events)
AWS IoT
Lambda function
Data transformation
Record
Up to 1 MB
Kinesis Data Firehose
Batch writes
All or Failed data
S3 backup bucket
3rd-party Partner Destinations
Datadog
splunk>
New Relic.
mongoDB
AWS Destinations
Amazon S3
Amazon Redshift (COPY through S3)
Amazon OpenSearch
Custom Destinations
API
HTTP Endpoint



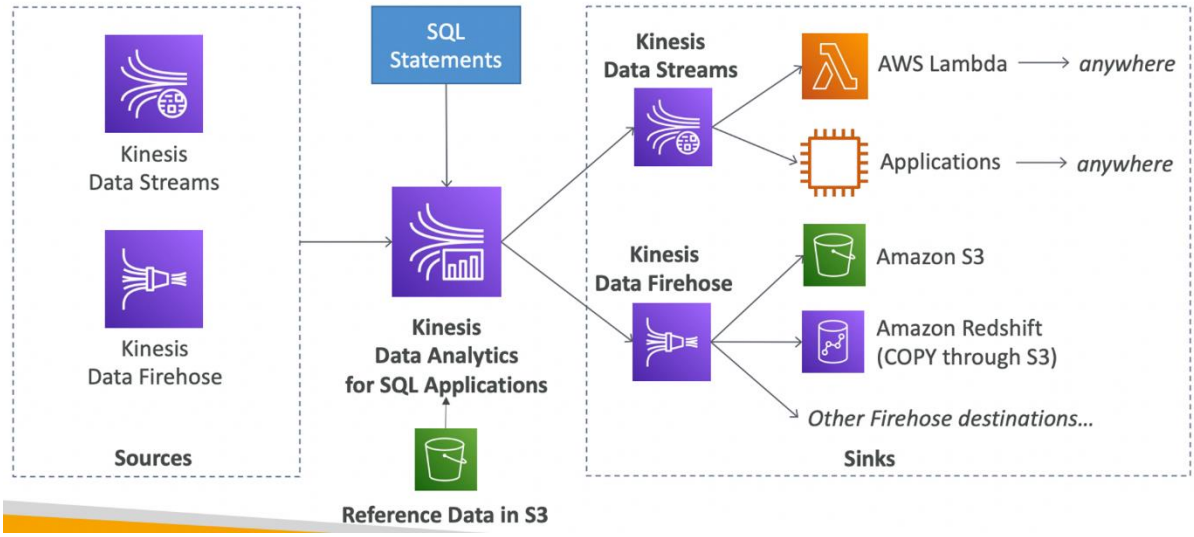
Kinesis Data Analytics: KDA

Amazon Kinesis Data Analytics is a fully managed service that enables you to process and analyze **real-time streaming data using SQL**.

- Perform **real-time analytics on Kinesis streams** using **SQL**
- Creates streams from SQL query response
- **Cannot ingest data directly from source** (ingests data from **KDS** or **KDF**)
- **Auto-scaling**
- **Serverless**
- Pay for the data processed (no provisioning)
- Use cases:
    - Time-series analytics
    - Real-time dashboards
    - Real-time metrics

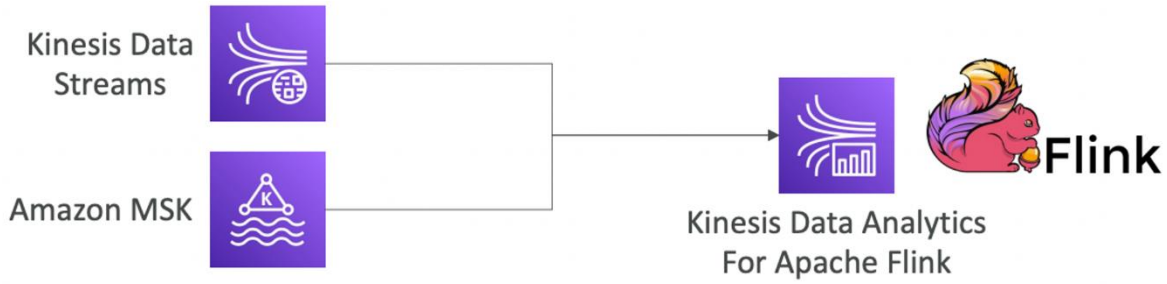
Use cases:

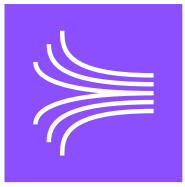
Fraud Detection: Analyze transactions in real-time to detect fraudulent activities and trigger alerts.
Monitoring and Alerting: Monitor system metrics, logs, or user activity in real-time and generate alerts based on predefined conditions.
Clickstream Analysis: Analyze user behaviour and clickstream data to understand user interactions and optimize the user experience.



