Amazon Kinesis and Amazon MSK overview

Analyzing real-time data streams on AWS

Zhang, Wen-Hao
Cloud Support Engineer, AWS Premium Support
July 10, 2021

Agenda

- Amazon Kinesis overview
  - Introduction for Kinesis family
  - Key concept of Kinesis data stream
  - Lab
- Amazon MSK overview
  - Key concept of MSK
  - Lab
- Amazon Kinesis vs. Amazon MSK
- Online testing

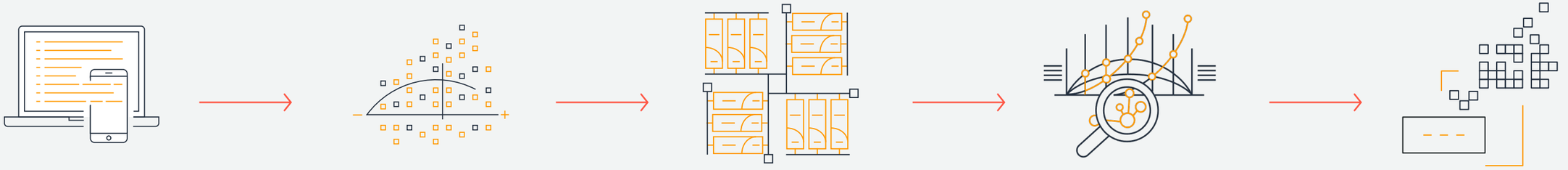
aws

Amazon Kinesis overview

aws

Enabling real-time analytics

Data streaming technology enables a customer to ingest, process and analyze high volumes of high velocity data from a variety of sources in real time



Source

Stream ingestion

Stream storage

Stream processing

Destination

aws

Streaming data with AWS

Easily collect, process, and analyze data streams in real time

Amazon Kinesis Data Streams

Capture and store data streams

Amazon Kinesis Data Analytics

Analyze data streams in real time

Amazon Kinesis Data Firehose

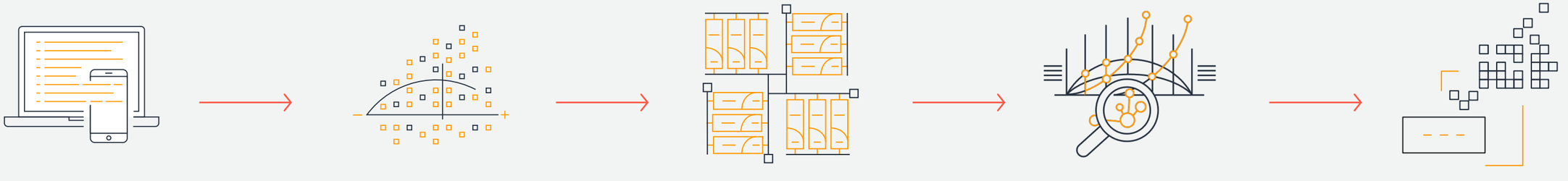
ETL streaming into streams, data lakes and warehouses

Amazon Managed Streaming for Kafka

Capture and store data streams

aws

An Example Architecture



Source | Stream ingestion | Stream storage | Stream processing | Destination

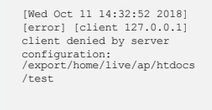
Mobile device

Metering

Click streams

IoT sensors

Logs



Stream Ingestion

Amazon Kinesis Agent

Amazon Kinesis Producer Library

AWS SDKs

Amazon Kinesis Data Streams

Real-Time Applications (seconds)

Amazon Kinesis Data Analytics

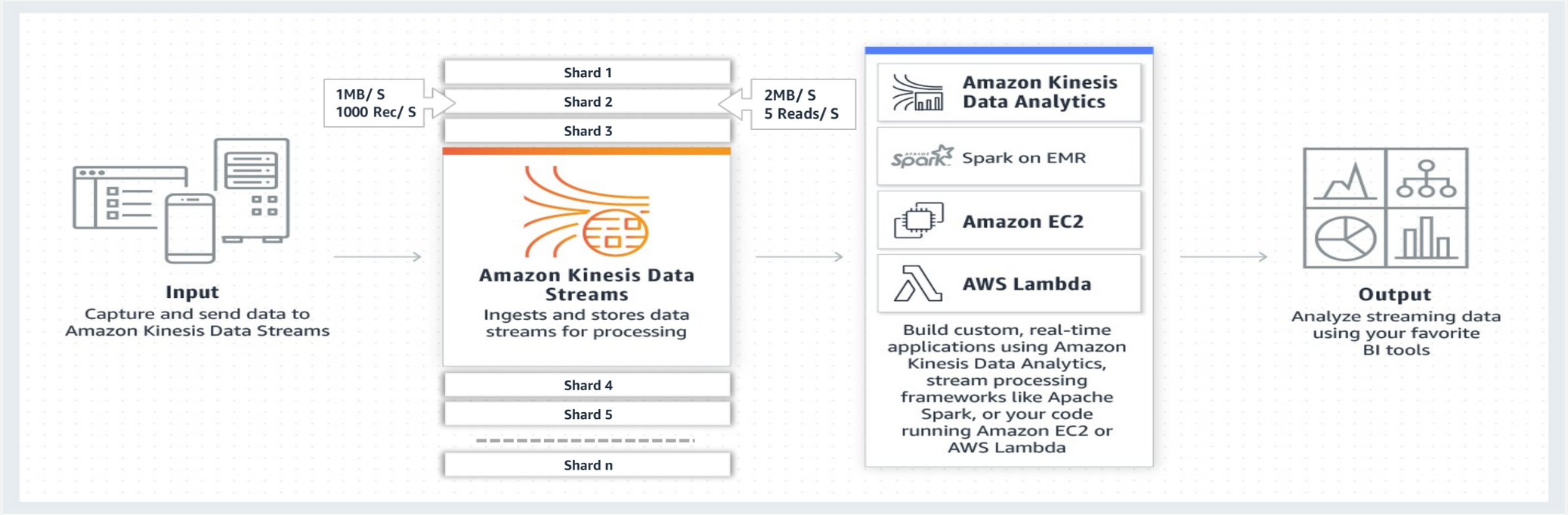
Amazon Kinesis Consumer Library

Streaming ETL (minutes)

Amazon Kinesis Data Firehose

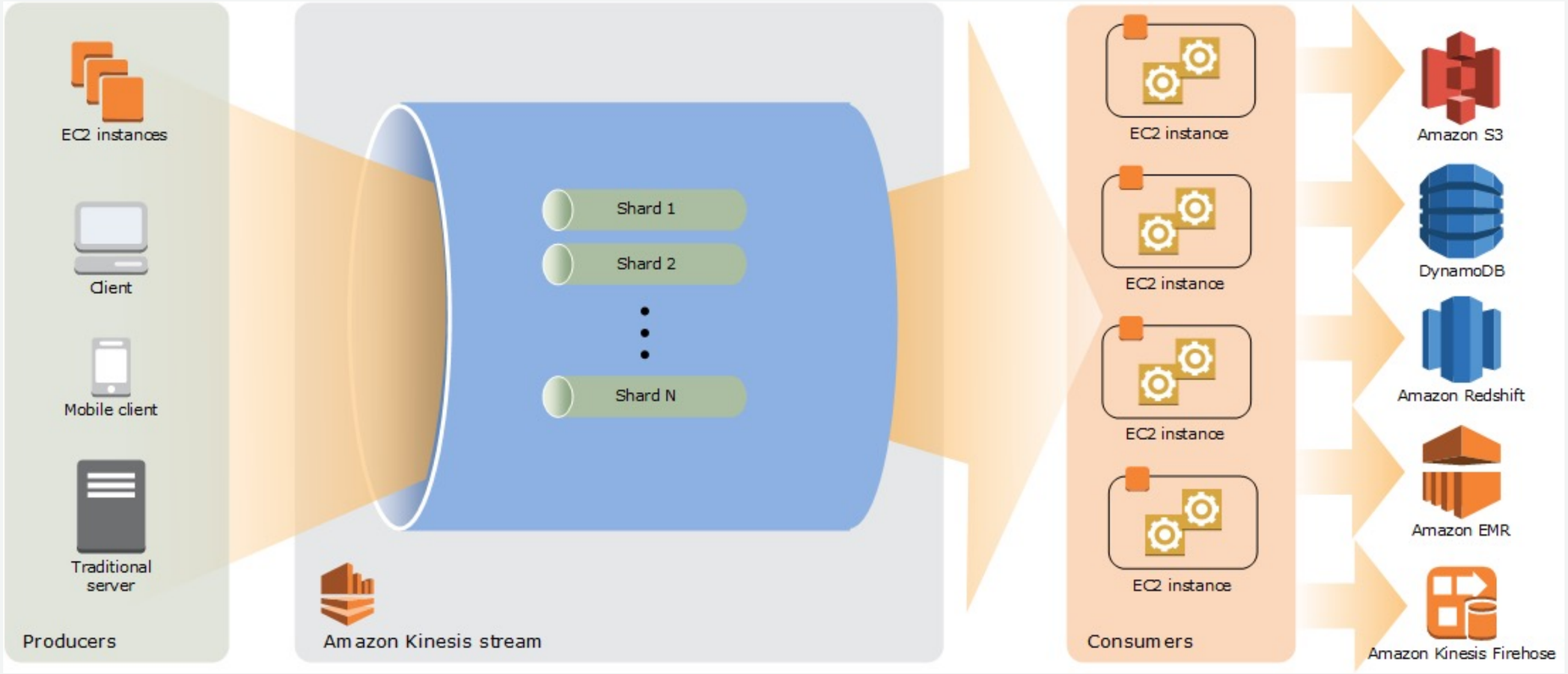
Amazon S3
Amazon Redshift
Amazon Elasticsearch
Splunk

