Working with Streaming Data

Streaming Data

What is streaming data?

- Streaming data is data that is emitted at high volume in a continuous, incremental manner with the goal of low-latency processing.
- It may be generated continuously through thousands of data sources typically in smaller sizes (order of Kilobytes).
- Examples:
  - Log files generated by customers using your mobile or web applications
  - E-commerce purchases
  - In-game player activity
  - Information from social networks
  - Stock market
  - Geospatial services
  - Telemetry from connected devices

Batch Processing vs Streaming

	Batch Processing	Streaming
Data scope	Queries or processing over all or most of the data in the dataset.	Queries or processing over data within a rolling time window, or on just the most recent data record.
Data size	Large batches of data.	Individual records or micro batches consisting of a few records.
Performance	Latencies in minutes to hours.	Requires latency in the order of seconds or milliseconds.
Analysis	Complex analytics.	Simple response functions, aggregates, and rolling metrics.

Streaming data characteristics

Streaming Data

High volume

Organizations that collect clickstream data about user behavior can often reach up to 2–3 TB every day

Nearly continuous

IoT devices like sensors collect data continuously and at regular intervals

Ordered

A chat application makes little sense with messages out of order

Time sensitive

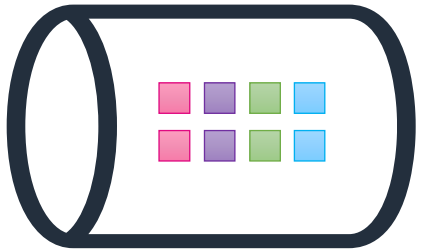
Fraud detection must be done as quickly as possible to prevent loss

Streaming components

Stream Producer

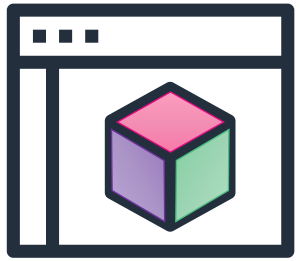
Puts records into streams. For example, a web server sending log data to a stream is a producer.

Stream Storage



Receives data from the producers and stores that data in the data stream for downstream consumers.

Stream Consumer



Gets records from streams and process them.

Stream Destination

Retain records to be further used by other applications.

Streaming Services on AWS

Streaming Services on AWS

Amazon Kinesis

Amazon Managed Service for Apache Flink

Amazon Kinesis Data Streams

Amazon Data Firehose
(successor to Kinesis Data Firehose)

Amazon Kinesis Video Streams

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

AWS Glue

Amazon Redshift

Amazon EC2

Amazon EMR

Streaming Services in AWS (Not an exhaustive list)

Stream Producer

- Amazon Kinesis Producer Library (KPL)
- AWS SDK
- Amazon Kinesis Agent
- Amazon MSK Connect
- Other AWS Services

Stream Storage

- Amazon Kinesis Data Streams
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Stream Consumer

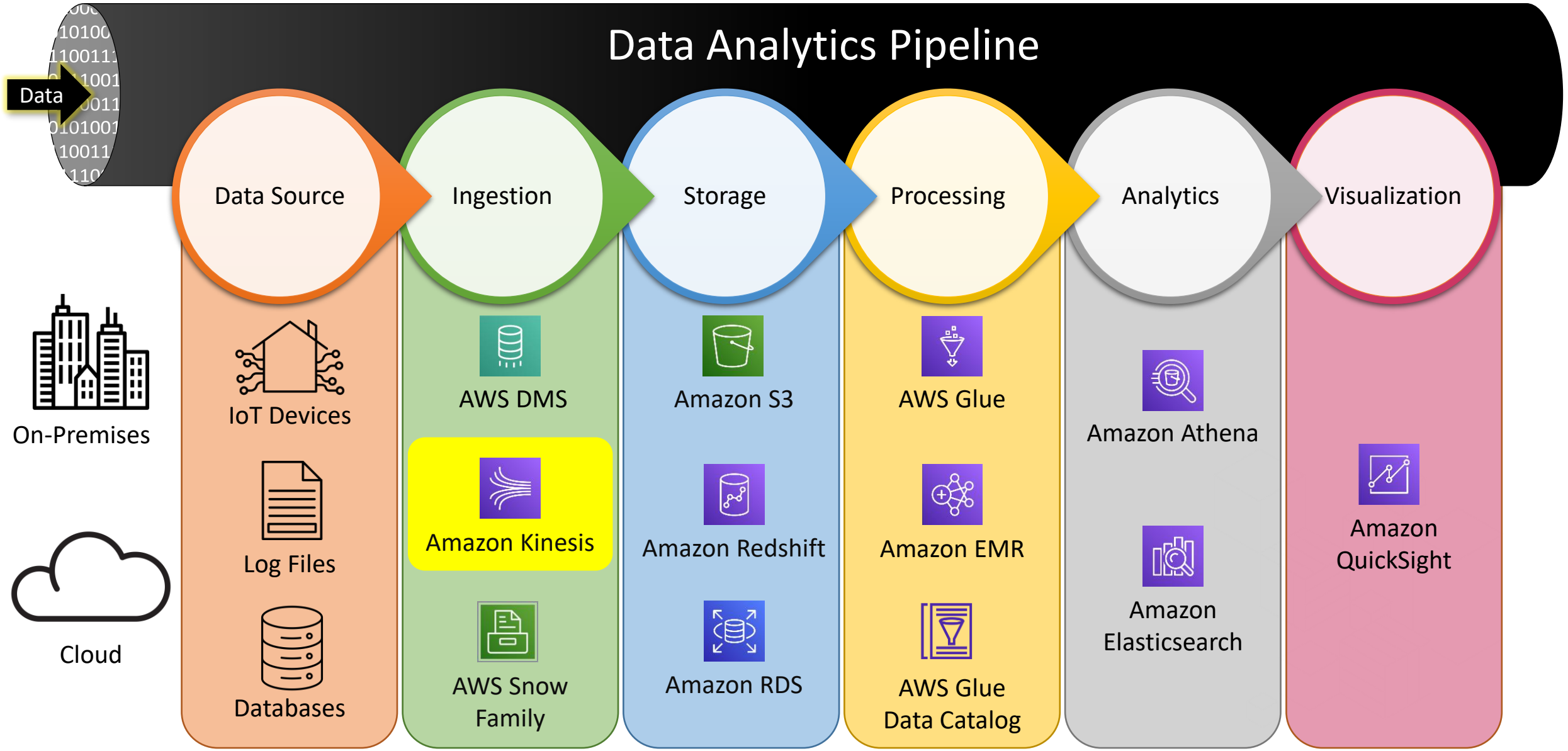
- Amazon Data Firehose
- Amazon Managed Service for Apache Flink
- AWS Glue
- Amazon EMR
- Custom Apps on EC2, Containers, Lambda using KCL

Stream Destination

- Amazon S3
- Amazon Redshift
- Amazon OpenSearch Service
- Amazon DynamoDB
- 3rd Party Services

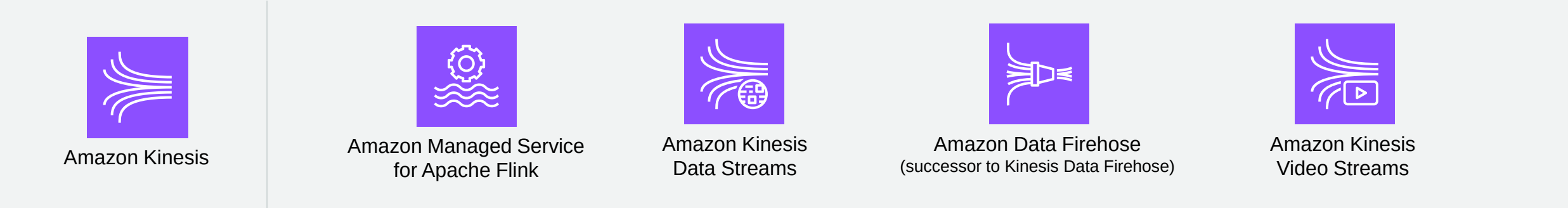
Amazon Kinesis

Data Analytics on AWS



Amazon Kinesis

- Kinesis is a platform for streaming data on AWS, to load and analyze streaming data.



- Naming Changes
  - Amazon Kinesis Data Firehose >> Amazon Data Firehose
  - Amazon Kinesis Data Analytics >> Amazon Managed Service for Apache Flink

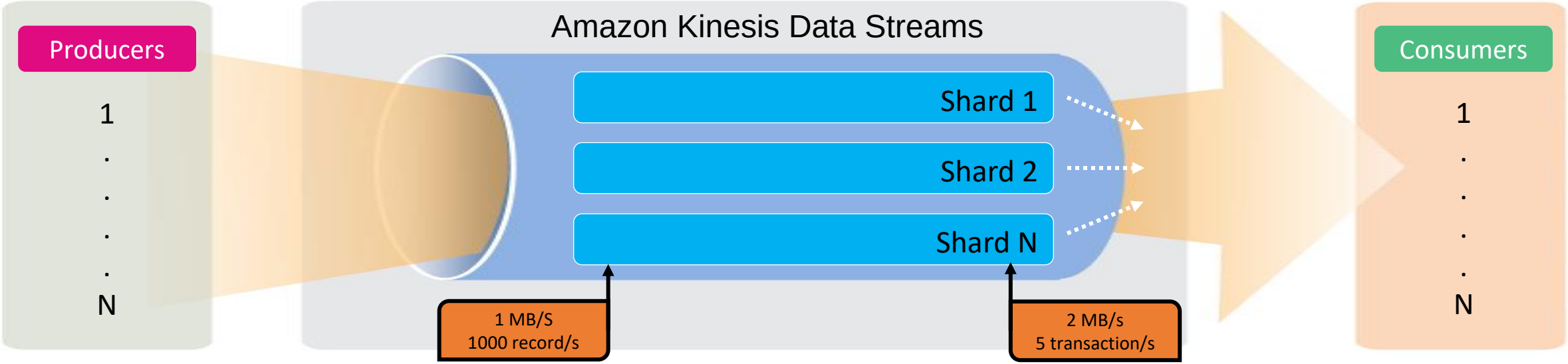
Amazon Kinesis Data Stream

Terminology

- Producer
  - Producers put records into Amazon Kinesis Data Streams. For example, a fleet of web server sending log data to a stream is a producer.
- Consumer
  - Consumers get records from Amazon Kinesis Data Streams and process them. Multiple consumers can consume data from the stream independently and concurrently.

Terminology

- Shard
  - A stream is composed of one or more shards, each of which provides a fixed unit of capacity. A shard is a uniquely identified sequence of data records in a stream.
  - Each shard can support up to 1,000 records per second for writes, up to a maximum total data write rate of 1 MB per second and can support up to 5 transactions per second for reads, up to a maximum total data read rate of 2 MB per second.
  - The total capacity of the stream is the sum of the capacities of its shards. If your data rate increases, you can increase or decrease the number of shards allocated to your stream (manually or automatically).

