Introduction to Streaming & Messaging AWS Big Data Demystified

Flume ,Kafka,SQS,Kinesis streams & firehose & Analytics

Omid Vahdaty, Big Data Ninja

What is batch Processing?

the execution of a series of programs each on a set or "**batch**" of inputs, rather than a single input (which would instead be a custom job

What is Streaming ?

Streaming Data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes)

Streaming VS. Batch Processing

	Batch	Stream
Data Scope	Query the entire batch, with slight delay	Query most recent events defined in a time window.
Data Size	Large data sets	A few Individual records
Latency?	Minutes ,hours	Seconds, Milliseconds
Analysis	Complex Analytics	Basic: aggregations, metrics etc.

Streaming vs messaging

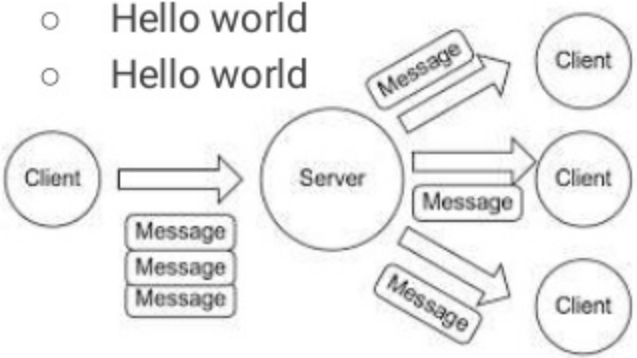
- Stream - when you need to do complex analytics in flight - e.g vote application. it's about processing infinite input stream (in contrast to batch processing that is applied to finite inputs).
- Message - when you need to do per event - an operation. - e.g log
- https://stackoverflow.com/questions/41744506/difference-between-stream-processing-and-message-processing

Challenges with Streaming Data

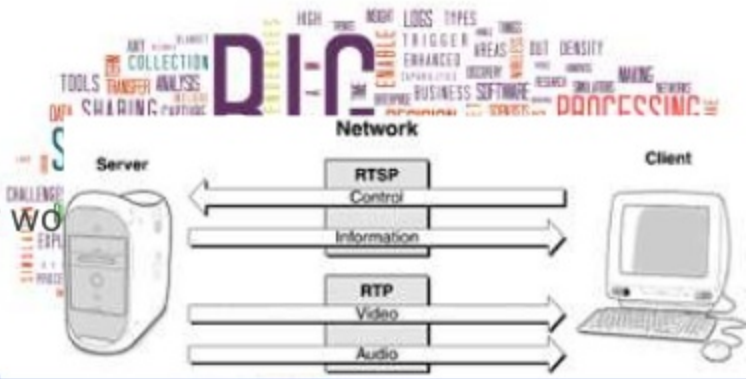
- Processing layer
  - Consuming data
  - Processing data
  - Notifying storage layer what to do.
- Storage layer
  - Ordering mechanism
  - Strong Consistency mechanism
- In general MUST have features:
  - scalability
  - data durability
  - fault tolerance

Messaging VS Streaming?

- Messaging: **framed message** based protocol.
- E.g 3 messages sent will look like:
  - Hello world
  - Hello world
  - Hello world



- Streaming: **unframed data** (bytes) stream based protocol
- E.g 3 messages sent will look like:
  - Hell
  - ow wo
  - rld Hel
  - low wor
  - ldHellow wo
  - rld



Messaging

Open Source: Kafka, flume
AWS: SQS

Flume

Flume Pros:

- Good documentation with many existing implementation patterns to follow
- Easy integration with existing monitoring framework
- Integration with Cloudera Manager to monitor Flume

Flume

Flume Cons:

- Event rather that stream centric
- Calculating capacity is not an exact science but rather confirmed through trials
- Throughput is dependent on the channel backing store.
- Flume lacks the clear scaling and resiliency configurations (trivial with Kafka and Kinesis)

Kafka

Kafka Pros:

- High achievable ingest rates with clear scaling pattern
- High resiliency via distributed replicas with little impact on throughput