Amazon Kinesis Data Streams

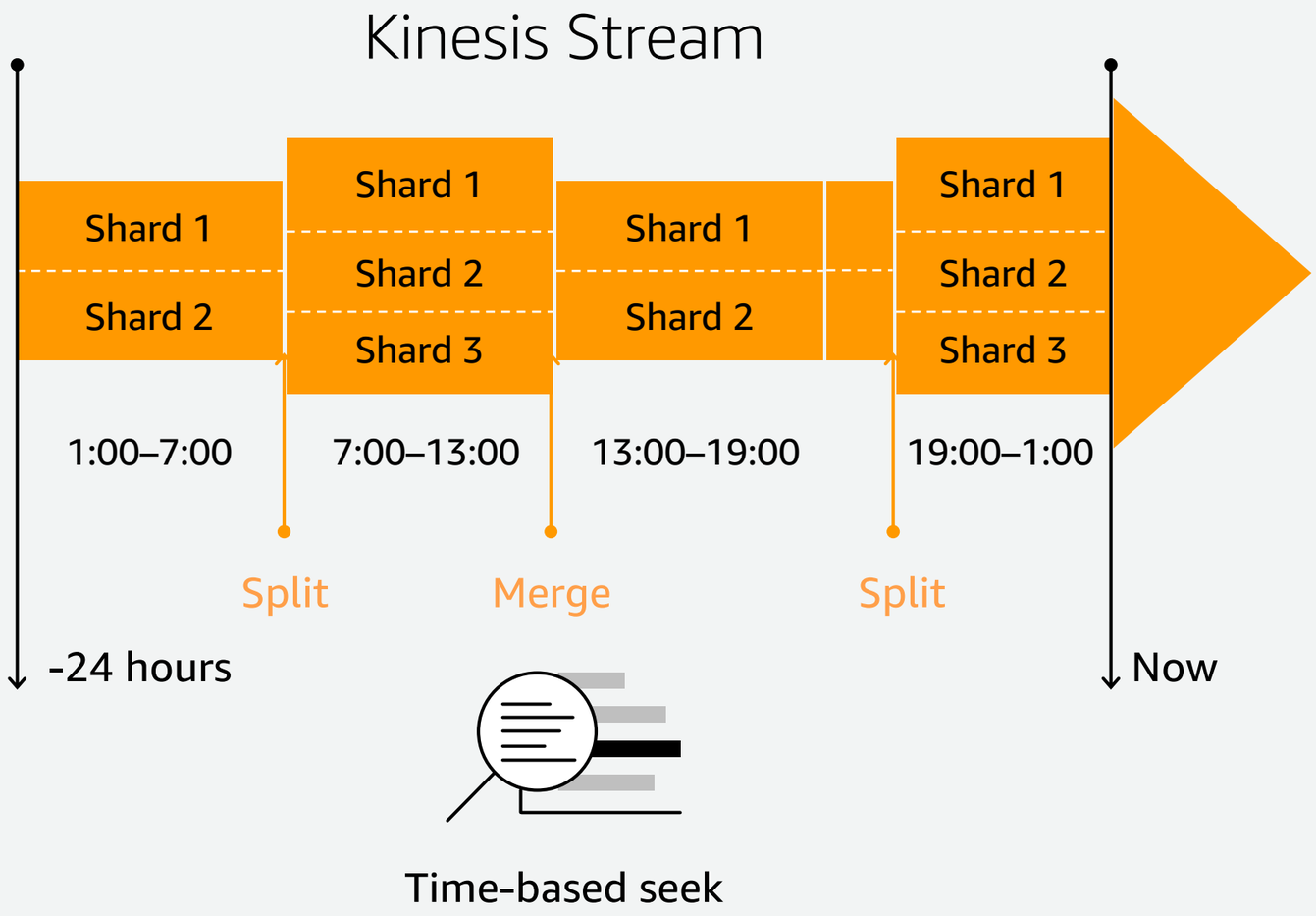


- Easy administration and low cost
- Real-time, elastic performance
- Secure, durable storage
- Available to multiple real-time analytics applications
- Average latency of 200ms with one standard consumer
- Enhanced Fan Out with SubscribeToShard API offers typical average latency of 70 ms

Kinesis Data Streams High-Level Architecture



Managed ability to capture & store data



Time-based seek

- Data streams are made of Shards
- Each shard ingests data up to 1MB/sec, and up to 1000 TPS
- Each Shard emits up to 2 MB/sec
- All data is stored by default for 24 hours and can be extended up to 365 days
- Scale Kinesis data streams by splitting or merging Shards
- Replay data inside of 24 hour – 365 day window

aws

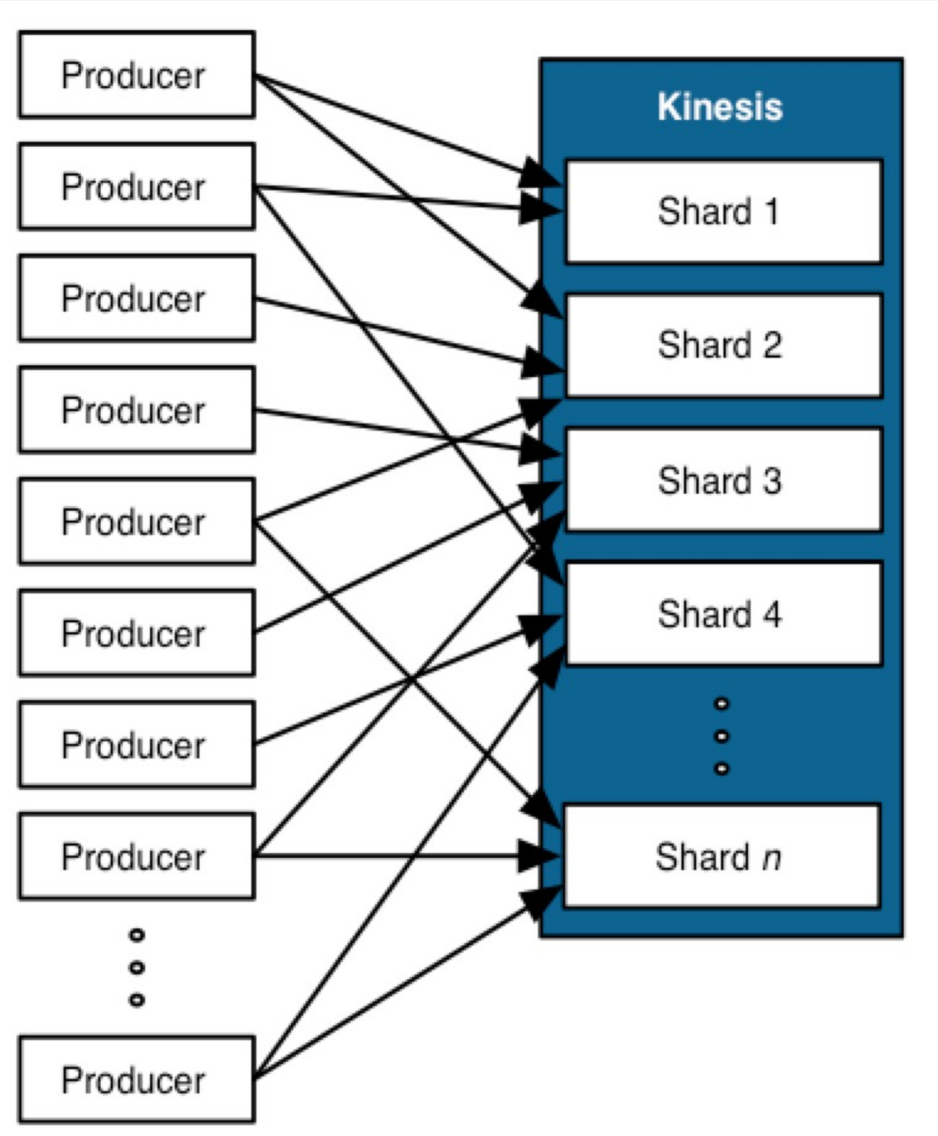
Amazon Kinesis Data Streams Core Concepts