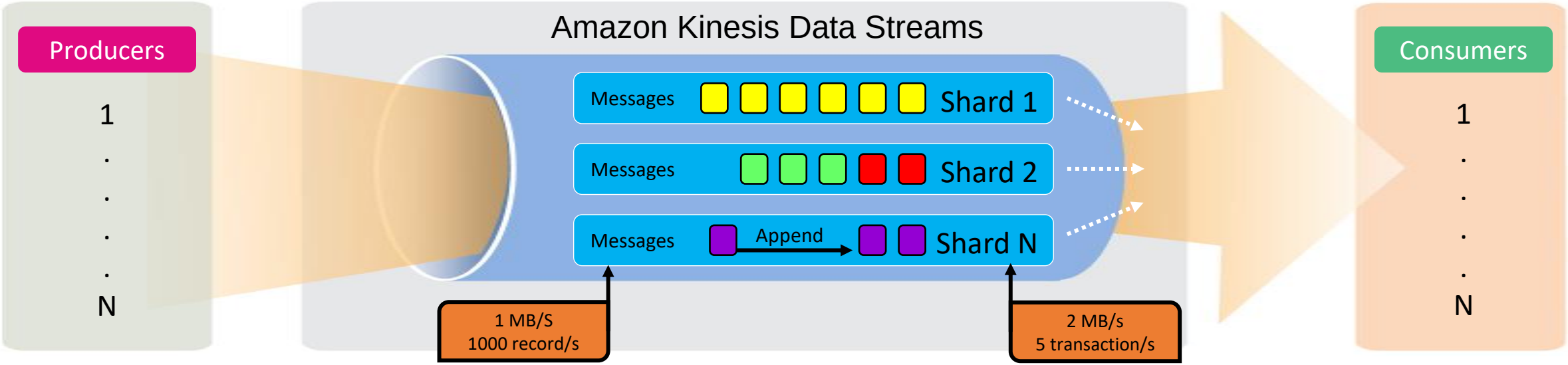


Terminology

- Record / Message

Partition Key	Seq. #	Data Blob

  - A record is the unit of data stored in an Amazon Kinesis data stream.
  - A record is composed of a sequence number, partition key, and data blob.
  - Data blob is the data of interest your data producer adds to a data stream.
  - The maximum size of a data blob is 1 megabyte (MiB).

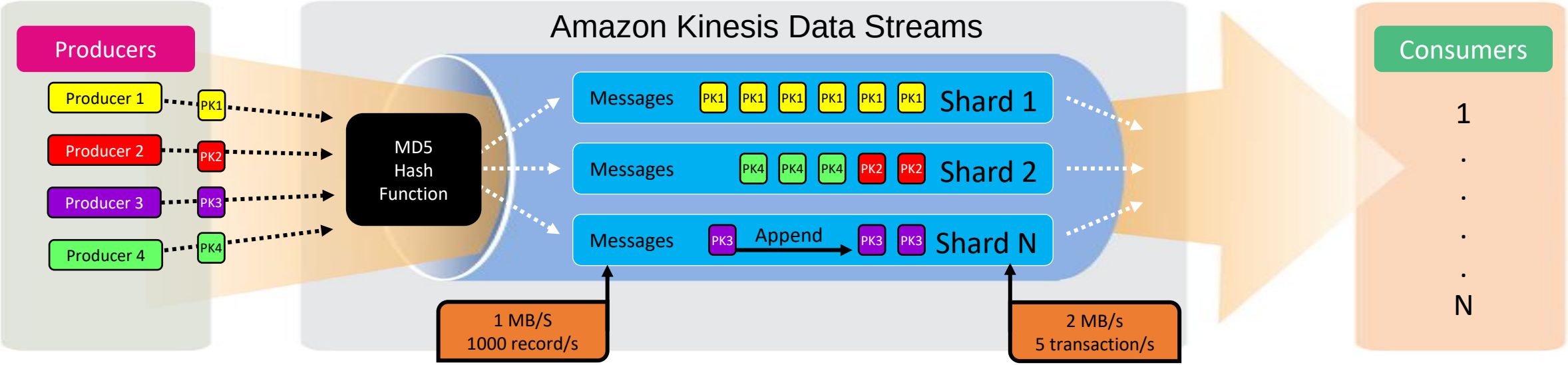


Terminology

- Partition Key

Partition Key	Seq. #	Data Blob

  - Kinesis Data Streams segregates the data records belonging to a stream into multiple shards. It uses the partition key that is associated with each data record to determine which shard a given data record belongs to.
  - Partition keys are Unicode strings, with a maximum length limit of 256 characters for each key. An MD5 hash function is used to map partition keys to 128-bit integer values and to map associated data records to shards using the hash key ranges of the shards. When an application puts data into a stream, it must specify a partition key.



Throttling

- Each Kinesis shard has 1 MiB of data per second or 1000 records in write capacity and 5 read transactions in read capacity.
- Exceeding these limits results in throttling (*WriteThroughputExceeded*) and your producers and consumers will not be able to carry out their operations.
- Causes of throttling include:
  - Hot shards: throttles are caused by a few shards in the stream that receive more requests than the average shard. This is usually caused by uneven partition key distribution
    - Possible solution – Carefully select your partition key so distribution is as balanced as possible.
  - Not enough capacity: throttles are caused by not having enough shards to service the number of requests your application
    - Possible solution – Increase capacity or use On-demand mode

Amazon Kinesis Hot Shard Advisor Tool

Amazon Kinesis Data Stream

Amazon Kinesis Data Stream

Stream Producer

- Amazon Kinesis Producer Library (KPL)
- AWS SDK
- Amazon Kinesis Agent
- Amazon MSK Connect
- Other AWS Services

Stream Storage

- Amazon Kinesis Data Streams
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)

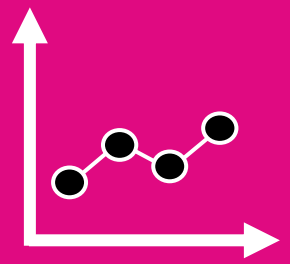
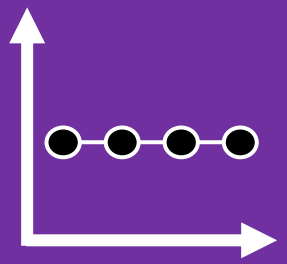
Stream Consumer

- Amazon Data Firehose
- Amazon Managed Service for Apache Flink
- AWS Glue
- Amazon EMR
- Custom Apps on EC2, Containers, Lambda using KCL

Stream Destination

- Amazon S3
- Amazon Redshift
- Amazon OpenSearch Service
- Amazon DynamoDB
- 3rd Party Services

Capacity modes

On Demand	Provisioned
	
New streams with unknown traffic	Predictable traffic
Unpredictable traffic	Traffic is consistent
Prefer the ease of hands-free management	Need tight control of shards
Lower cost of ownership	Predictable costs
Limited to 200 MB/s of write and 400 MB/s of read throughput	No capacity limits for a provisioned stream

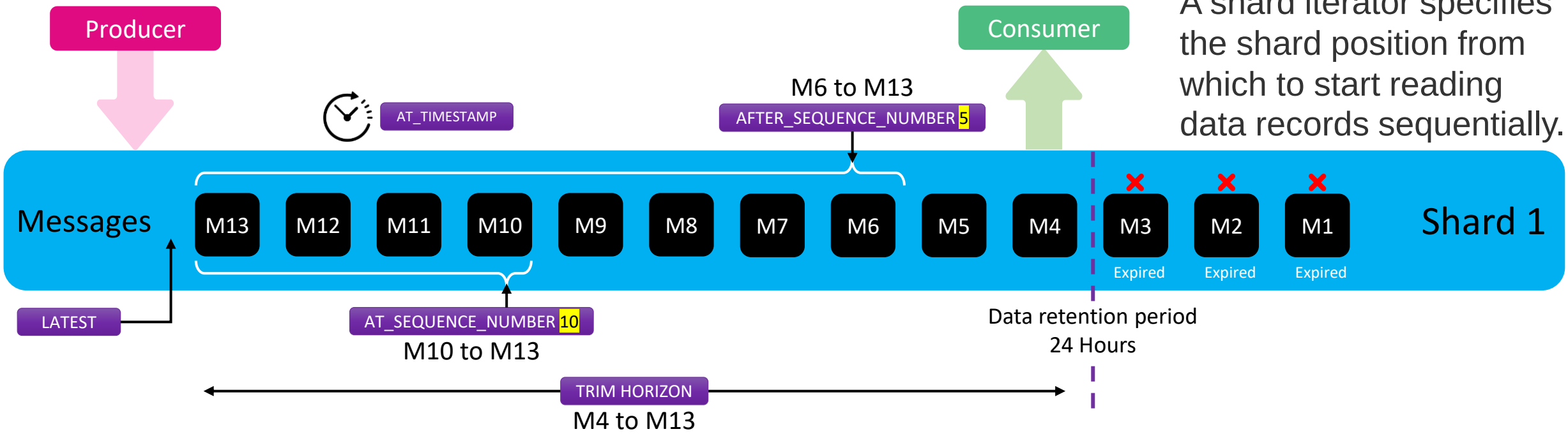
Perform Basic Kinesis Data Stream Operations Using the AWS CLI

- Step 1: Create a Stream
  - aws kinesis create-stream --stream-name Foo

- Step 2: Put a Record
  - aws kinesis put-record --stream-name Foo --partition-key 123 --data testdata
    - You can randomly generate partition key using *--partition-key `uuidgen`*

- Step 3: Get the Record
    - Before you can get data from the stream you need to obtain the shard iterator for the shard you are interested in. A shard iterator represents the position of the stream and shard from which the consumer (get-record command in this case) will read.
  - aws kinesis get-shard-iterator --shard-id shardId-000000000000 --shard-iterator-type TRIM_HORIZON --stream-name Foo
  - aws kinesis get-records --shard-iterator <Shard-iterator>

Shard iterator types

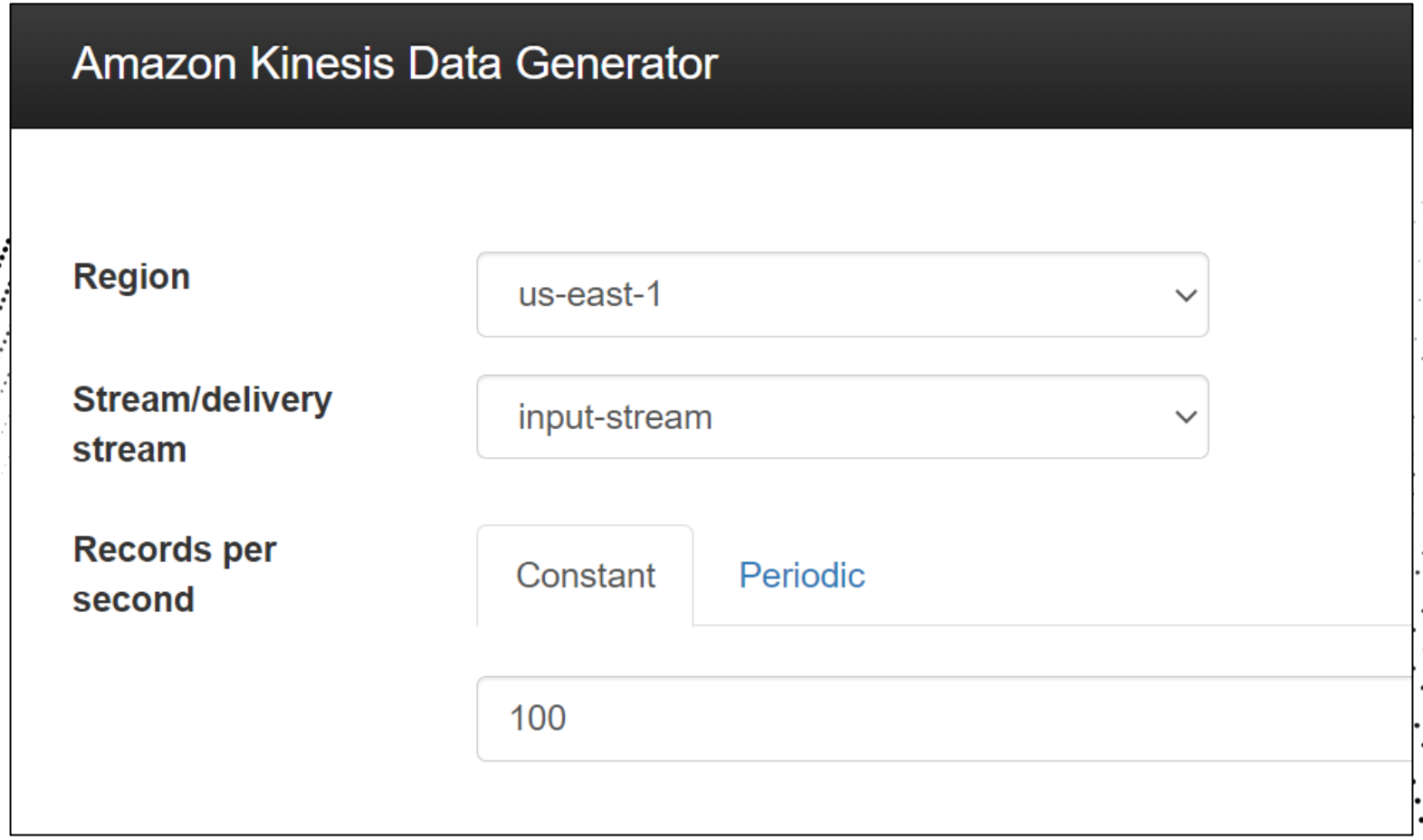


A shard iterator specifies the shard position from which to start reading data records sequentially.