KDA for Apache Flink

- Use Flink (Java, Scala or SQL) to process and analyze streaming data (advanced querying capability)
- Can ingest data from **KDS** or **MSK** (to ingest data from KDF, use KDA for SQL)
- Used to run Apache Flink application on a managed cluster on AWS
    - provisioning compute resources, parallel computation, automatic scaling
    - application backups (implemented as checkpoints and snapshots)





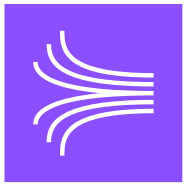
Kinesis Video Streams: KVS

Amazon Kinesis Video Streams (KVS) is a fully managed service designed to collect, process, and analyze video streams in real-time.

Use Cases

Surveillance and Monitoring: Collect and process video from security cameras or IoT devices for real-time monitoring, alerting, and long-term storage.

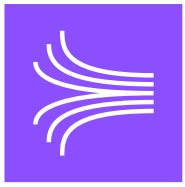
Media and Entertainment: Stream live events, such as concerts or sports, and provide real-time analytics or interactive experiences for viewers.



AWS Kinesis

At last, …!!

Service	*When to Choose*	*Fundamental Differences*
Kinesis Data Streams (KDS)	When you need to collect and process real-time streaming data with custom processing.	**Real-time Data Ingestion:** Allows for custom data processing and transformation. **Shards:** Data is divided into shards for parallel processing. **Custom Consumers:** Requires custom applications or Kinesis Client Library (KCL) for data consumption.
Kinesis Data Firehose (KDF)	When you need to automatically deliver streaming data to destinations like S3, Redshift, Elasticsearch, or Splunk.	**Managed Data Delivery:** Automatically handles data delivery to destinations. **Transformation:** Supports data transformation using AWS Lambda. **No Custom Code:** Simple setup with minimal code for delivery.
Kinesis Data Analytics (KDA)	When you want to process and analyze streaming data in real-time using SQL.	- **Real-Time Analytics:** Applies SQL queries to real-time data streams. **Integration:** Works with KDS and KDF. **No Coding Required:** Uses SQL for processing, simplifying analytics tasks.
Kinesis Video Streams (KVS)	When you need to collect, process, and analyze video data streams from cameras and other video sources.	**Video Data:** Specifically designed for video streaming and processing. **Integration:** Works with AWS services for video analytics, like Amazon Rekognition. **Storage and Playback:** Stores and enables time-indexed retrieval of video data.



AWS Kinesis

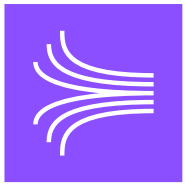
Single line Summary

Kinesis Data Streams (KDS): Ideal for real-time applications needing custom data processing, like log aggregation or real-time event processing.

Kinesis Data Firehose (KDF): Best for scenarios where you need to easily deliver data to AWS destinations without managing infrastructure, such as archiving logs or delivering real-time data to data warehouses.

Kinesis Data Analytics (KDA): Suitable for applications requiring real-time data analysis using SQL, like monitoring metrics or performing real-time analytics on streaming data.

Kinesis Video Streams (KVS): Designed for video data applications, such as surveillance, media streaming, or IoT video analysis.



AWS Kinesis Numerical Questions:

https://medium.com/@lisireddy/aws-kinesis-numerical-questions-b2b357a35bae

AWS Kinesis scenario-based Questions:

https://lisireddy.medium.com/aws-kinesis-scenario-based-questions-d792eafea699

Wish you the best …! Happy Learning ..!

Yours' Love (@lisireddy across all the platforms)