Partition Keys

- Supplied by producer with each message
- Used to distribute data across shards

Sequence Numbers

- Returned to producer for each successfully written message
- Not an index to data in shard
- Can be used to enforce strict ordering



aws

Amazon Kinesis Data Streams Core Concepts

Message Ordering

Unordered processing

- Randomize partition key to distribute events over many shards and use multiple workers

Exact order processing

- Control partition key to ensure events are grouped into the same shard and read by the same worker

Need both? Use global sequence number

Amazon Kinesis Data Streams Data Producers

SDKs
- Publish directly from application code via *PutRecord* and *PutRecords* APIs

Kinesis Agent
- Tail log files and forward lines as messages to Kinesis Data Streams.

Kinesis Producer Library (KPL)
- Background process aggregates and batches messages.
- Producer application calls *addUserRecord* method.

3rd-party and open source
- Log4j appender
- Flume, fluentd source libraries

aws

Kinesis Data Streams Standard Consumer

No additional cost

5 transactions/sec shard limit means 5 consumers can run concurrently

All consumers share the 2MB/sec shard limit

Lowest latency with a single consumer is about 200 millisecs

Uses the polling model to get records from a shard

To enable more than 5 consumers, use the fan-out pattern

aws

Kinesis Data Streams Enhanced Fan-Out

Enhanced fan-out allows customers to scale the number of functions reading from a stream in parallel while maintaining performance.

HTTP/2 data retrieval API improves data delivery speed between data producers and Lambda functions by more than 65%

Provides dedicated 2MB/s bandwidth per consumer – no bandwidth sharing with other consumers