- AT_SEQUENCE_NUMBER: read exactly from the position denoted by a specific sequence number.
- AFTER_SEQUENCE_NUMBER: read right after a specific sequence number.
- AT_TIMESTAMP: read records from an arbitrary point in time.
- TRIM_HORIZON: cause the Shard Iterator to point to the last untrimmed record in the shard in the system (the oldest data record in the shard).
- LATEST: show records just after the most recent record in the shard.

Amazon Kinesis Data Generator

- The KDG makes it simple to send test data to your Amazon Kinesis stream or Amazon Kinesis Firehose delivery stream.



Stream Producers (1)

Stream Producer

Amazon Kinesis Producer Library (KPL)

AWS SDK

Amazon Kinesis Agent

Other AWS Services

- Amazon Kinesis Producer Library (KPL)
  - Amazon Kinesis Producer Library (KPL) is an easy to use and highly configurable library that helps you put data into an Amazon Kinesis data stream. Amazon Kinesis Producer Library (KPL) presents a simple, asynchronous, and reliable interface that enables you to quickly achieve high producer throughput with minimal client resources.
- Amazon Kinesis Data Streams API
  - Amazon Kinesis Data Streams provides two APIs for putting data into an Amazon Kinesis Stream: PutRecord and PutRecords. PutRecord allows a single data record within an API call and PutRecords allow multiple records within API call.
- Amazon Kinesis Agent
  - Amazon Kinesis Agent is a pre-built Java application that offers an easy way to collect and send data to your Amazon Kinesis stream. You can install the agent on Linux-based server environments such as web servers, log servers, and database servers. The agent monitors certain files and continuously sends data to your stream.

Stream Producers (2)

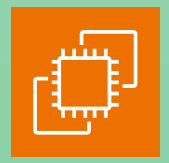
Stream Producer

- Amazon Kinesis Producer Library (KPL)
- AWS SDK
- Amazon Kinesis Agent
- Other AWS Services

- Other AWS Services
  - AWS Amplify
  - Amazon Aurora
  - Amazon CloudFront
  - Amazon CloudWatch Logs
  - Amazon Connect
  - AWS Database Migration Service
  - Amazon DynamoDB
  - Amazon EventBridge
  - AWS IoT Core
  - Amazon Relational Database Service
  - Amazon Pinpoint
  - Amazon Quantum Ledger Database
- 3rd party integrations
  - Apache Flink
  - Fluentd
  - Debezium
  - Oracle GoldenGate
  - Kafka Connect
  - Adobe Experience
  - Striim

Stream Consumers (1)

Stream Consumer



Custom Apps on EC2, Containers, Lambda using KCL

- Custom Application using Amazon Kinesis Client Library (KCL)
  - You can build custom applications using Amazon Kinesis Client Library, a prebuilt library, or the Amazon Kinesis Data Streams API.
  - KCL for Java, Python, Ruby, Node.js, and .NET is a prebuilt library that helps you easily build Amazon Kinesis applications for reading and processing data from an Amazon Kinesis data stream.
  - KCL handles complex issues such as adapting to changes in data stream volume, load-balancing streaming data, coordinating distributed services, and processing data with fault tolerance. KCL enables you to focus on business logic while building applications.

Processing data stored in Kinesis Data Streams (2)

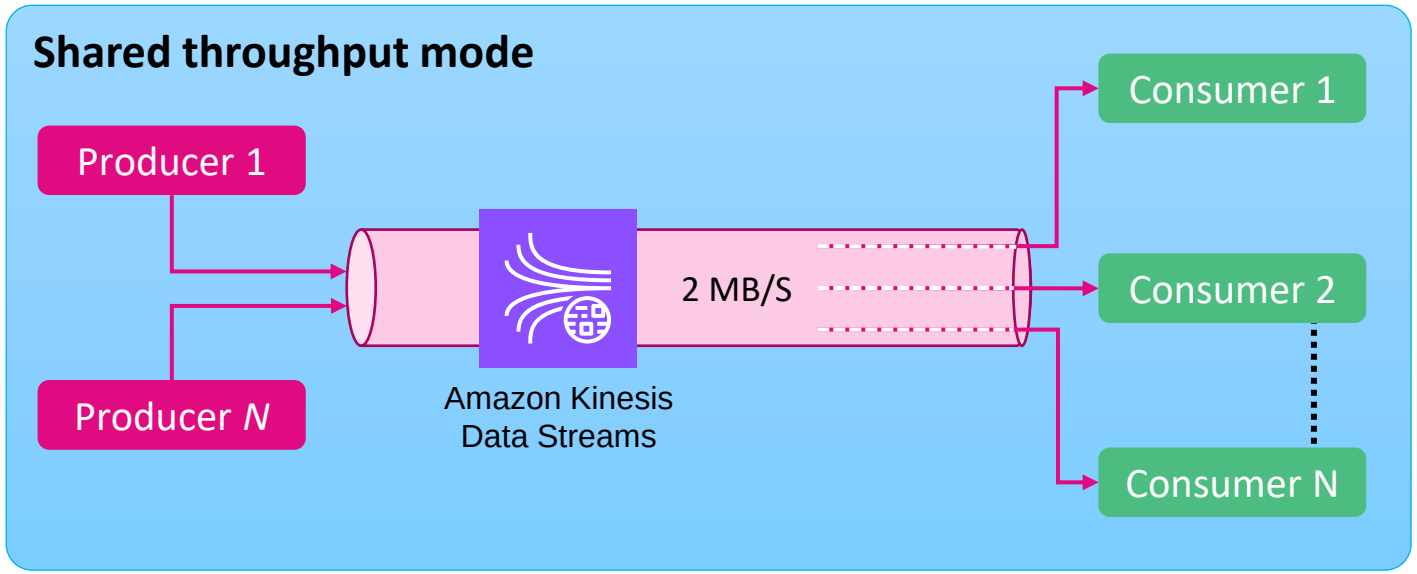
Stream Consumer

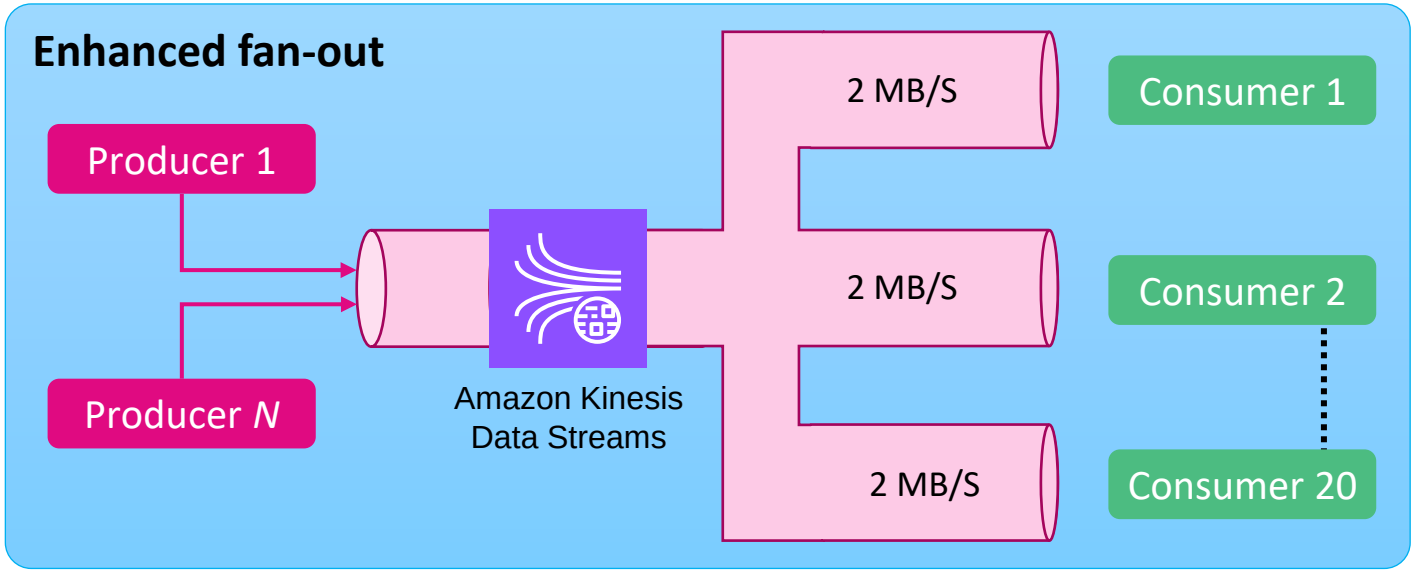
- Amazon Data Firehose
- Amazon Managed Service for Apache Flink
- AWS Glue
- Amazon EMR
- Custom Apps on EC2, Containers, Lambda using KCL

- Using other AWS Services
  - You can use managed services such as AWS Lambda, Amazon Managed Service for Apache Flink, and AWS Glue to process data stored in Kinesis Data Streams.
- 3rd party integrations
  - Apache Flink
  - Adobe Experience Platform
  - Apache Druid
  - Apache Spark
  - Databricks
  - Kafka Confluent Platform
  - Kinesumer
  - Talend

Enhanced fan-out

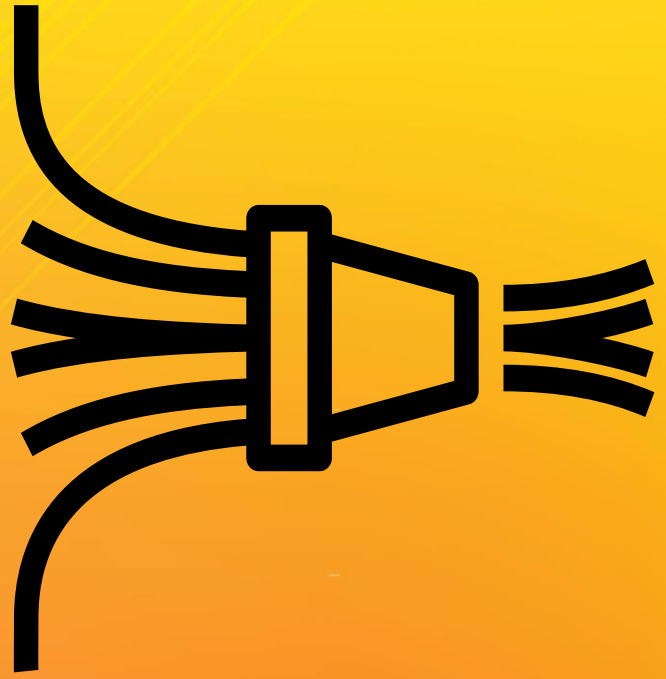
- An enhanced fan-out consumer gets its own 2 MB/second allotment of read throughput, allowing multiple consumers to read data from the same stream in parallel, without contending for read throughput with other consumers.
- You need to use the SubscribeToShard API with the enhanced fan-out consumers.