Kafka Cons:

- No current framework for monitoring and configuring producers

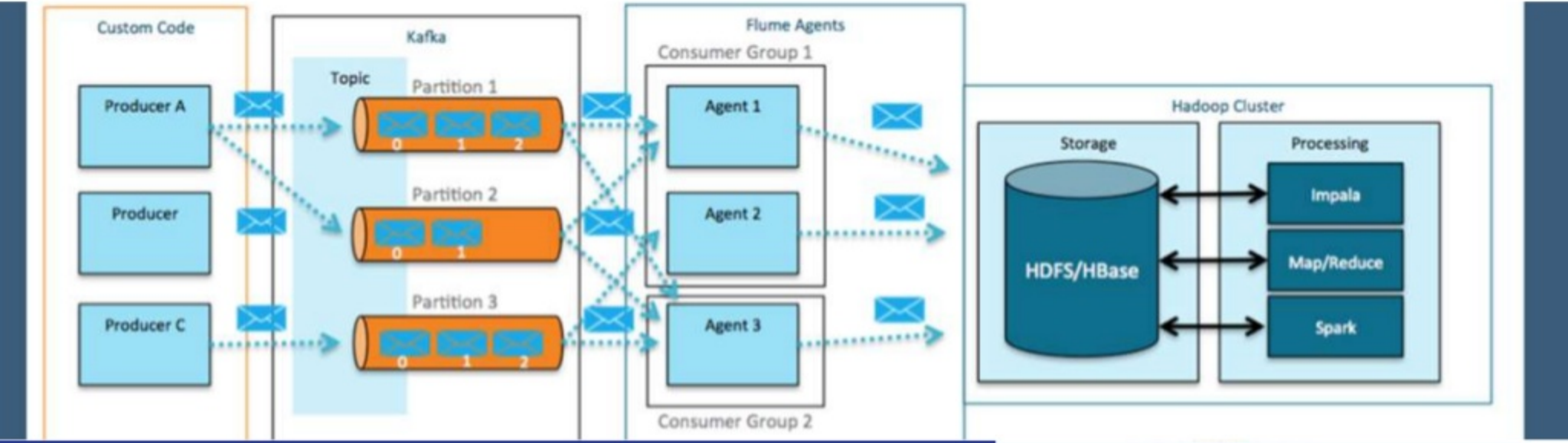
Flume VS. Kafka

	Flume	Kafka
Choose when you desire	No need for customization. Need out of the box components such HDFS sink	Need a custom made high availability delivery system
Velocity	high	higher
Event processing		