Uses a push model to push records to consumers instead of a polling model

Not subject to the 5 transactions/sec/shard limit of the standard consumer

Achieves an average latency of 70 millisec

Does have an additional cost

aws

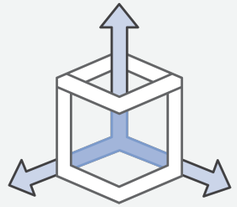
Amazon MSK overview

aws

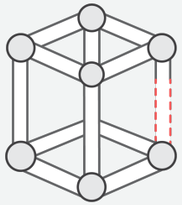
Challenges operating Apache Kafka

Difficult to setup

Tricky to scale



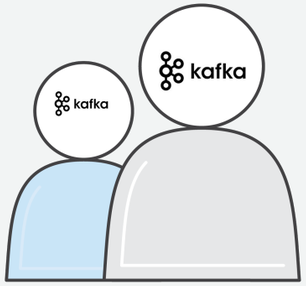
Hard to achieve high availability



AWS integrations = development

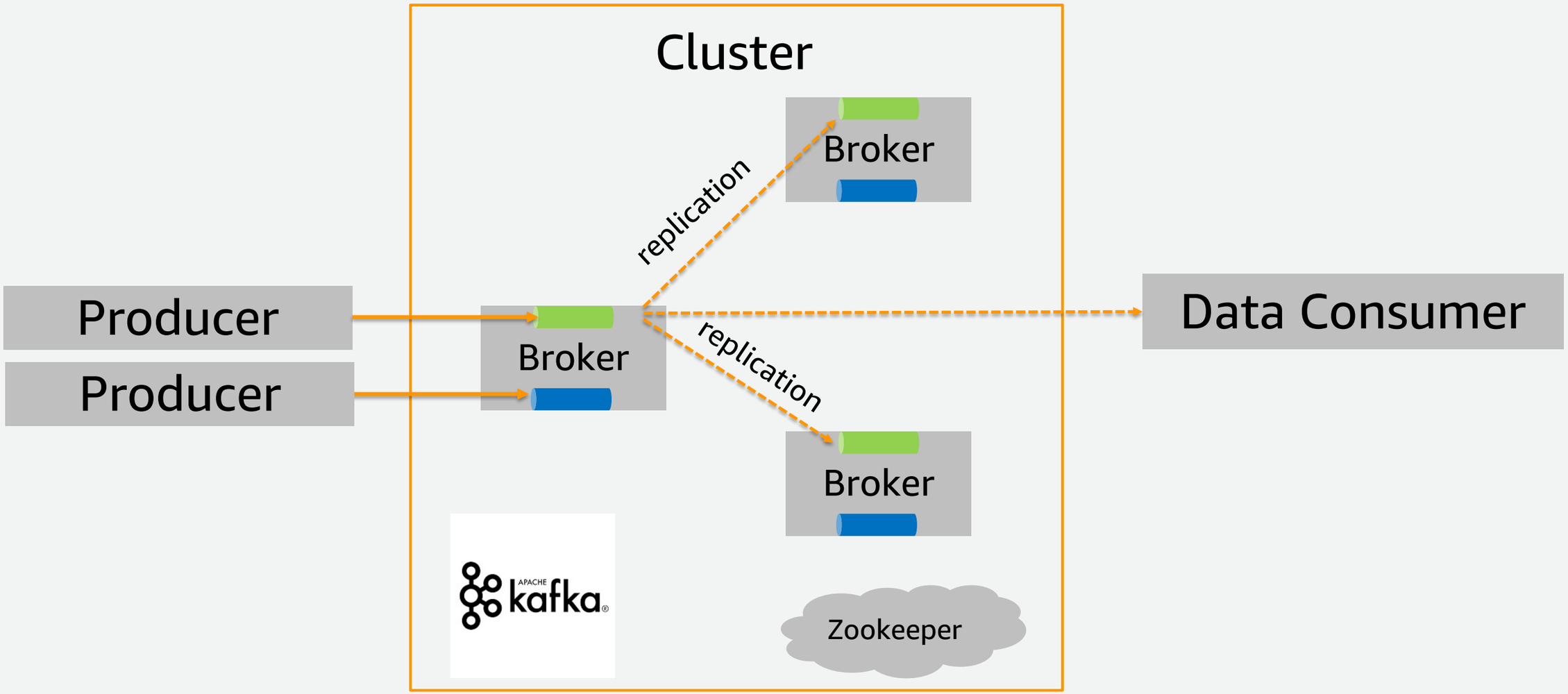
No console, no visible metrics

$$f(kafka_{usage}) = \sum_{n=1}^{\infty}(SRE)$$



aws

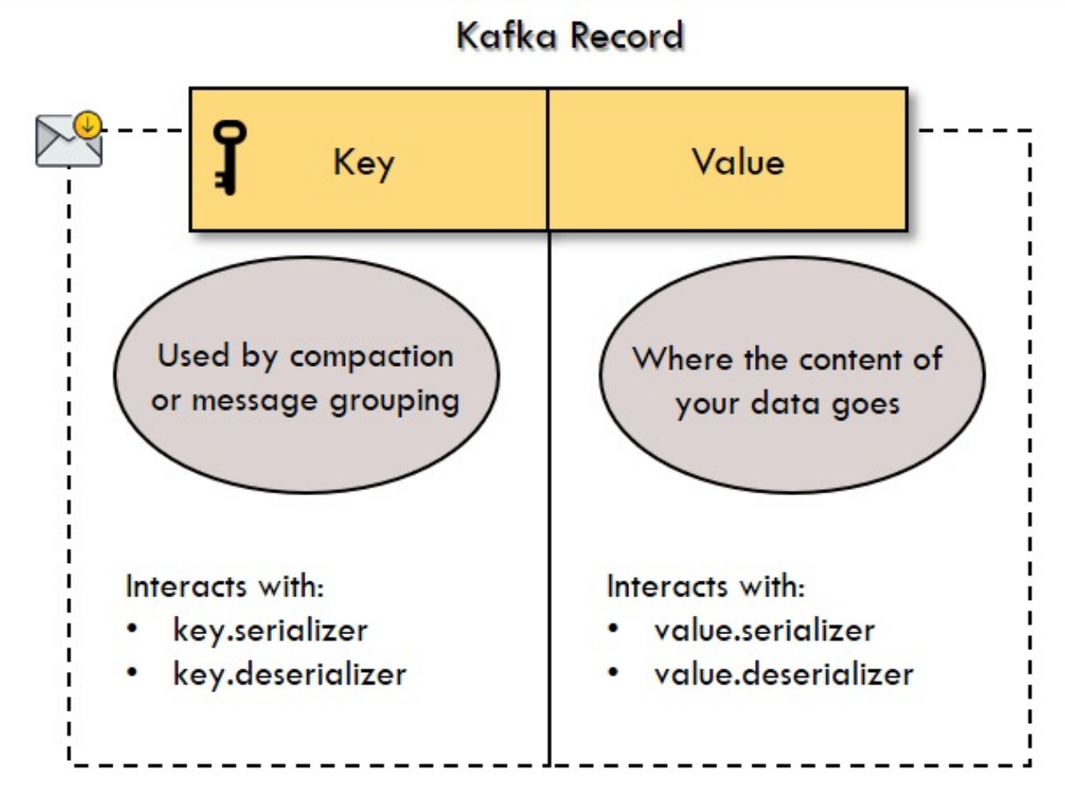
Apache Kafka Anatomy 101



aws

Kafka Record

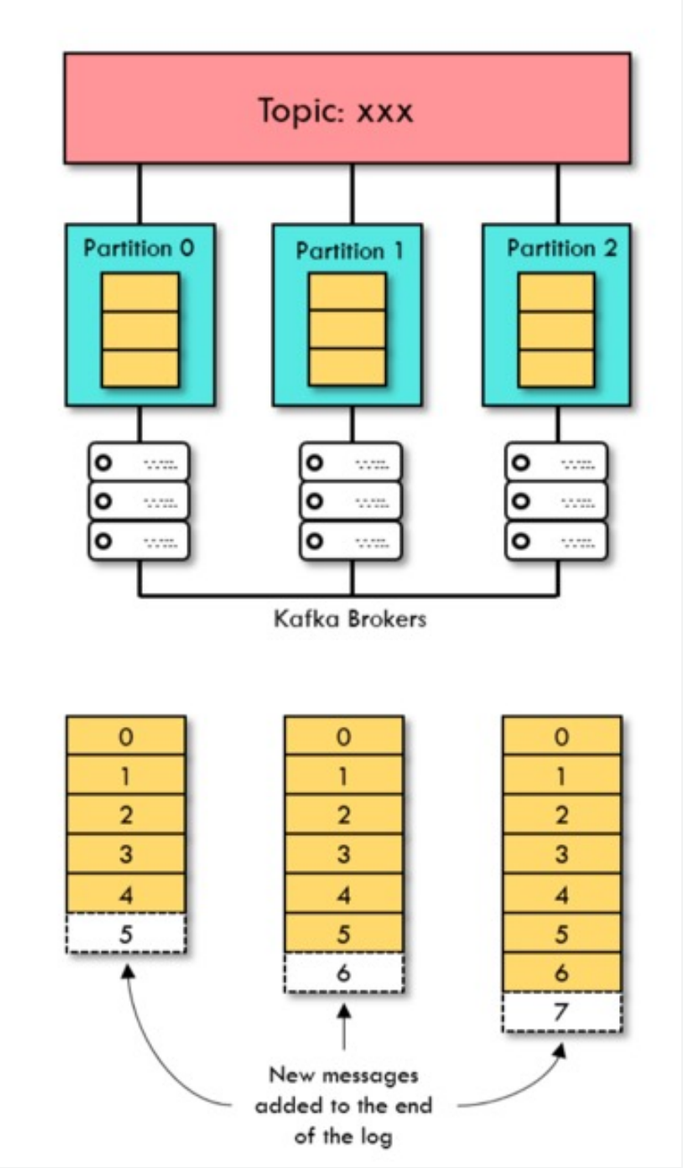
- Every message publish to Kafka called “**Record**”
- **Record** contain two parts:
    - **Key :** Used by compaction or for message grouping
    - **Value :** The content of data goes



aws

Topics and partitions

- Topics: a particular stream of data
    - Similar to a table in a database (without all the constraints)
    - You can have as many topics as you want
    - A topic is identified by its name
- Topics are split in partitions
    - Each partition is ordered
    - Each message within a partition gets an incremental id, called offset



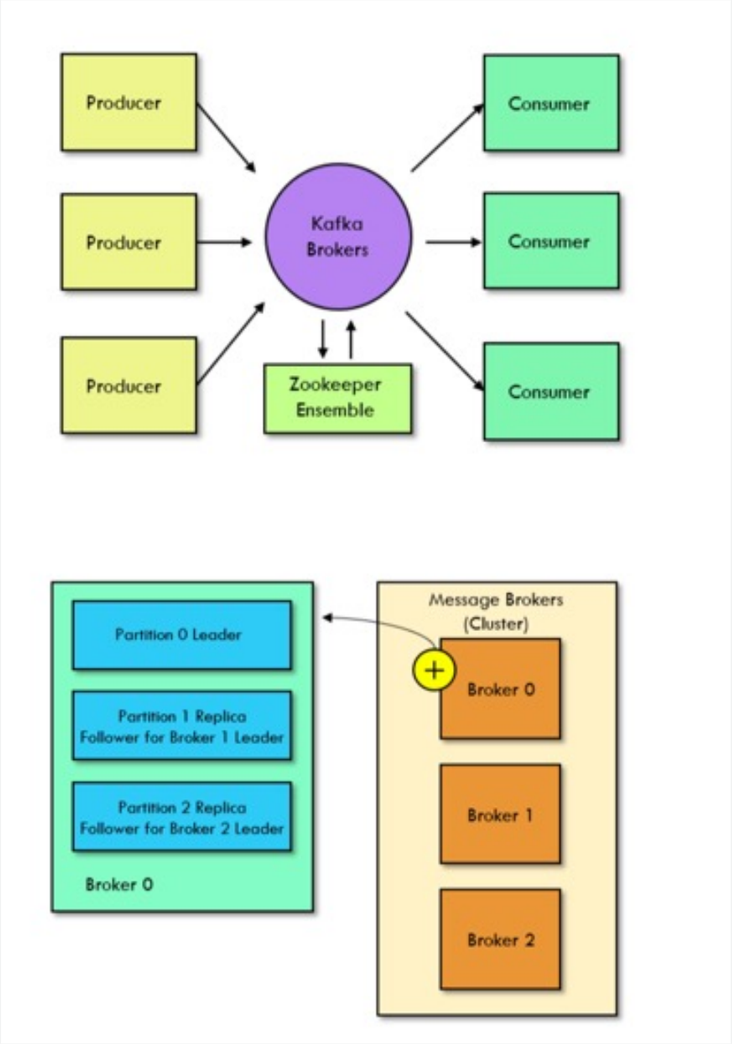
Topics and partitions

- **Offset** only have a meaning for a specific partition.
  • E.g. offset 3 in partition doesn’t represent the same data as offset 3 in another partition
- Order is guaranteed only within a partition (not across partitions)
- Data is kept only for a limited time (default is **one** weeks)
- Once the data is written to a partition, it can’t be changed (immutability)
- Data is assigned randomly to a partition unless a **key** is provided (more on this later)
- You can have as many partitions per topics as you want)