Comparision

	Shared throughput	Enhanced fan-out
When to use	• Total number of consuming applications is low (< 3) • Consumers are not latency-sensitive	• Multiple consumer applications • Low-latency requirements for data processing
Shard Read Throughput	• Fixed at a total of 2 MB/sec per shard. If there are multiple consumers reading from the same shard, they all share this throughput. The sum of the throughputs they receive from the shard doesn't exceed 2 MB/sec.	• Scales as consumers register to use enhanced fan-out. Each consumer registered to use enhanced fan-out receives its own read throughput per shard, up to 2 MB/sec, independently of other consumers.
Message propagation delay	• An average of around 200 ms if you have one consumer reading from the stream. This average goes up to around 1000 ms if you have five consumers.	• Typically an average of 70 ms whether you have one consumer or five consumers.
Cost	• N/A	• There is a data retrieval cost and a consumer-shard hour cost.
Record delivery model	• Pull model over HTTP using GetRecords.	• Kinesis Data Streams pushes the records to you over HTTP/2 using SubscribeToShard.



Amazon Data Firehose

A Firehose

Amazon Data Firehose

Stream Producer

- Amazon Kinesis Producer Library (KPL)
- AWS SDK
- Amazon Kinesis Agent
- Amazon MSK Connect
- Other AWS Services

Stream Storage

- Amazon Kinesis Data Streams
- Amazon Managed Streaming for Apache Kafka (Amazon MSK)

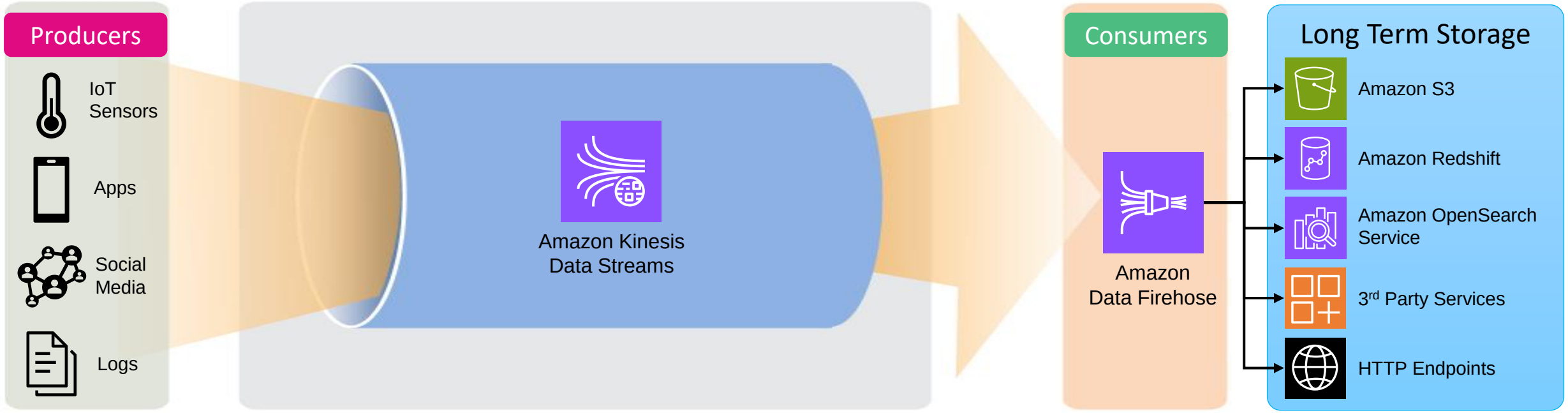
Stream Consumer

- Amazon Data Firehose

Stream Destination

- Amazon S3
- Amazon Redshift
- Amazon OpenSearch Service
- HTTP Endpoints
- 3rd Party Services

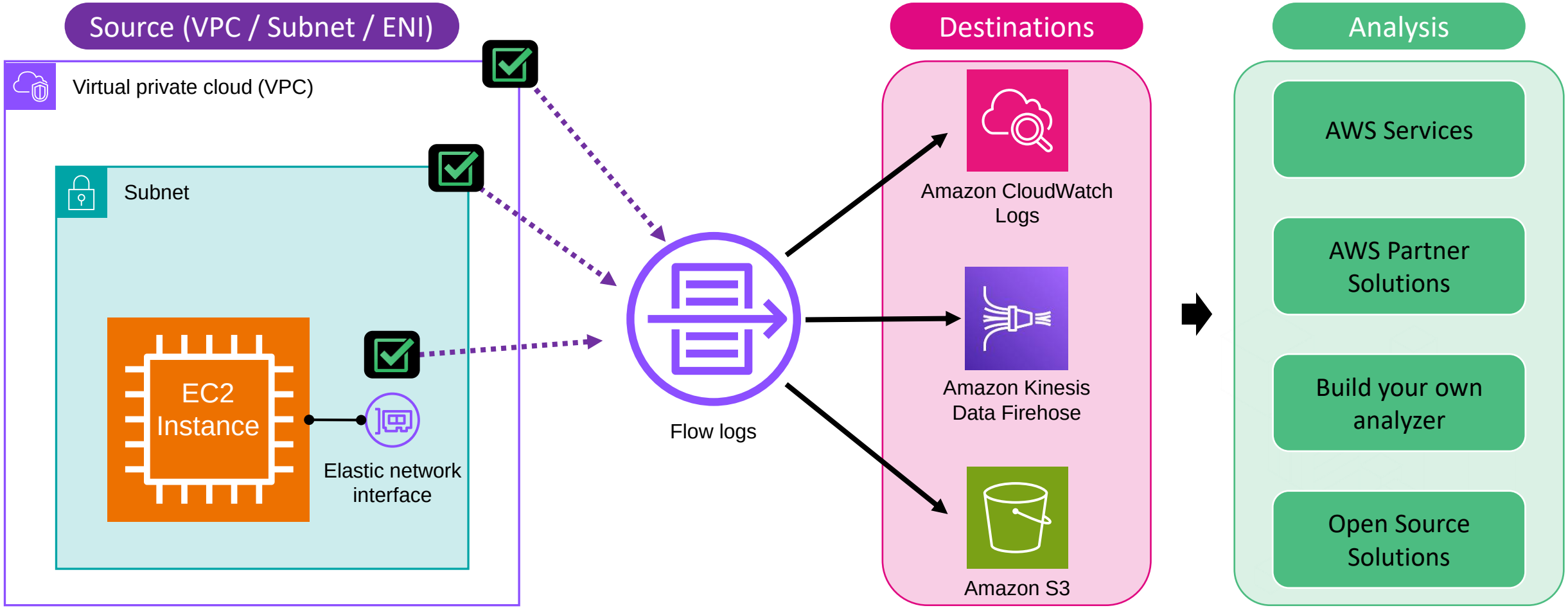
Amazon Data Firehose



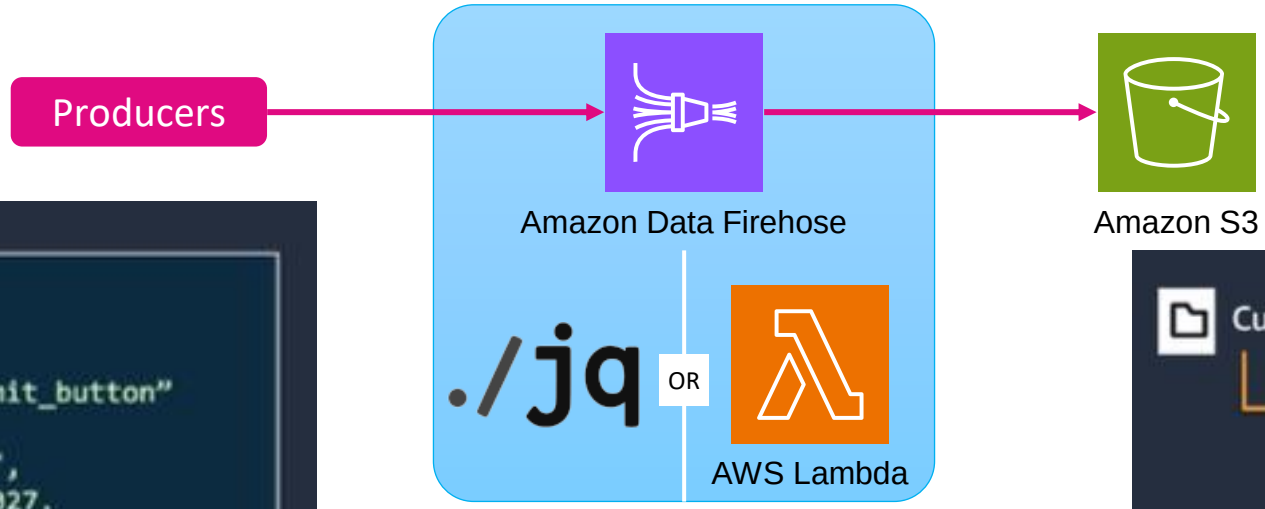
- Zero administration and seamless elasticity
- Direct-to-data store integration
- Serverless continuous data transformations
- Data format conversion to Parquet/ORC
- Deliver data directly to Datadog, Sumo Logic, New Relic, MongoDB, Snowflake and more…

VPC Flow Logs

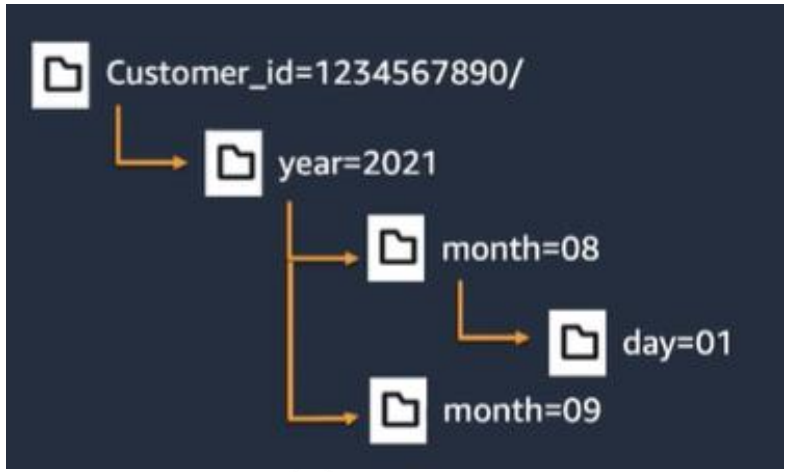
- VPC Flow Logs is a feature that enables you to capture information about the IP traffic going to and from network interfaces in your VPC.