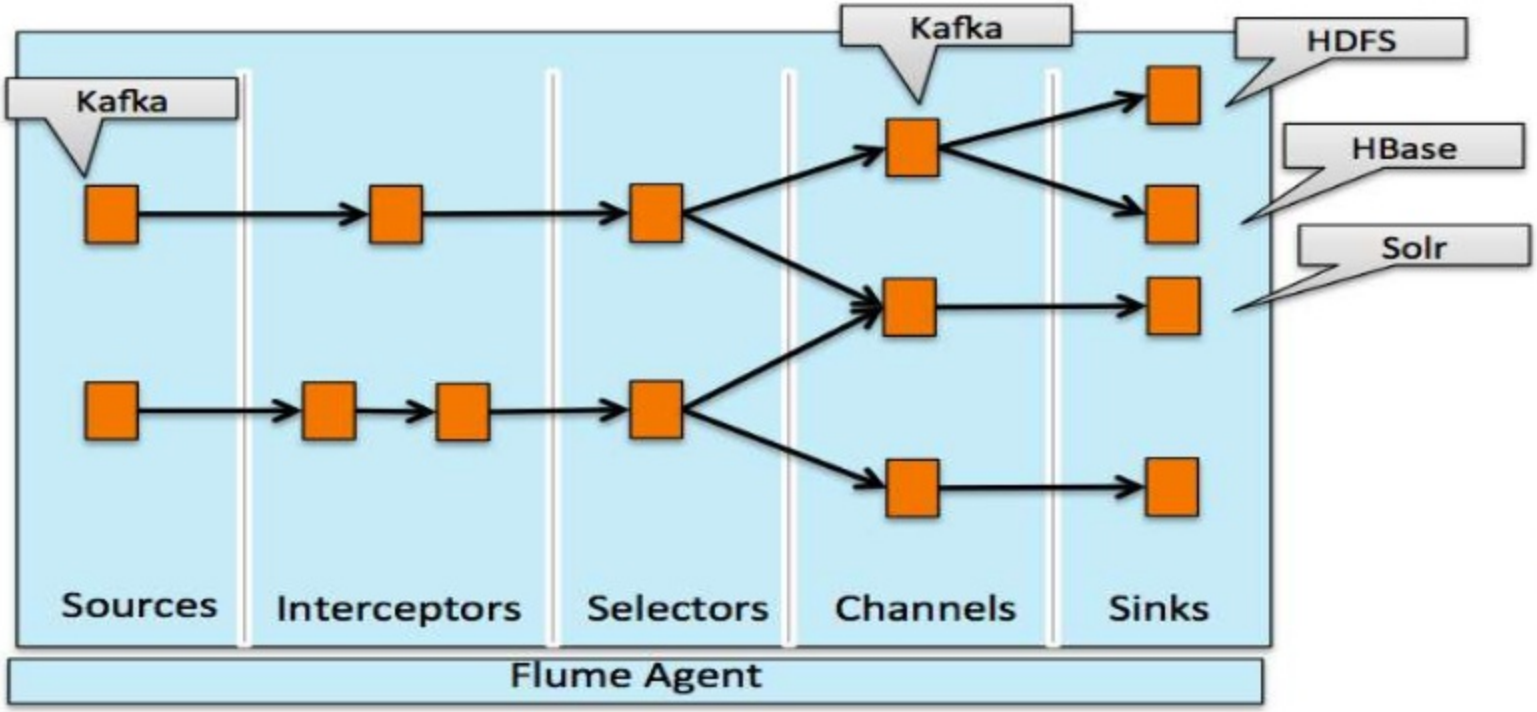
	Flume	Kafka
Original Motivation	distributed, reliable, and available system for efficiently collecting, aggregating and moving large amounts of log data from many different sources to a centralized data store. Built around hadoop ecosystem	general purpose distributed publish-subscribe messaging system Multi-consumer ultra-high availability messaging system.
Data Flow	push	pull
event availability	JDBC Databases Channel, file Channel. Loose flume agent = losing data.	replication of your events data by design.
Commercial support	Cloudera	Cloudera
Collectors built in	Yes.	just the messaging

Use Case: Kafka and Flume combined

- Flume supports: Kafka source, Kafka channel, Kafka sink
- So, take the advantage of both and combine them to your needs.