aws

Brokers

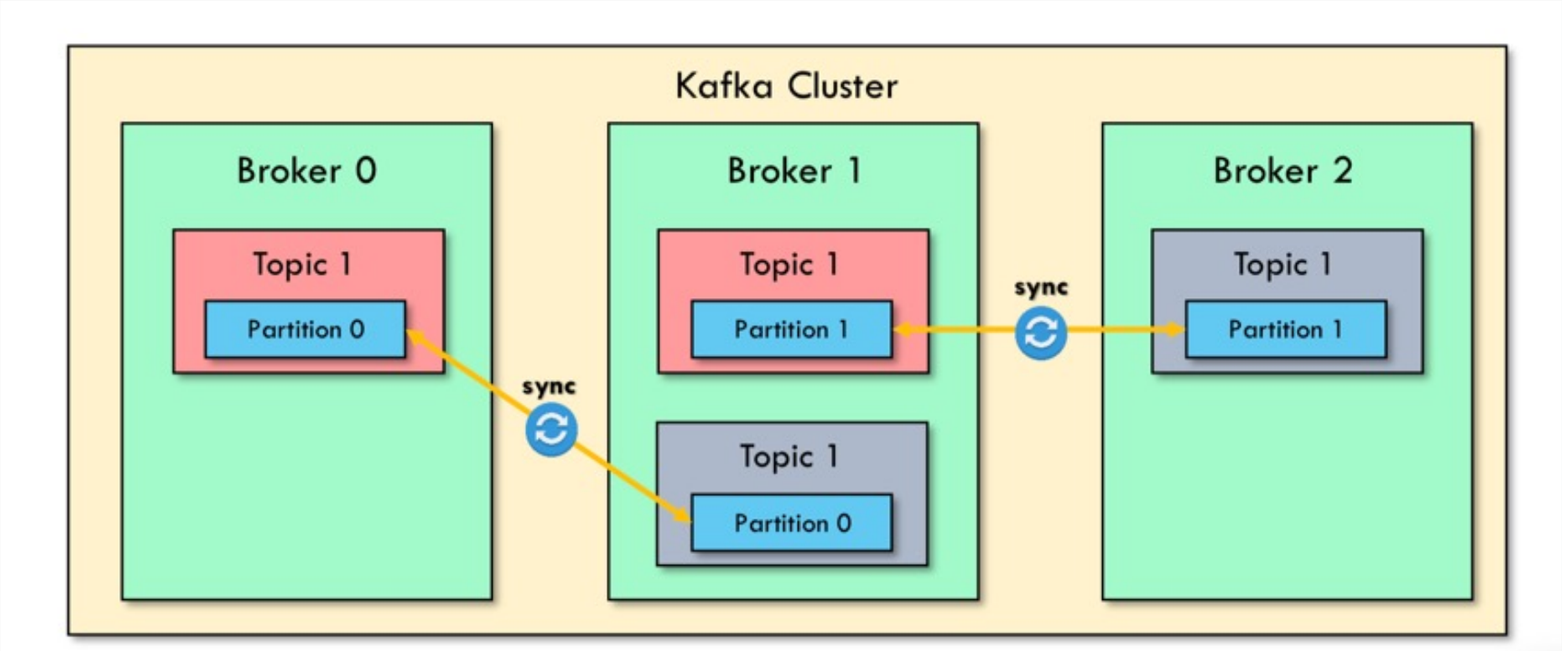
- A Kafka cluster is composed of multiple **brokers** (servers)
- Each broker is identified with its ID (integer)
- Each broker contains certain topic partitions
- After connecting to any broker (called a bootstrap broker), you will be connected to the entire cluster
- A good number to get started is **3** brokers, but some big clusters have over 100 brokers



aws

Topic replication factor

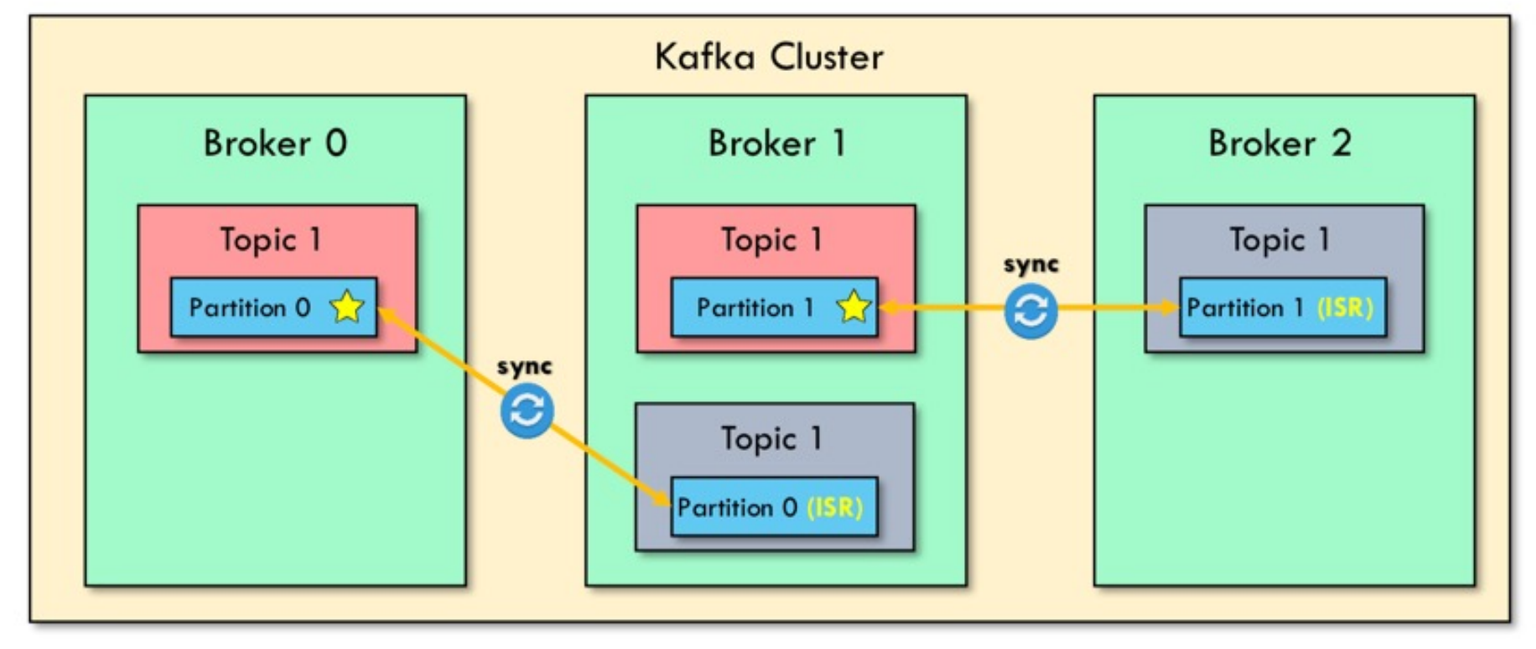
- Topics should have a replication factor > 1 (usually between 2 and 3)
- This way if a broker is down, another broker can serve the data
- Example: 1 **topic** with 2 **partitions** and **replication factor** of 2



aws

Concept of Leader for a partition

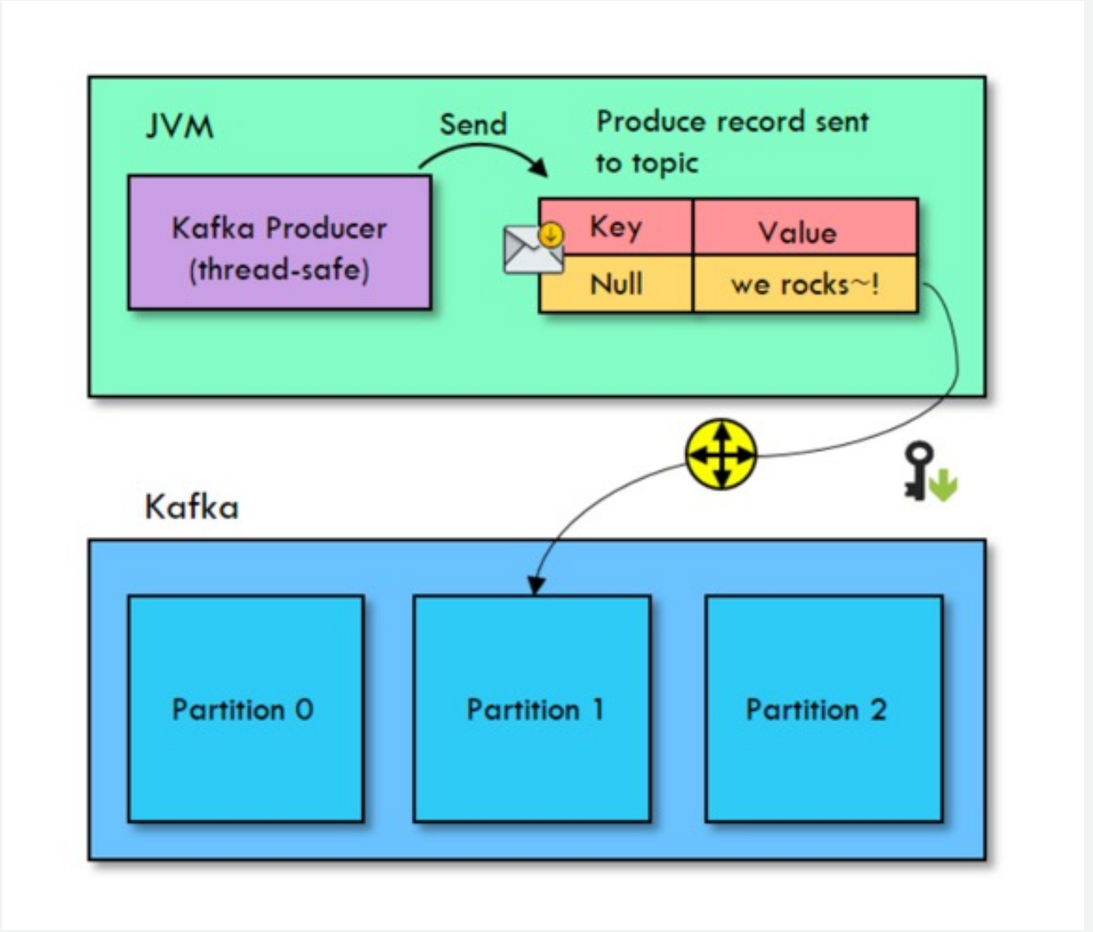
- At any time only 1 broker can be a leader for a given partition
- Only that leader can retrieve and serve data for a partition
- The other brokers will synchronize the data
- There each partition has: one **leader**, and multiple **ISR** (in-sync replica)