Dynamically partition streaming data for Amazon S3 destination




```
{
  "type": {
  "device": "mobile",
  "event": "user_clicked_submit_button"
  },
  "customer_id": "1234567890",
  "event_timestamp": 1565382027,
  "region": "pdx"
}
```



- Organize streaming data into partitions on Amazon S3
- Eliminate post processing using another application to partition data
- Partitioned data enhances query performance, lowers cost, improves security

	Amazon Kinesis Data Streams	Amazon Data Firehose
Objective	• Streaming data ingestion and storage	• Streaming data transfer and format conversion
Scaling	• Provisioned mode – Manual • On demand mode – Automated	• Automated
Data Retention	• Configuration (1 day to 365 days)	• Not applicable
Processing	• Real-time processing capabilities with almost 200ms latency for classic tasks and almost 70ms latency for enhanced fan-out tasks.	• Near real-time processing capabilities, depending on the buffer size or minimum buffer time of 60 seconds.
Replay Capabilities	• Supported	• Unsupported
Parallel Consumption	• Supported	• Unsupported
Max message size for put requests	• 1 MB	• 1 MB • 10 MB if your data source is Amazon MSK
Data Producers (Partial list)	• Amazon Kinesis Agent • AWS SDK • Amazon Kinesis Producer Library (KPL)	• Amazon Kinesis Data Stream • Amazon MSK • Direct Put
Data Consumers (Partial list)	• Managed Apache Flink • Amazon Data Firehose • Amazon Kinesis Client Library (KCL)	• Amazon OpenSearch • Amazon Redshift • Amazon S3 • HTTP Endpoints & 3rd Party Services

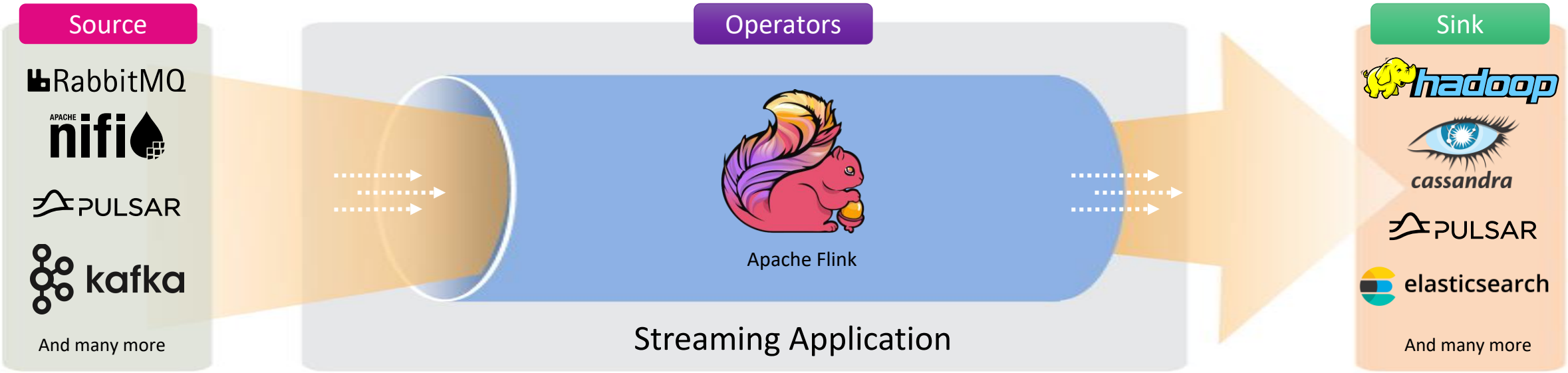
Amazon Managed Service for Apache Flink

What is Apache Flink?

- Apache Flink is an open-source framework and engine for processing batch and streaming data.
- It supports diverse use cases:
  - Process both batch and streaming data
  - Transform and act on streaming data
  - Extract value from streaming data
  - Event-driven applications
  - Streaming Analytics & ETL
  - Batch analytics
- Apache Flink can be programmed with multiple languages, from the high level streaming SQL to Python, Scala, Java, but also other JVM languages like Kotlin.

How it works?

- A typical Apache Flink application has the following components:
  - Sources: The application consumes data from one or more sources. A source uses a connector to read data from an external system.
  - Sinks: The application sends data to external sources through sinks. Data sinks consume streaming data and forward them to files, sockets, external systems, or print them.
  - Operators: The application processes data by using one or more operators. An operator can transform, enrich, or aggregate data.



Apache Flink on AWS

Amazon Managed Service for Apache Flink

Fully managed stream processing

Build and run Apache Flink applications, without setting up infrastructure and managing resources and clusters.

AWS managed scaling

Amazon EMR

Flexibility with less management

Amazon EMR supports Apache Flink as a YARN application.

Amazon EKS

Full flexibility with complete control

Apache Flink natively supports Kubernetes. You can self-host Apache Flink on Amazon EKS or Amazon EC2.

Customer managed scaling

Timeline

2016
Launched Kinesis Data Analytics for SQL
(Not Flink-based)

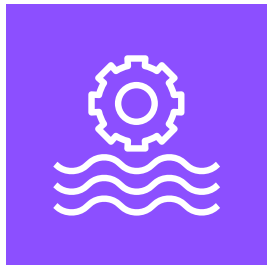
2018
Launched Kinesis Data Analytics for "Java" based on Flink

2019
Launched Amazon MSK (Managed Kafka), outside the Kinesis brand

2020
Rebranded Kinesis Data Analytics for Java to Kinesis Data Analytics for Apache Flink

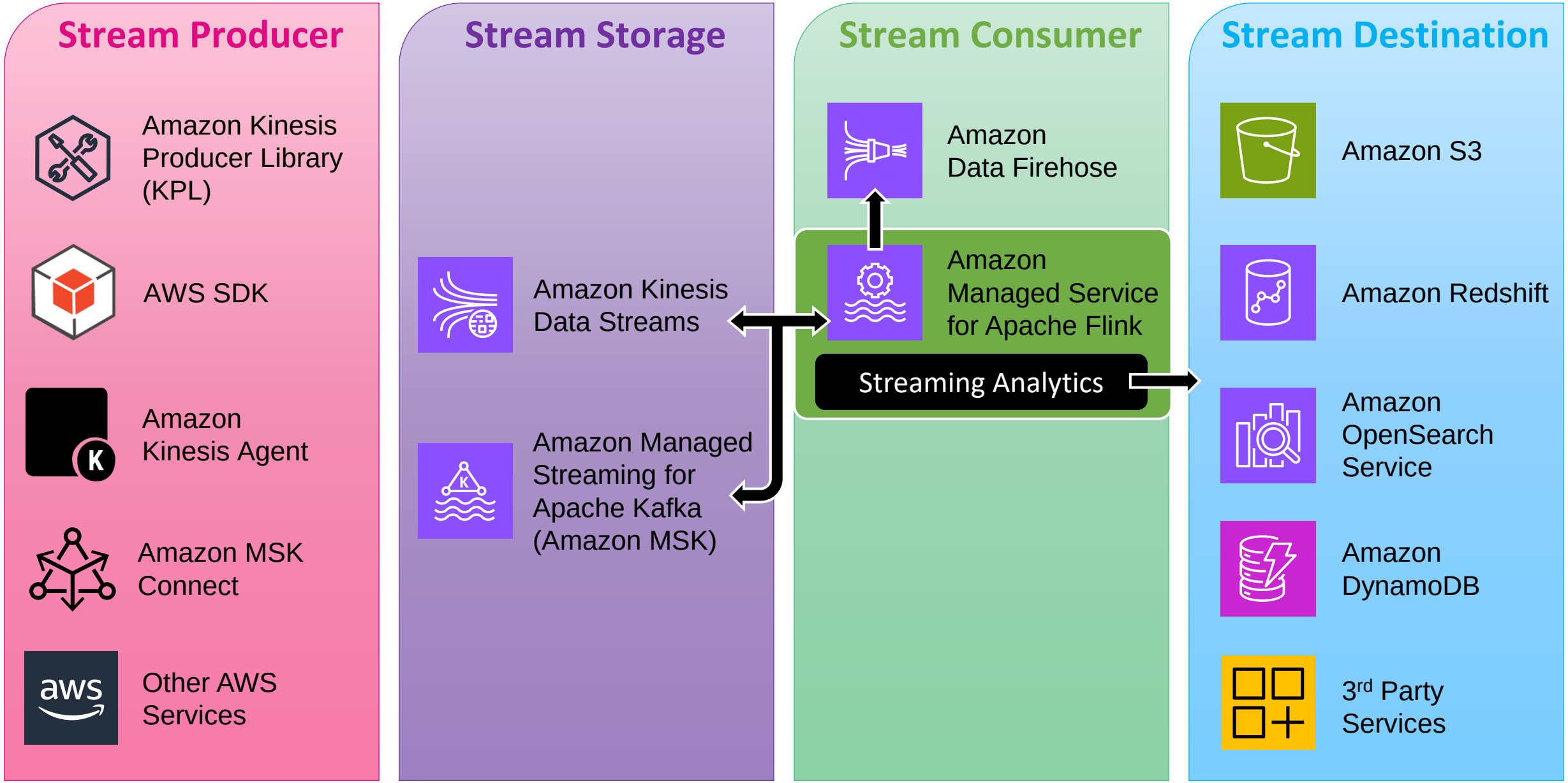
2023
Launched Amazon Managed Service for Apache Flink

Amazon Managed Service for Apache Flink

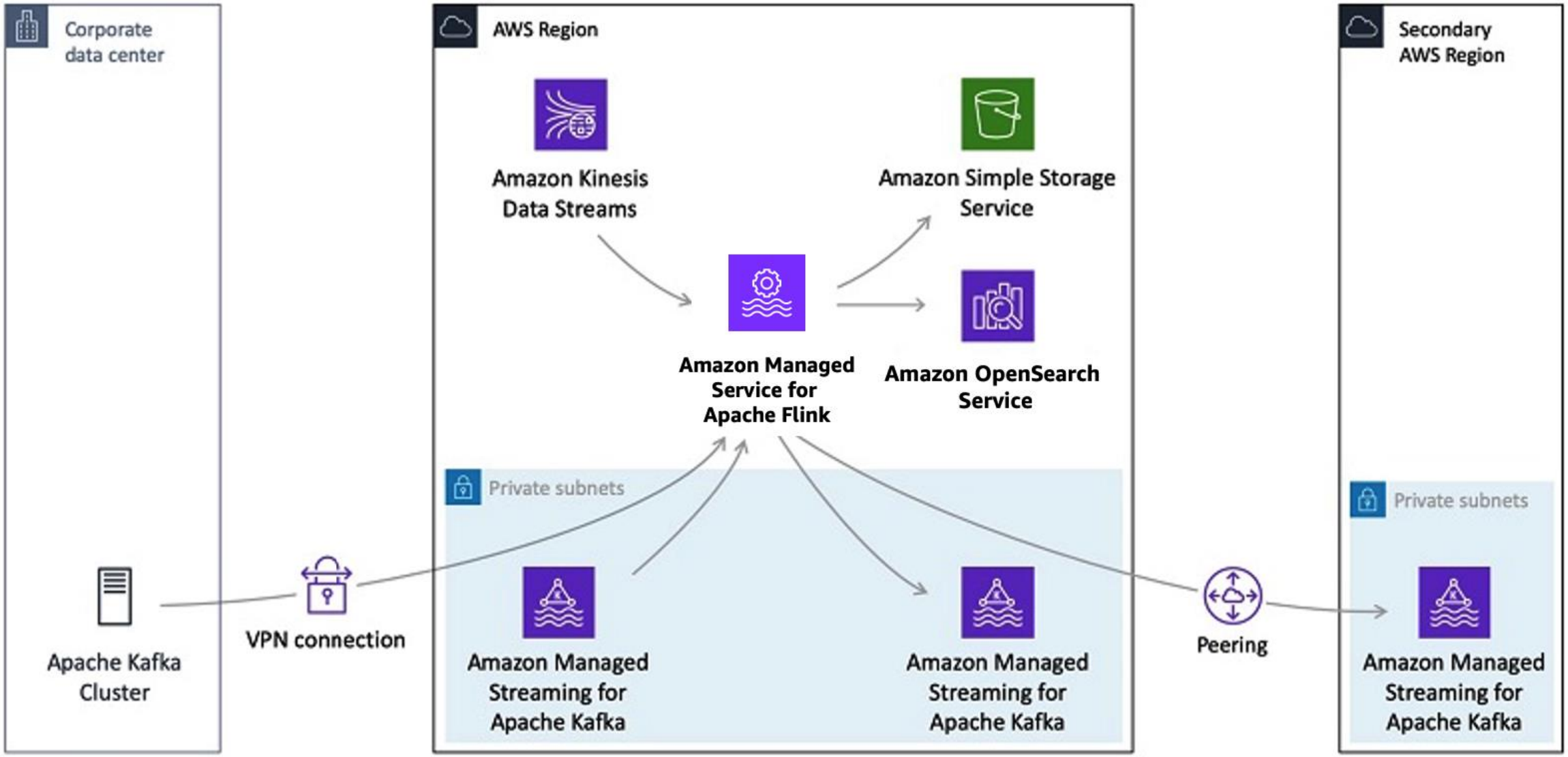


- Amazon managed Service for Apache Flink is a serverless service that provides the underlying infrastructure for your Apache Flink applications.
- It handles core capabilities like provisioning compute resources, AZ failover resilience, parallel computation, automatic scaling, and application backups.
- You can use the high-level Flink programming features (such as operators, functions, sources, and sinks) in the same way that you use them when hosting the Flink infrastructure yourself.
- It allows you to set up and integrate data sources and destinations with minimal code, continuously process data with sub second latencies, and respond to events in real time.
- You can use Java, Scala, Python, or SQL to process and analyze streaming data.