Use Case: Kafka as a Channel



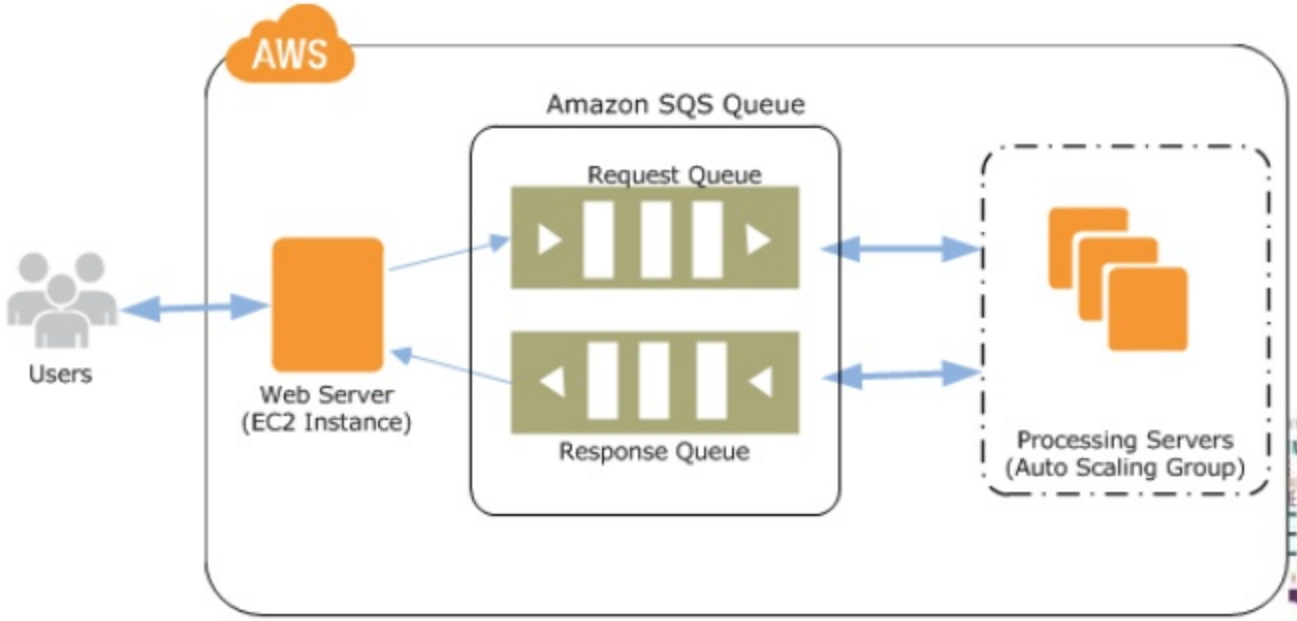
AWS SQS

- a fast, reliable, scalable, fully managed **message queuing** service
- **decouple the components** of a cloud application, move data between diverse, distributed application components without losing messages and without requiring each component to be always available.
- high throughput and **at-least-once processing**, and ***FIFO queues***
- all messages are stored **redundantly** across multiple servers and data centers.
- Start with t**hree API calls : SendMessage, ReceiveMessage, and DeleteMessage.** Additional APIs are available to provide advanced functionality.
- **Queues**
  - **Standard queues** offer **maximum throughput**, best-effort ordering, and at-least-once delivery.
  - **FIFO queues** are designed to ensure **strict ordering** and exactly-once processing, with limited throughput.
- scales dynamically
- Authentication mechanisms

AWS SQS use cases

- Messaging semantics (such as **message-level ack/fail**) and visibility timeout. For example, you have a queue of work items and want to track the successful completion of each item independently. Amazon SQS tracks the ack/fail, so the application does not have to maintain a persistent checkpoint/cursor. Amazon SQS will delete acked messages and redeliver failed messages after a configured visibility timeout.
- **Individual message delay**. For example, you have a job queue and need to schedule individual jobs with a delay. With Amazon SQS, you can configure individual messages to have a delay of up to 15 minutes.
- **Dynamically increasing concurrency/throughput at read time.** For example, you have a work queue and want to add more readers until the backlog is cleared. With Amazon Kinesis, you can scale up to a sufficient number of shards (note, however, that you'll need to provision enough shards ahead of time).
- **Leveraging Amazon SQS's ability to scale transparently.** For example, you buffer requests and the load changes as a result of occasional load spikes or the natural growth of your business. Because each buffered request can be processed independently.

Typical SQL use case: decoupling APP layers.