aws

Producers

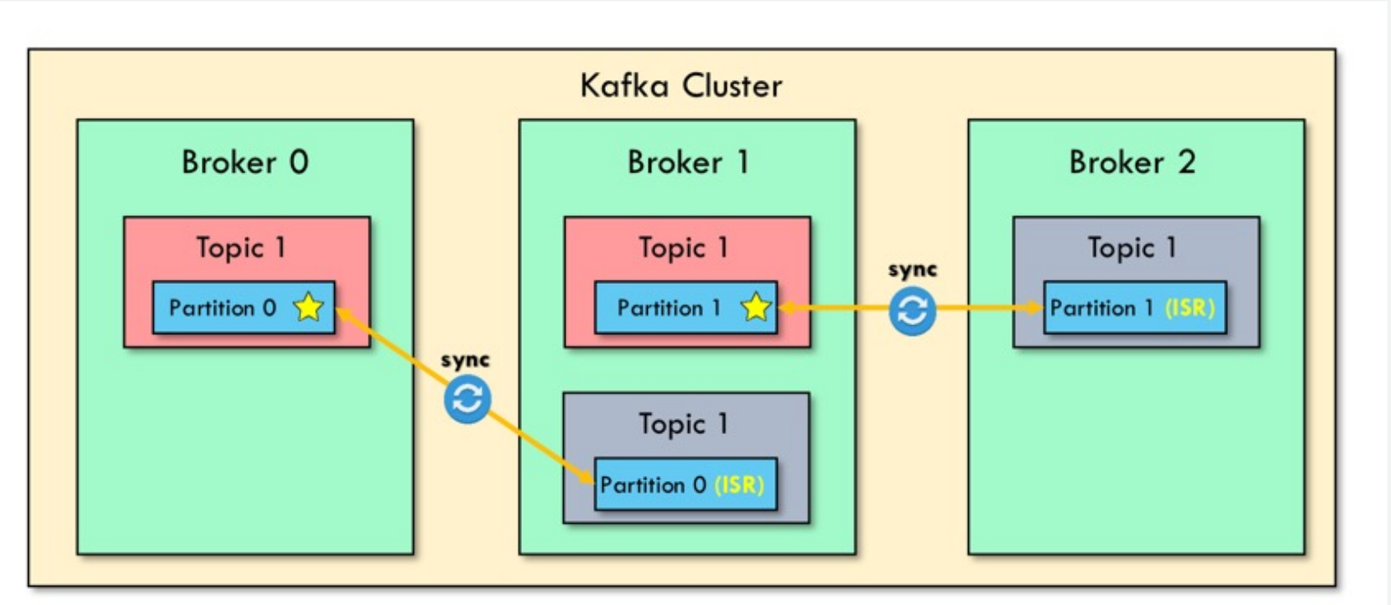
- **Producers** write data to **topics**.
- They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of routing the data to the right brokers.



aws

Producers: Broker acknowledgement

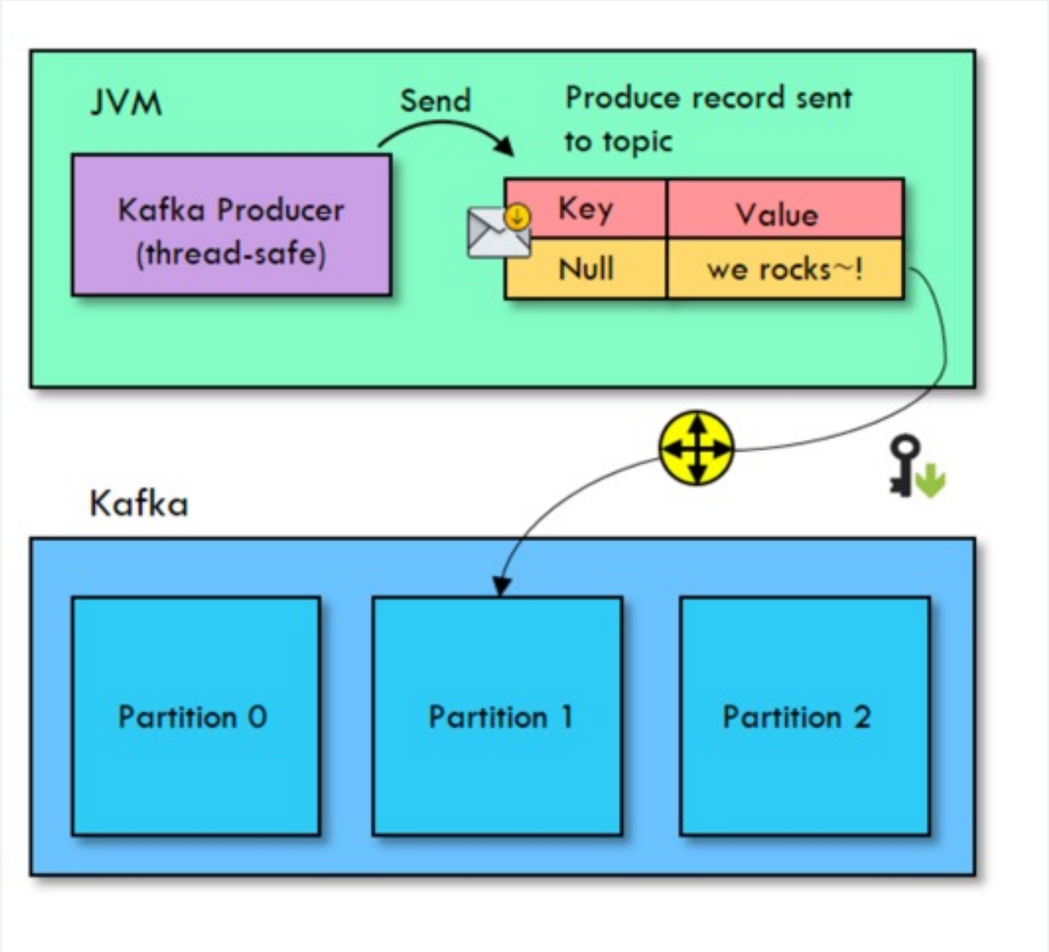
- Producer can choose to receive acknowledgement of data writes:
- **acks** = 0 : Producer won't wait for acknowledgement (possible data loss)
- **acks** = 1 : Producer will wait for leader acknowledgement (limited data loss)
- **acks** = all (-1) : Leader + replicas acknowledgment (no data loss)



aws

Producers: Message keys

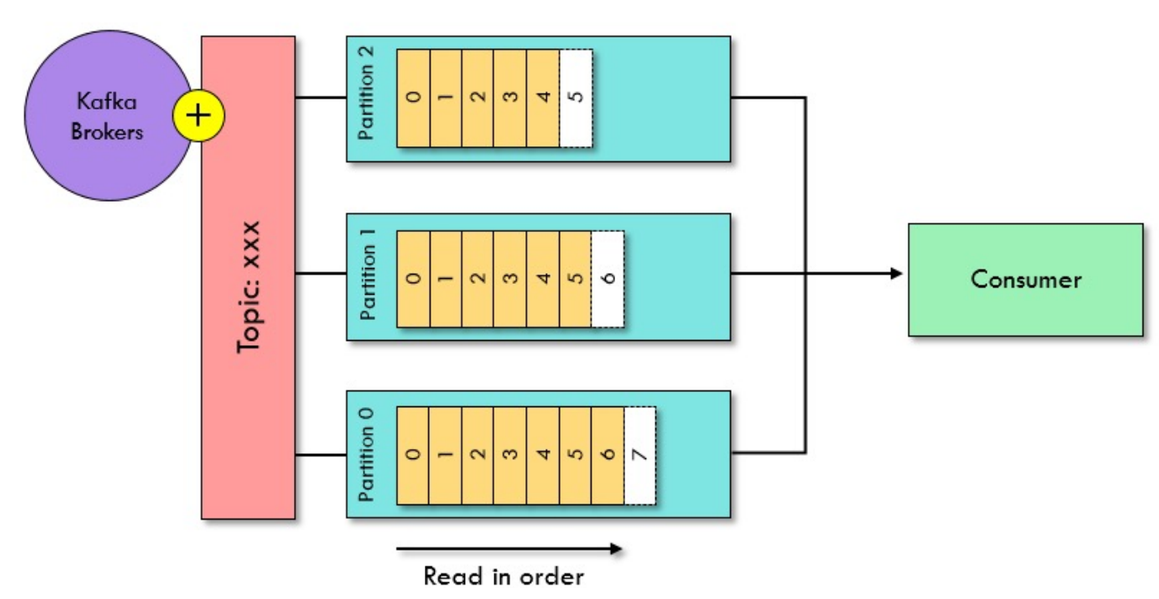
- Producers can choose to send a key with the message
- If a **key** is sent, then the producer has the guarantee that all messages for that key will always go to the same partition
- This enables to guarantee ordering for a specific **key**