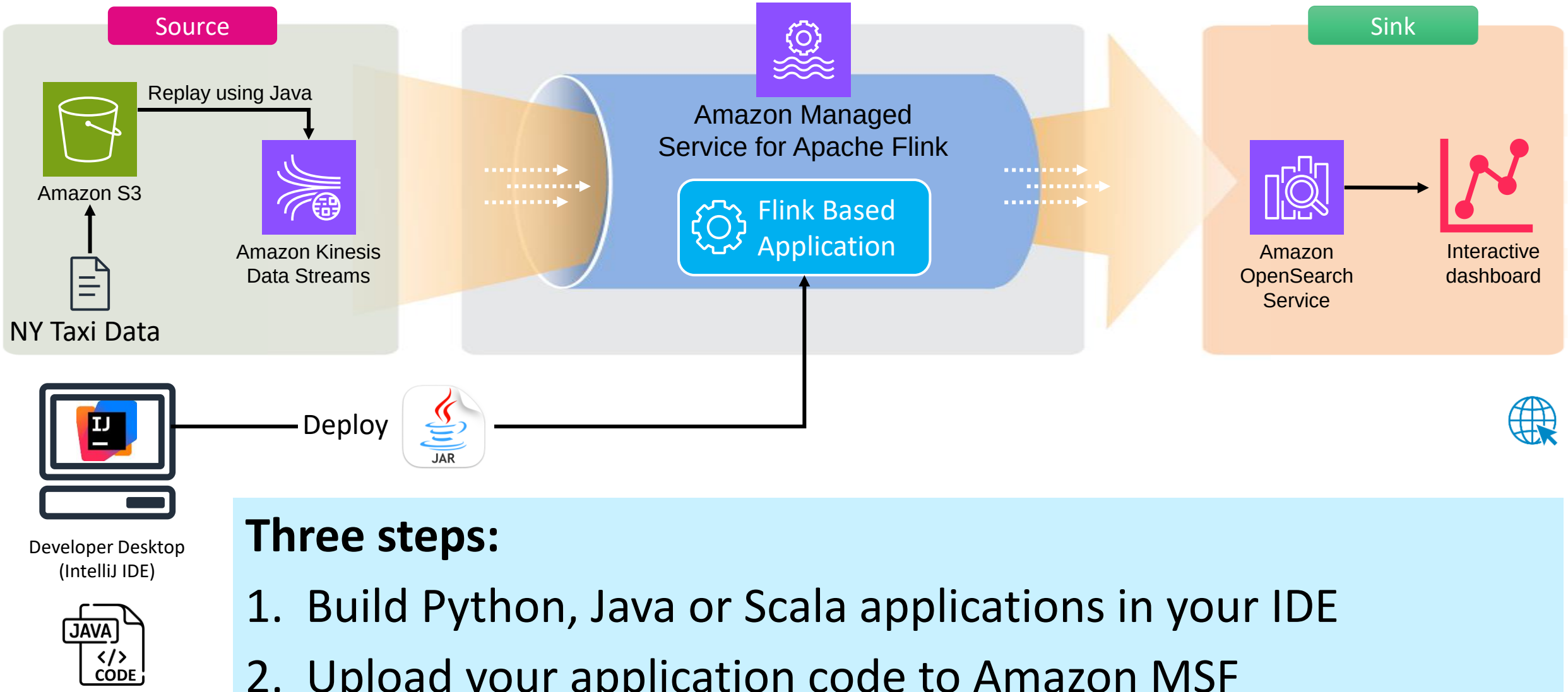
Integration with AWS Streaming Services



Architecture for streaming ETL with Apache Flink

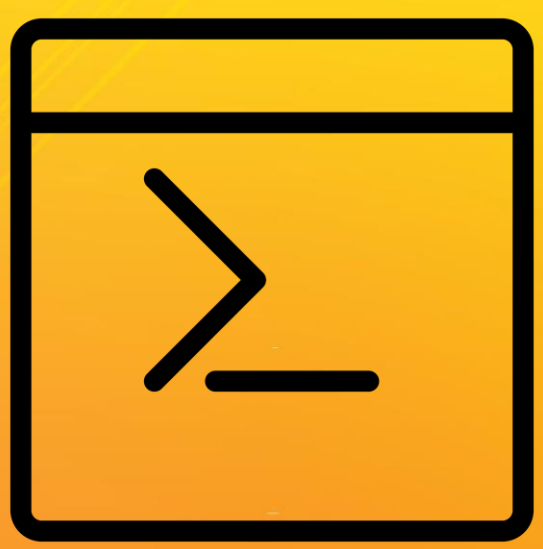


Amazon Managed Service for Apache Flink Workshop



Three steps:

1. Build Python, Java or Scala applications in your IDE
2. Upload your application code to Amazon MSF
3. Run your application in a fully managed and elastic service



Amazon Managed Flink Studio

Amazon Managed Flink Studio

Stream inspection & visualization

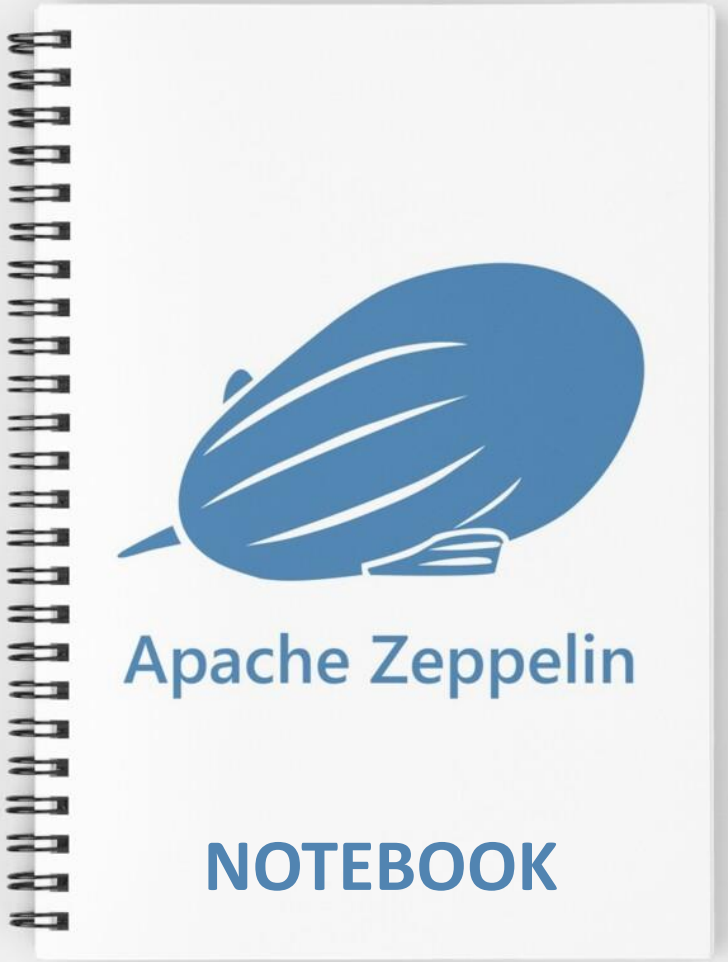
Simple build and run environment

Serverless, rapid development of stream processing applications

Process using SQL, Python, or Scala

Integrates with AWS Glue Data Catalog

Studio notebooks



- Studio notebooks uses notebooks powered by Apache Zeppelin, and uses Apache Flink as the stream processing engine
- Apache Zeppelin is a browser-based IDE for managing code, computation inputs and outputs, visualizations, and labels.
- With notebooks, you get a simple interactive development experience combined with the advanced capabilities provided by Apache Flink.
- You can use Scala/Python or SQL for Flink application testing, development and troubleshooting.

Amazon Managed Flink Studio

Zeppelin Notebook

sql_1.13

Zeppelin notebooks (ie. this notebook) is organized into blocks of code know as paragraphs.

You can run a paragraph by clicking on the play button in the top right corner of each paragraph or by clicking on the paragraph and using the keyb

To stop the execution of a paragraph click the pause button in the top right corner of each paragraph or click on the paragraph and use the key boa

Flink's Zeppelin Interpreters

Name	Class	
%flink	FlinkInterpreter	Creates ExecutionEnvironment/StreamExecutionEnvironment/BatchTableEnvironment/StreamTable
%flink.pyflink	PyFlinkInterpreter	
%flink.ipyflink	IPyFlinkInterpreter	
%flink.ssql	FlinkStreamSqlInterpreter	
%flink.bsql	FlinkBatchSqlInterpreter	

Create Table in Glue Data Catalog

First lets create a table using `format = raw` this will allow us to visually inspect the data sent to Kinesis unformatted. After we inspect the raw data

```
%flink.ssql

DROP TABLE IF EXISTS yellow_cab_raw_1v13;

CREATE TABLE yellow_cab_raw_1v13 (
   `raw_data` STRING
)
 WITH (
   'connector' = 'kinesis',
   'stream' = 'yellow-cab-trip',
   'aws.region' = 'us-east-1',
   'scan.stream.initpos' = 'LATEST',
   'format' = 'raw'
);
```

```
Table has been dropped.
Table has been created.
```

Zeppelin Notebook

sql_1.13

```
%flink.ssql(type=update)

SELECT * FROM yellow_cab_1v13 WHERE trip_distance > 3.0
```

settings

VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag
1	2020-01-01T00:12:11	2020-01-01 00:26:26.0	1	3.4	1	N
1	2020-01-01T00:18:18	2020-01-01 00:42:46.0	1	3.6	1	N
1	2020-01-01T00:12:07	2020-01-01 00:26:56.0	3	8.3	1	N
1	2020-01-01T00:15:25	2020-01-01 00:34:02.0	1	8.7	1	N
1	2020-01-01T00:59:28	2020-01-01 01:28:44.0	1	5.1	1	N
1	2020-01-01T00:11:37	2020-01-01 00:23:22.0	1	6.8	1	N
1	2020-01-01T00:48:52	2020-01-01 01:08:47.0	1	6.6	1	Y

User Defined Function

We can create Python and Scala user defined functions to enhance our queries

```
%flink.pyflink

class PythonLower(ScalarFunction):
  def eval(self, s):
    return s.lower()

bt_env.register_function("python_lower", udf(PythonLower(), DataTypes.STRING(), DataTypes.STRING()))
```

```
%flink.ssql(type=update)

SELECT store_and_fwd_flag, python_lower(store_and_fwd_flag) as lower_store_and_fwd_flag FROM yellow_cab_1v13
```

settings

store_and_fwd_flag	lower_store_and
N	n

kafka

Apache Kafka

What is Kafka?