Streaming

AWS Kinetics:
Streams,Firehose,Analytics

AWS Kinesis (streams)

- build custom applications that process or analyze streams
- continuously capture and store terabytes of data per hour
- Hundreds sources
- allows for real-time data processing
- Easy to use, get started in minutes
  - Kinesis Client Library
  - Kinesis Producer Library
- allows you to have multiple Applications processing the same stream concurrently.
- The throughput can scale from megabytes to terabytes per hour
- synchronously replicates your streaming data across three AZ
- preserves your data for up to 7 days

AWS Kinesis (streams) use cases

- Log and Event collection
- Mobile Data collection
- Real Time Analytics
  - when loading data from transactional databases into data warehouses.
  - Multi-stage processing using specialized algorithms
  - stream partitioning for finer control over scaling
- Gaming Data feed

AWS Kinesis (streams) use cases

- Routing related records to the same record processor (as in streaming MapReduce). For example, **counting and aggregation are simpler when all records for a given key are routed to the same record processor.**
- Ordering of records. For example, you want to **transfer log data** from the application host to the processing/archival host while maintaining the order of log statements.
- Ability for multiple applications to **consume the same stream concurrently**. For example, you have one application that updates a real-time dashboard and another that archives data to Amazon Redshift. You want both applications to consume data from the same stream concurrently and independently.
- Ability to **consume records in the same order a few hours later.** For example, you have a billing application and an audit application that runs a few hours behind the billing application. Because Amazon Kinesis stores data for up to 24 hours, you can run the audit application up to 24 hours behind the billing application.

AWS Kinesis (streams)

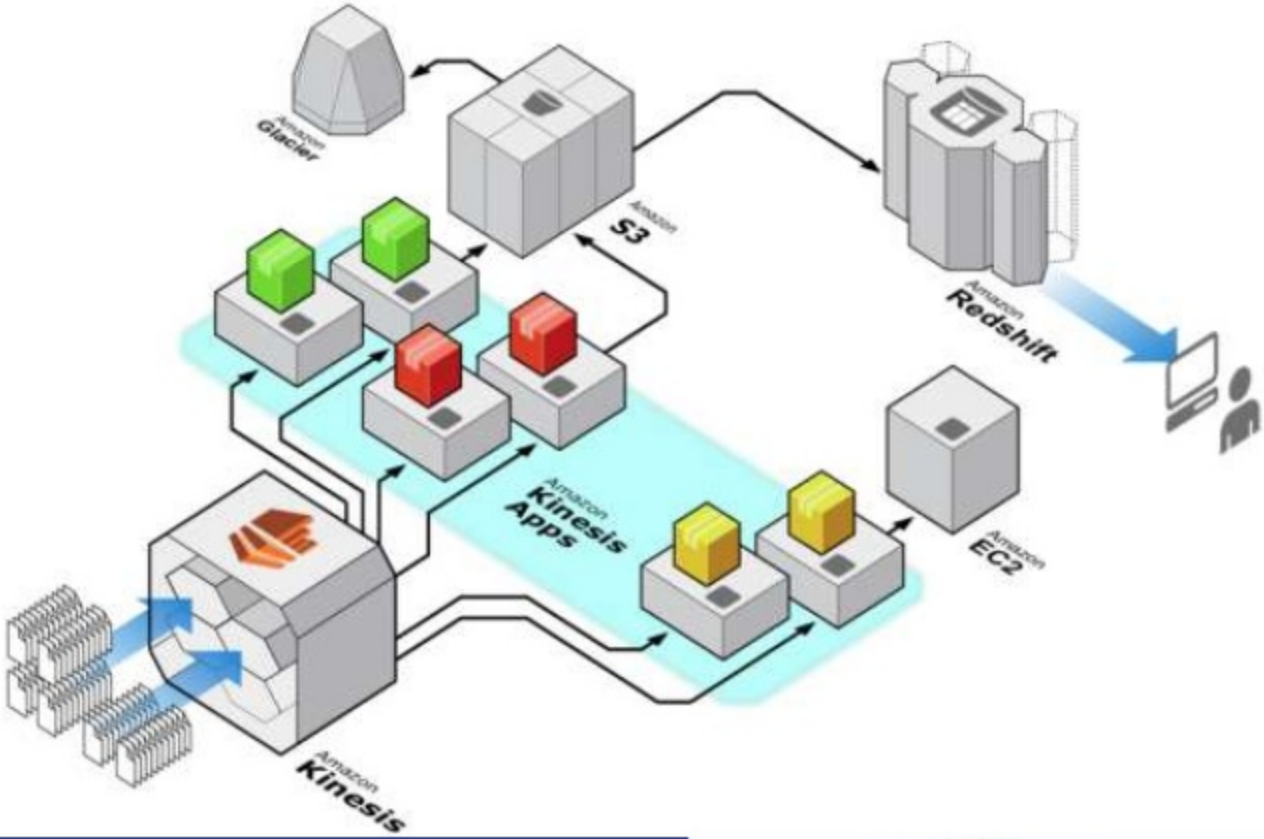
Kinesis Pros:

- High achievable ingest rates with clear scaling pattern
- Similar throughput and resiliency characteristics to Kafka
- Integrates with other AWS services like EMR and Data Pipeline.

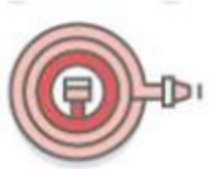
Kinesis Cons:

- No current framework for monitoring and configuring producers
- Cloud service only. Possible increase in latency from source to Kinesis.

AWS Kinesis (streams)



AWS Kinesis Firehose

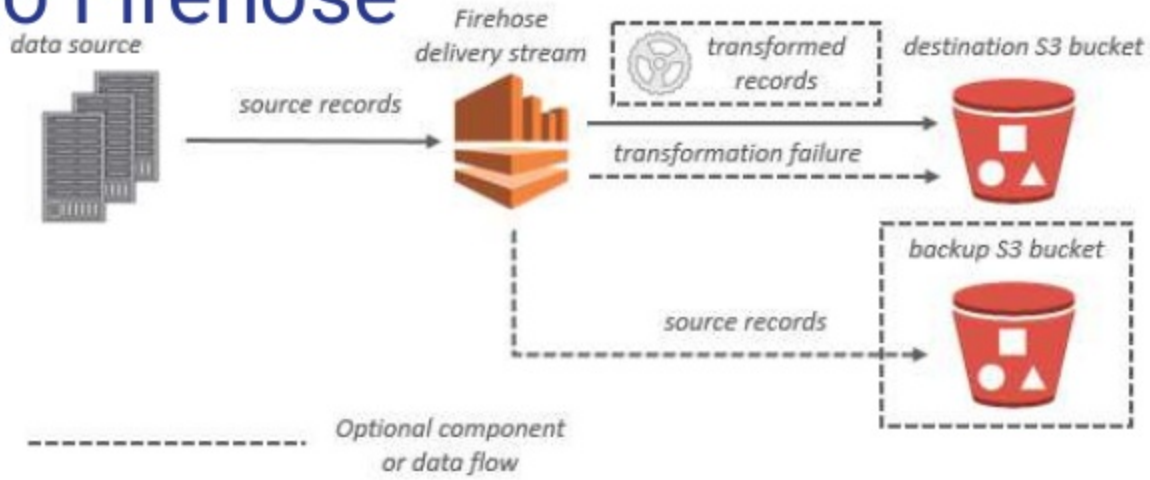


- the easiest way to load streaming data into AWS.
- **capture, transform, and load streaming data**
    - integrates into Kinesis Analytics, S3, Redshift, Elasticsearch Service
    - Serverless Transformation on RAW data. (lambda function)
        - E.g transform log file into CSV format
- Firehose can back up all untransformed records to your S3 bucket concurrently while delivering transformed records to the destination. You can enable source record backup
- enabling near real-time analytics
- Easy to use.
- Monitoring options.
- Limits
    - 20 stream per regions
    - Each stream
        - 2000 transaction per sec
        - 5000 records per sec
        - 5MB/s
        - Support 24 hours replay in cases on downtime