aws

Consumers

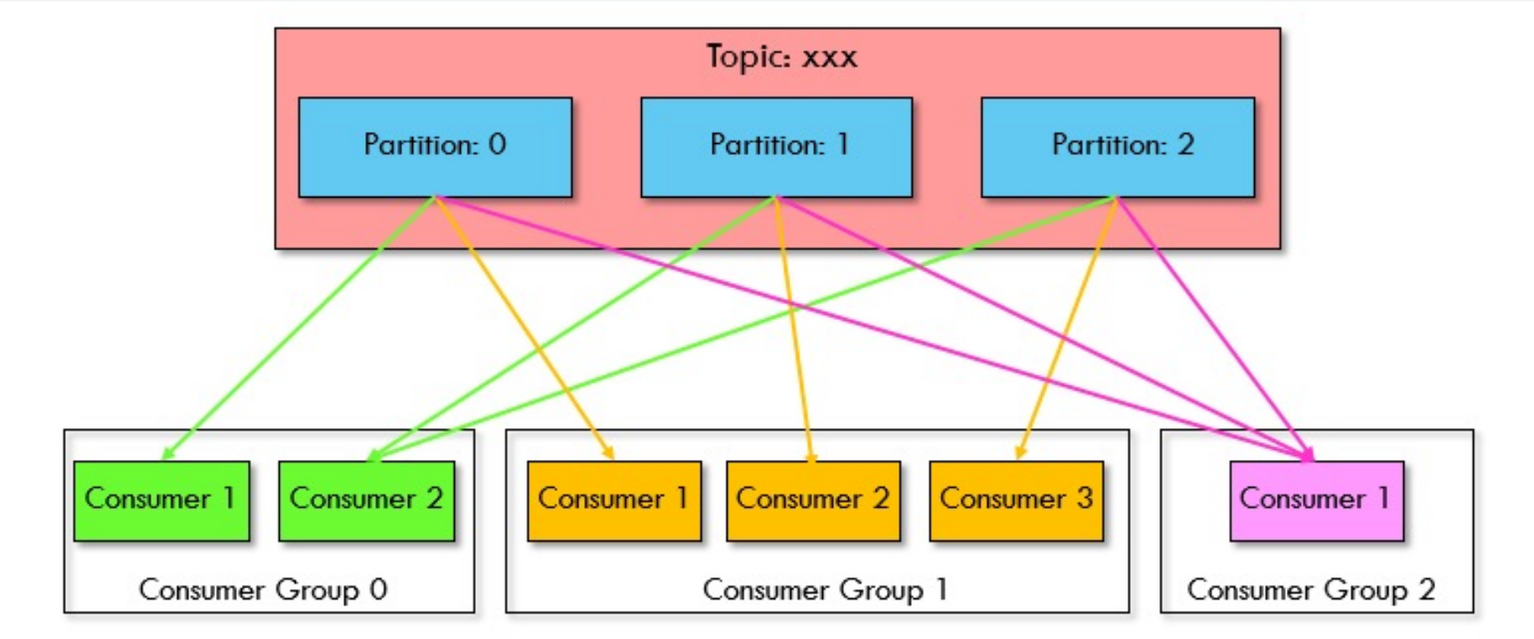
- **Consumers** read data from a topic
- They only have to specify the topic name and one broker to connect to, and Kafka will automatically take care of pulling the data from the right brokers
- Data is read in order **for each partitions**



aws

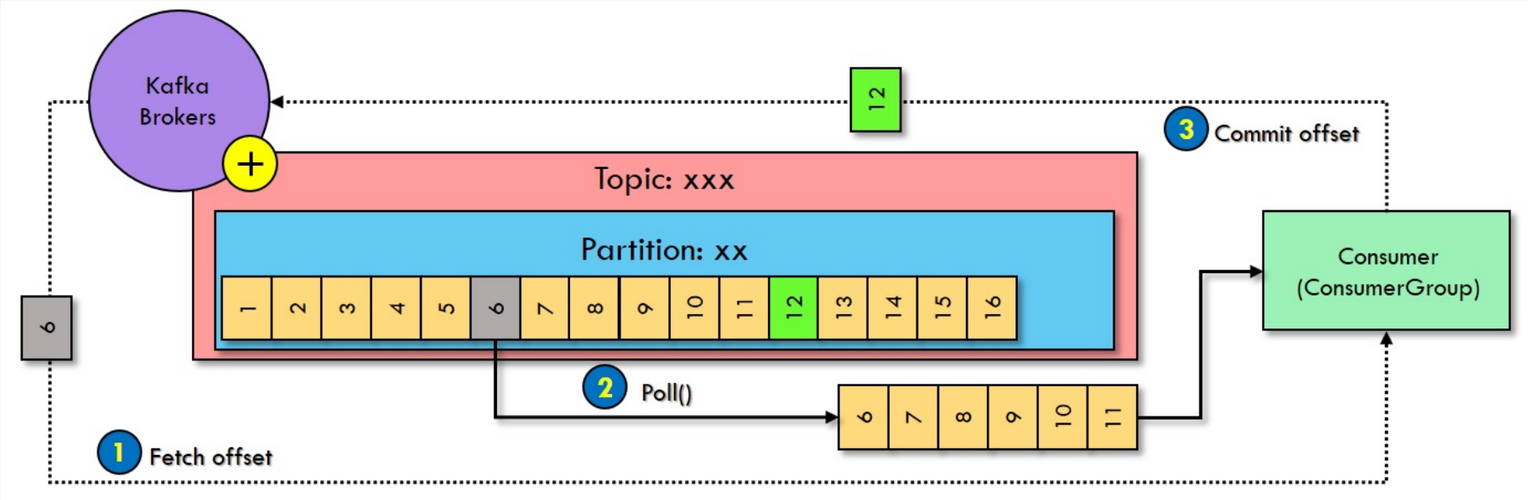
Consumer Groups

- Consumers read data in **consumer groups**
- Each consumer within a group reads from exclusive partitions
- You should control consumers instances less or equal than partitions (otherwise some will be inactive)



Consumer Offsets

- Kafka stores the offsets at which a **consumer group** has been reading
- The **offsets** commit live in a Kafka topic named "**__consumer_offsets**"
    - Key = [group, topic, partition] , Value=offset
- When a consumer has processed data received from Kafka, it should be **committing** the **offsets**
- If a consumer process dies, it will be able to read back from where it left off



aws

Kafka Guarantees

- Messages are appended to a **topic-partition** in the order they are sent
- Consumers read messages in the order stored in a **topic-partition**
- With a replication factor of N, producers and consumers can tolerate up to N-1 brokers being down
- This is why a replication factor of 3 is a good idea:
    - Allows for one broker to be taken down for maintenance
    - Allows for another broker to be taken down unexpectedly
- As long as the number of partitions remains constant for a topic (no new partitions), the same key will always go to the same partition

Log Cleanup Policies

- Many Kafka clusters make data expire, according to a policy
- That concept is called "**log cleanup**".
    - Policy 1: **log.cleanup.policy=delete** (Kafka default for all user topics)
        - Delete based on age of data (default is a week)
        - Delete based on max size of log (default is -1 == infinite)
    - Policy 2: **log.cleanup.policy=compact** (Kafka default for topic **__consume_offsets**)
        - Delete based on keys of your messages
        - Will delete old duplicate keys **after** the active segment is committed
        - Infinite time and space retention