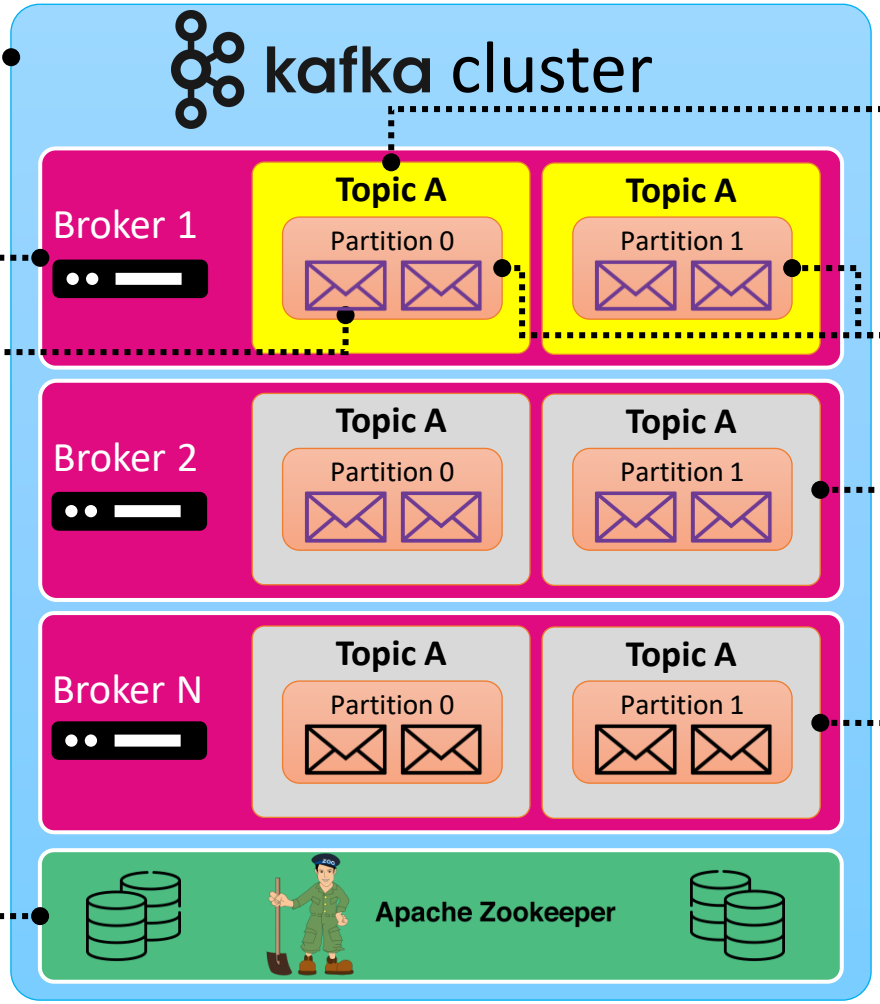
- Apache Kafka is an open-source distributed event streaming platform.
- Apache Kafka anatomy

Cluster: a collection of brokers that are coordinating and working together to service producers and consumers

Broker - the individual kafka binary running on a host, or the server running kafka.

Message - a constrained bit of data - could be a JSON object, a log line, an image file, a database transaction.

Zookeeper - a distributed data storage engine that Kafka uses to store metadata about the service, as well as using the Zookeeper leader election protocol to make decisions about which broker is the leader for which partitions.



Topic - a collection of messages that are generally related. Messages are produced to a topic, and consumed from a topic. Topics can be consumed by multiple consumers.

Partition - a portion of a Topic. Topics are divided up into partitions as a method of scaling throughput, as well as allowing for strict message ordering with appropriate configuration.

Replica - a copy of a partition held on another broker in the cluster. Automatically kept in sync with the leader.

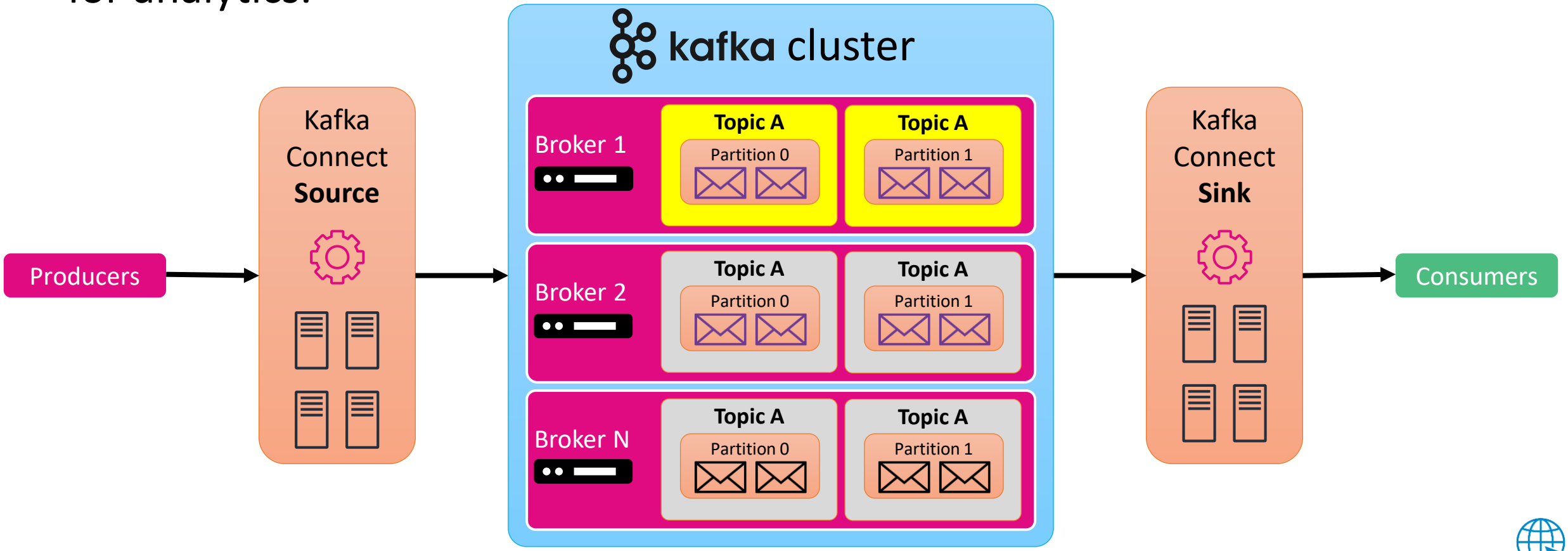
Topic distribution

- Each broker is assigned a number of partitions
- A partition is assigned to a broker at the creation time of the topic.

```
KafkaClientEC2Instance1 [ec2-user@ip-10-0-1-73 kafka]$ cd /home/ec2-user/kafka
KafkaClientEC2Instance1 [ec2-user@ip-10-0-1-73 kafka]$ bin/kafka-topics.sh --bootstrap-server $brokers --describe --topic test10
Topic: test10   PartitionCount: 10      ReplicationFactor: 3    Configs: min.insync.replicas=2,message.format.version=2.7-IV2,unclean.leader
.election.enable=true
        Topic: test10   Partition: 0    Leader: 2       Replicas: 2,1,3 Isr: 2,1,3
        Topic: test10   Partition: 1    Leader: 3       Replicas: 3,2,1 Isr: 3,2,1
        Topic: test10   Partition: 2    Leader: 1       Replicas: 1,3,2 Isr: 1,3,2
        Topic: test10   Partition: 3    Leader: 2       Replicas: 2,3,1 Isr: 2,3,1
        Topic: test10   Partition: 4    Leader: 3       Replicas: 3,1,2 Isr: 3,1,2
        Topic: test10   Partition: 5    Leader: 1       Replicas: 1,2,3 Isr: 1,2,3
        Topic: test10   Partition: 6    Leader: 2       Replicas: 2,1,3 Isr: 2,1,3
        Topic: test10   Partition: 7    Leader: 3       Replicas: 3,2,1 Isr: 3,2,1
        Topic: test10   Partition: 8    Leader: 1       Replicas: 1,3,2 Isr: 1,3,2
        Topic: test10   Partition: 9    Leader: 2       Replicas: 2,3,1 Isr: 2,3,1
KafkaClientEC2Instance1 [ec2-user@ip-10-0-1-73 kafka]$
```

Kafka Connect

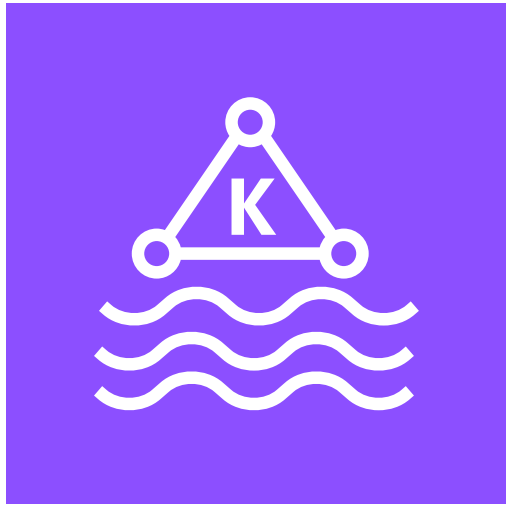
- Kafka Connect allow you to connects data sources and data sinks to Kafka.
- It allows you to move data between several different source and target systems, convert data from one form to another, and transform your data for analytics.



Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

- Eliminate operational overhead, including the provisioning, configuration, and maintenance of highly available Apache Kafka and Kafka Connect clusters.
- Amazon MSK provides the control-plane operations, such as those for creating, updating, and deleting clusters.
- It lets you use Apache Kafka data-plane operations, such as those for producing and consuming data.
- It runs open-source versions of Apache Kafka. This means existing applications, tooling, and plugins from partners and the Apache Kafka community are supported without requiring changes to application code.



Amazon MSK

Amazon MSK

Stream Producer

Amazon MSK Connect

Stream Storage

Amazon Managed Streaming for Apache Kafka (Amazon MSK)