Kinesis Firehose agent

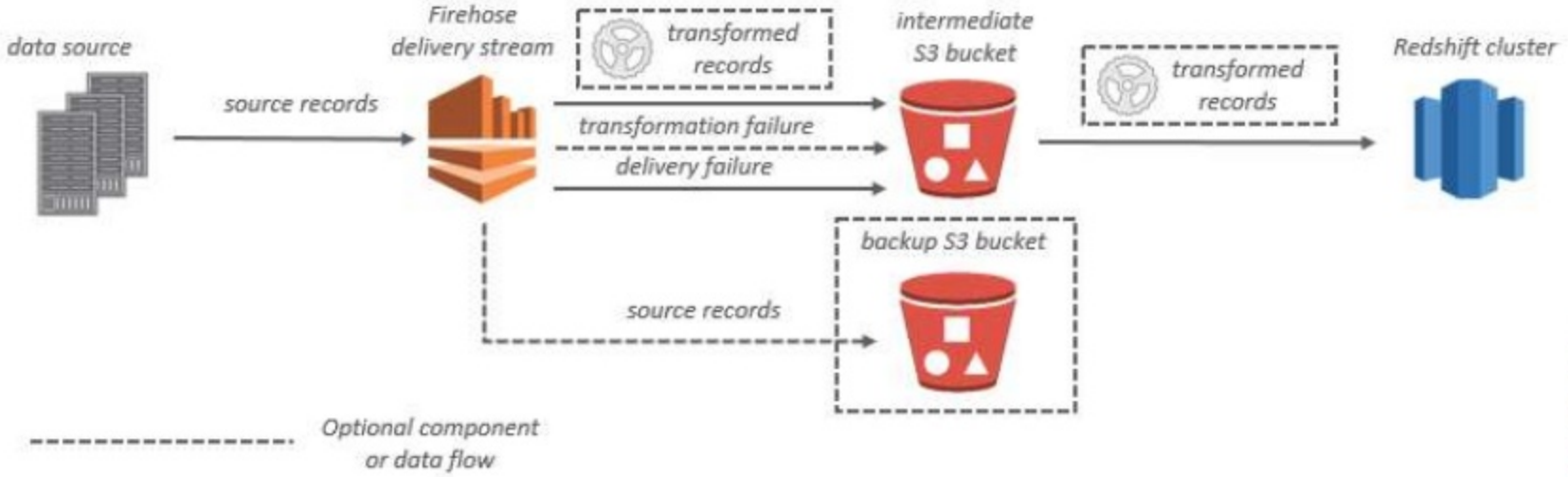
- Java software app that send data to streams/firehose
- monitors a set of files for new data and then sends streams/firehose
- It handles file rotation, checkpointing, and retrial upon failures.
- supports Amazon CloudWatch so that you can closely monitor and troubleshoot the data flow from the agent.
- Data processing options:
  - SINGLELINE – This option converts a multi-line record to a single line record by removing newline characters, and leading and trailing spaces.
  - CSVTOJSON – This option converts a record from delimiter separated format to JSON format.
  - LOGTOJSON – This option converts a record from several commonly used log formats to JSON format. Currently supported log formats are Apache Common Log, Apache Combined Log, Apache Error Log, and RFC3164 (syslog).
- https://github.com/awslabs/amazon-kinesis-agent
- Amazon Kinesis Firehose will only output to Amazon S3 buckets and Amazon Redshift clusters in the same region.

Write a JAVA agent to Firehose



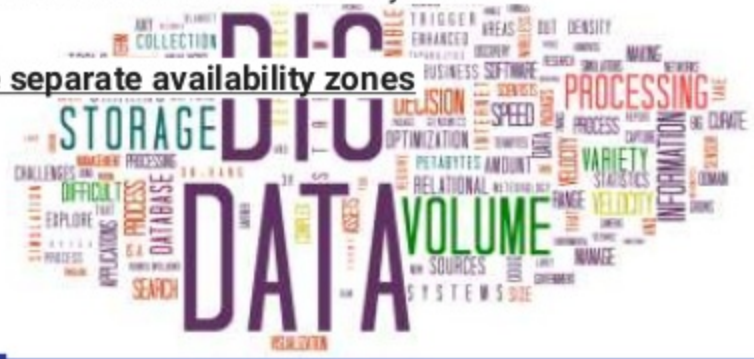
- AWS java SDK
- Firehose API
  - Single record: PutRecord
  - Batch: PutRecordBatch.
- Key concepts:
  - **Firehose delivery stream**
  - **Data producer - i.e web server creating log.**
  - **Record:** The data of interest that your data producer sends to a Firehose delivery stream. A record can be as large as 1000 KB.
  - **buffer size** (in MB)
  - **buffer interval** (seconds)
- **Java examples:**
  - http://docs.aws.amazon.com/firehose/latest/dev/writing-with-sdk.html