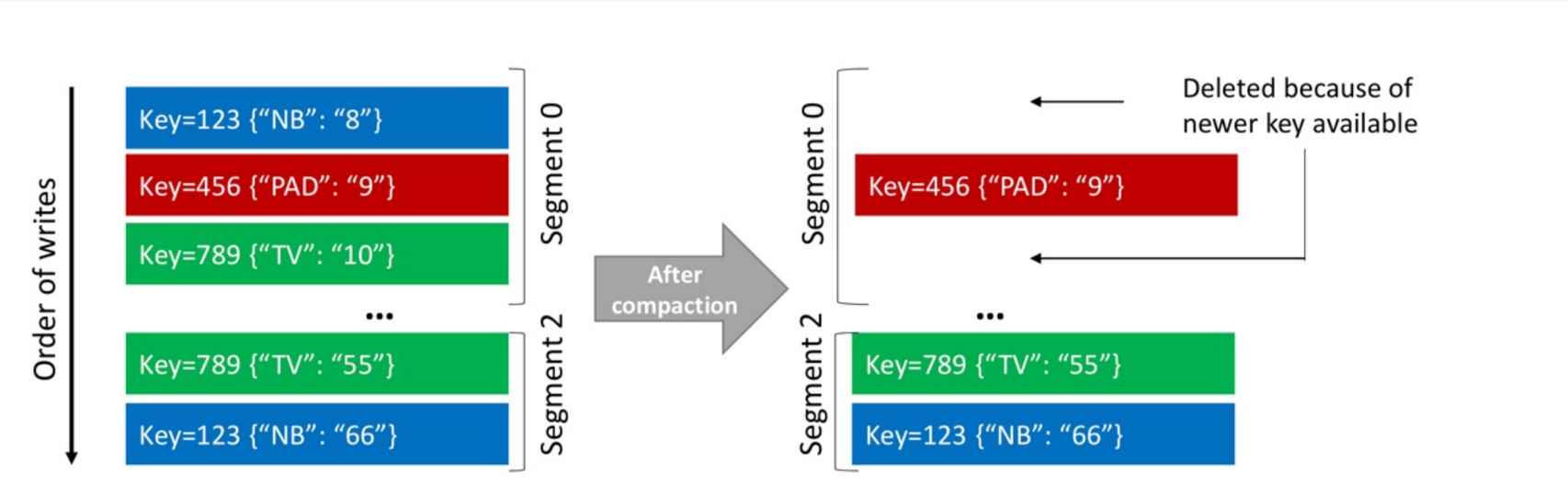
aws

Log Cleanup Policy: **log.cleanup.policy=delete**

- **log.retention.hours**:
    - Number of hours to keep data for (default is 168 – one week)
    - Higher number means more disk space
    - Lower number means that less data is retained (your consumers may need to replay more data than less)
- **log.retention.bytes**:
    - Max size in Bytes for each partition (default is -1 == infinite)
    - Useful to keep the size of a log under a threshold

Log Cleanup Policy: **log.cleanup.policy=compact**

- Log compaction ensures that your log contains *at least the last know value for a specific key within a partition*
- Very useful if we just require a SNAPSHOT instead of full history (such as for a data table in a database)
- The idea is that we only keep the latest “update” for a key in our log

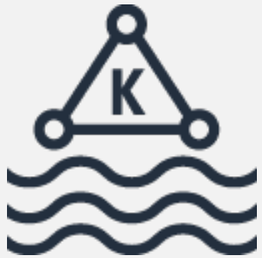


Other advanced configurations

- **max.messages.bytes** (default is **1MB**): if your messages get bigger than 1MB, increase this parameter on the topic and your consumers buffer
- **min.isync.replicas** (default is **1**): if using acks=all, specify how many brokers need to acknowledge the write
- **unclean.leader.election** (danger zone! – default **false**): if set to true, it will allow replicas which are not in sync to become leader as a last resort if all ISRs are offline. This can lead to data loss. If set to false, the topic will go offline until the ISRs come back up

Further thinking

- Topics are made of partitions , and the partitions are made of …??
- How kafka knows where to find data in a constant time or offset?
- Producer Synchronous and Asynchronous send.
- What is the consumer group rebalance? Describe the mechanism of rebalance process.



Amazon Managed Streaming for Kafka

Comparing Amazon MSK with Amazon Kinesis Data Streams

aws

Comparing Amazon Kinesis Data Streams to MSK

Amazon Kinesis Data Streams

- Streams and shards
- AWS API experience
- Throughput provisioning model
- Seamless scaling
- Typically lower costs
- Deep AWS integrations

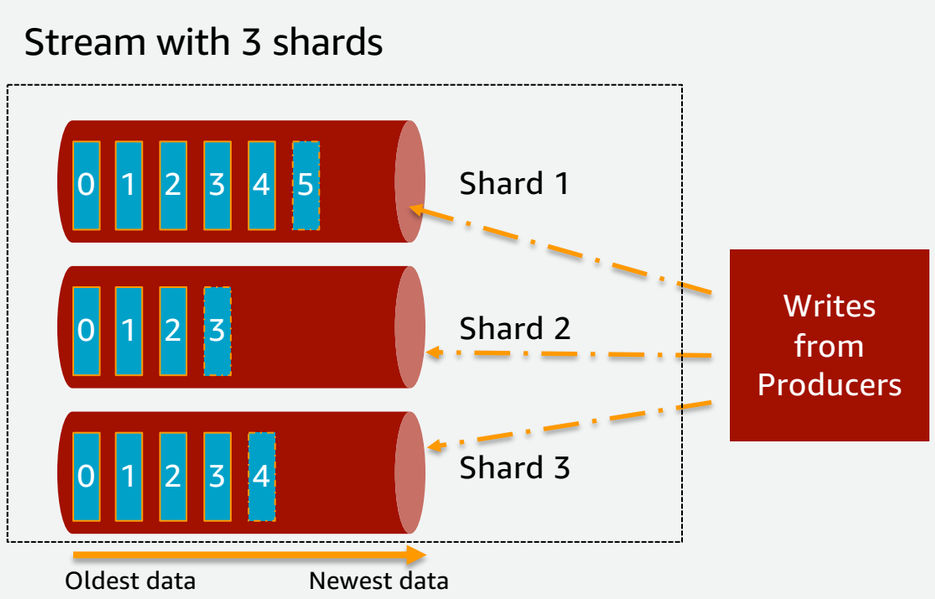
Amazon MSK

- Topics and partitions
- Open-source compatibility
- Strong third-party tooling
- Cluster provisioning model
- Apache Kafka scaling isn’t seamless to clients
- Raw performance

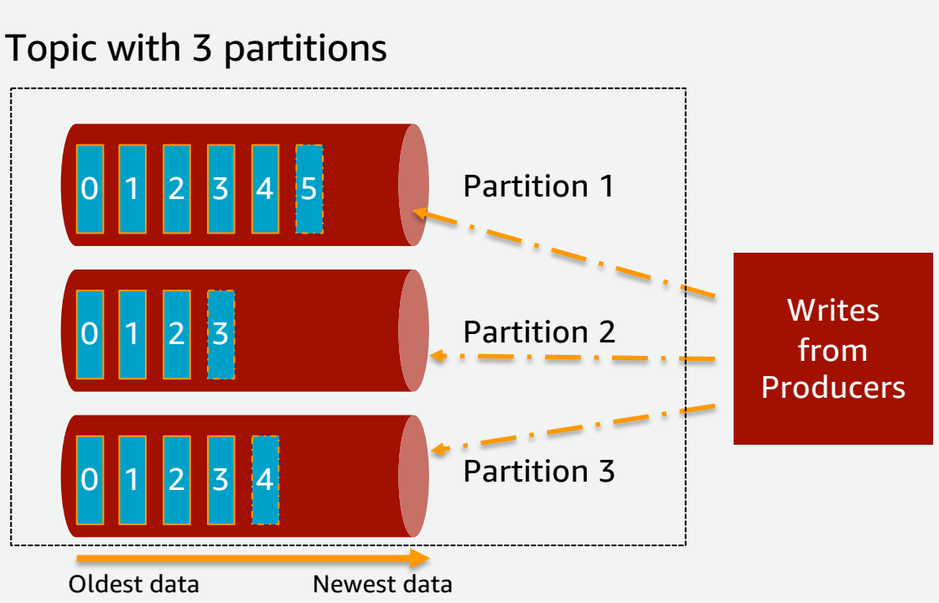
aws

Comparing Amazon Kinesis Data Streams to MSK

Amazon Kinesis Data Streams



Amazon MSK



Online-testing

- https://github.com/zhwenhao-amzn/ELS-MSK-Kinesis/tree/main/streaming-quiz

- Mail the result to zhwenhao@amazon.com

aws

Thank you!