Firehose and redshift use case



Streams VS. Firehose

- Data producers: logs , web, mobile
- Data consumers: EMR , S3, redshift
- Stream delivery : streams & firehose
- Key concepts for **Streams**:
    - Basic unit: Shard
        - 1MB ingerss, 2MB egress per shard
        - 10 shards? X10 performance.
        - default limit of 10 shards per region
        - no limit to the number of shards or streams in an account.
        - *Partition keys* are used to identify different shards in a stream
        - *Sequence numbers* are unique identifiers for records inserted into a shard. They increase monotonically, and are specific to individual shards.
        - **Streaming data is replicated by Kinesis across three separate availability zones**
        - data is available in a stream for 24 hours
    - Streams API to control scale. Upt TBs per hours
    - Monitoring is available through Amazon Cloudwatch.



Streams VS. Firehose

- Data producers: logs , web, mobile
- Data consumers: EMR , S3, redshift
- Stream delivery : streams & firehose
- Key concepts for Firehose:
    - can scale to **gigabytes of streaming data per second**
    - batching, encrypting and **compressing** of data
    - automatically scale to meet demand, which is in contrast to Kinesis Streams

Stream VS. Firehose

	Streams	Firehose
Purpose	real-time processing of streaming big data/. "**real time**" "**custom**"	real-time processing of streaming big data, It builds on the existing Kinesis framework "**Zero Administration**" "**Direct**" no need to write code.
Loading methods	HTTPS, the Kinesis Producer Library, the Kinesis Client Library, and the Kinesis Agent, Java SDK	HTTPS, the Kinesis Producer Library, the Kinesis Client Library, and the Kinesis Agent, Java SDK
Transform methods		Encryption, compression, Lambda

Stream VS. Firehose

	Streams	Firehose
Target	S3.redshift, DynamoDB, elasticsearch, Apache Storm, kibana	S3.redshift
Replay	Default 24 hours, up to 7 days, data replication to 3 AZ automatically.	
Monitoring	Cloud Watch	Cloud Watch
Scaling	manual	Automatic

Kinesis Streams VS. SQS

	Kinesis streams	SQS
Purpose	real-time processing of streaming big data	message queue to store messages transmitted between distributed application components.
	routing of records using a given key, ordering of records, the ability for multiple clients to read messages from the same stream concurrently,	messaging semantics so that your application can track the successful completion of work items in a queue
Scale	manual	Auto
redundancy	3 AZ by default, replay of messages up to 7 days	

Kinesis Analytics : in-flight analytics.

- process streaming data in real time with standard SQL
- Amazon Kinesis Analytics enables you to create and run SQL queries on streaming data
- Easy 3 steps
  1. Configure Input stream (kinesis stream, kinesis firehose)
     a. Automatically created Schema
     b. Manually change schema if you like
  2. Write SQL query
  3. Configure output stream: s3, redshift, elastics search
- Elastic: scale up down
- Managed service
- Standard SQL

Kinesis Analytics : in-flight analytics.

Capture streaming data with Kinesis Streams or Kinesis Firehose

Run standard SQL queries against data streams

Kinesis Analytics can send processed data to analytics tools so you can create alerts and respond in real-time

Stay in touch...

- Omid Vahdaty
- +972-54-2384178
- https://amazon-aws-big-data-demystified.ninja/
- https://www.meetup.com/AWS-Big-Data-Demystified/
- https://www.facebook.com/groups/amazon.aws.big.data.demystified/