Stream Consumer

Amazon MSK Connect

Stream Destination

Amazon S3

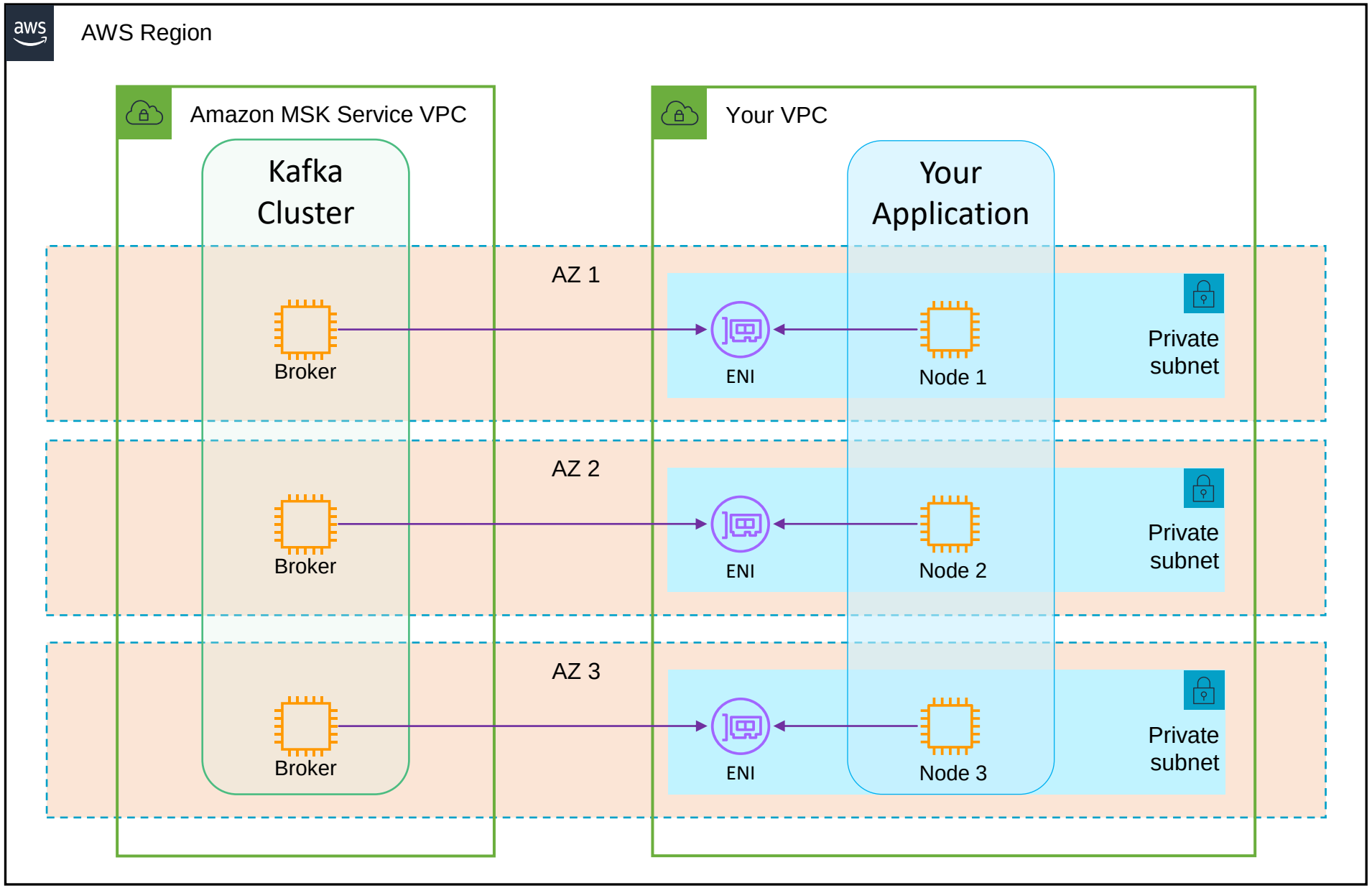
Amazon Redshift

Amazon OpenSearch Service

Amazon DynamoDB

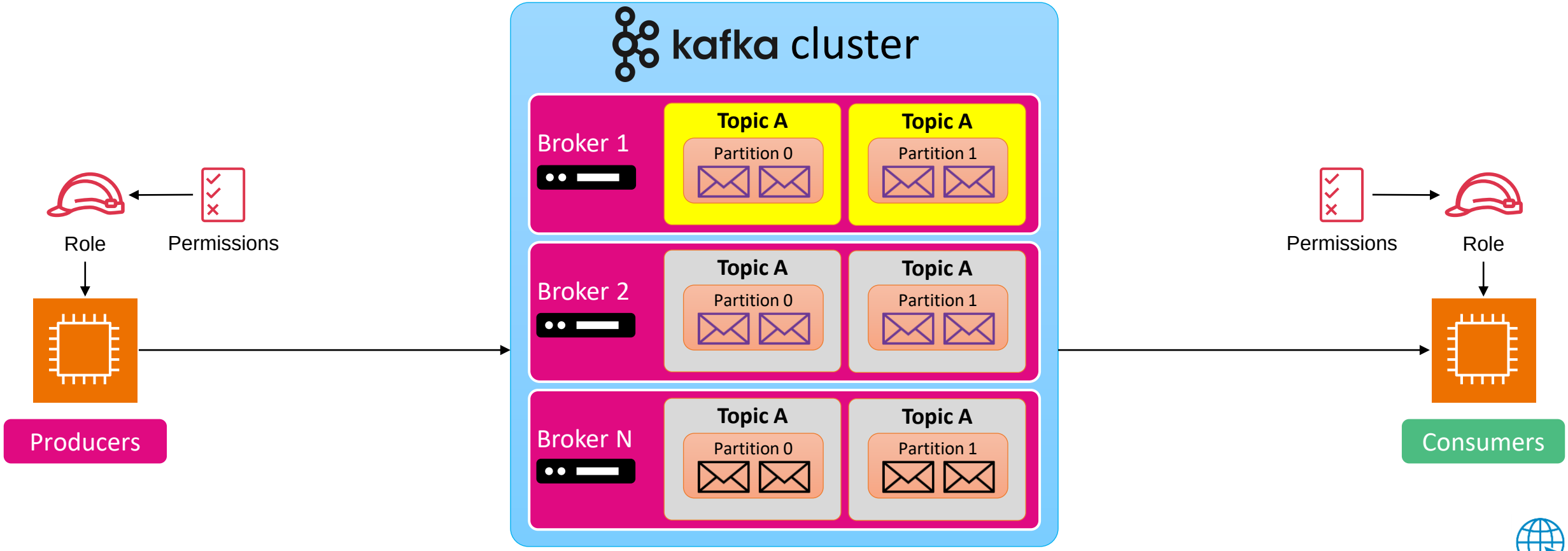
3rd Party Services

Amazon MSK Connectivity



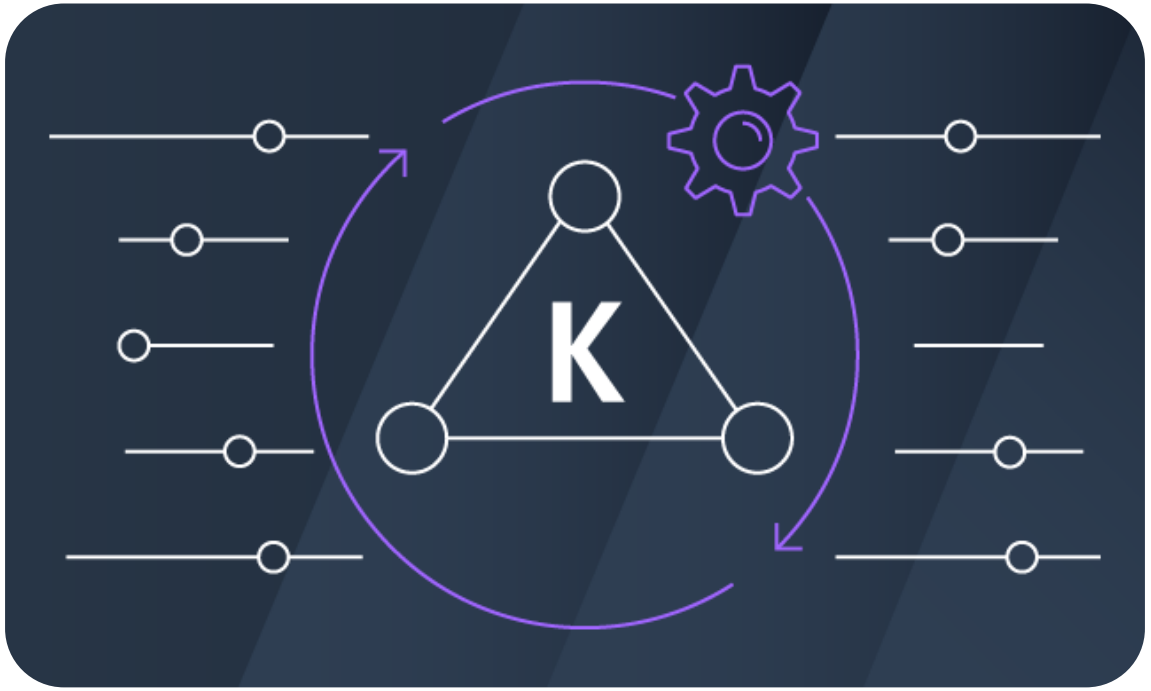
Creating an Amazon MSK cluster

- Cluster creation method
  - Quick Create
  - Custom Create
- Cluster type
  - Serverless
  - Provisioned



Amazon MSK Serverless

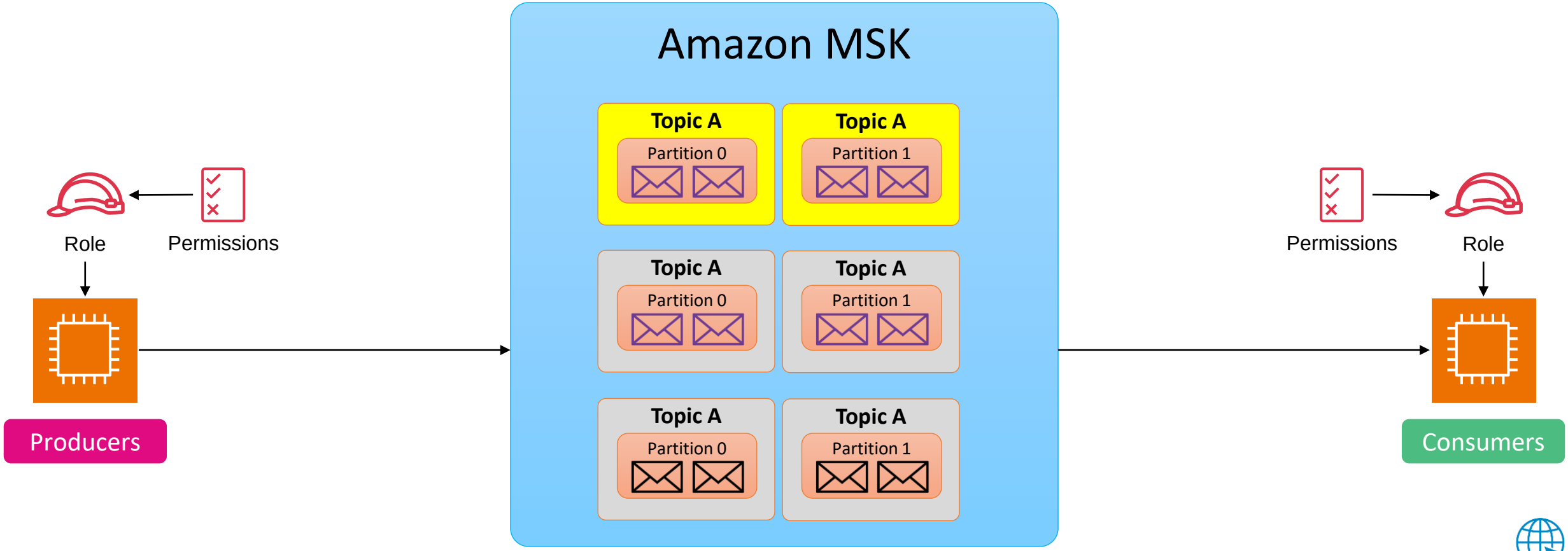
Amazon MSK Serverless



- Easily run Apache Kafka clusters without needing to right-size cluster or overprovisioning
- Instantly scale I/O without needing to worry about scaling up and down
- Automated partition reassignment and scaling
- Throughput based pricing

Creating an Amazon MSK Serverless cluster

- Cluster creation method
  - Quick Create
  - Custom Create
- Cluster type
  - Serverless
  - Provisioned

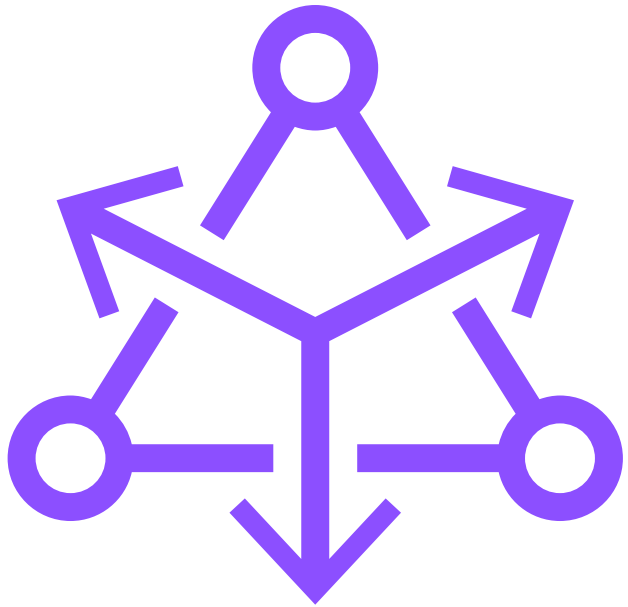


Amazon MSK Provisioned Cluster vs. Serverless

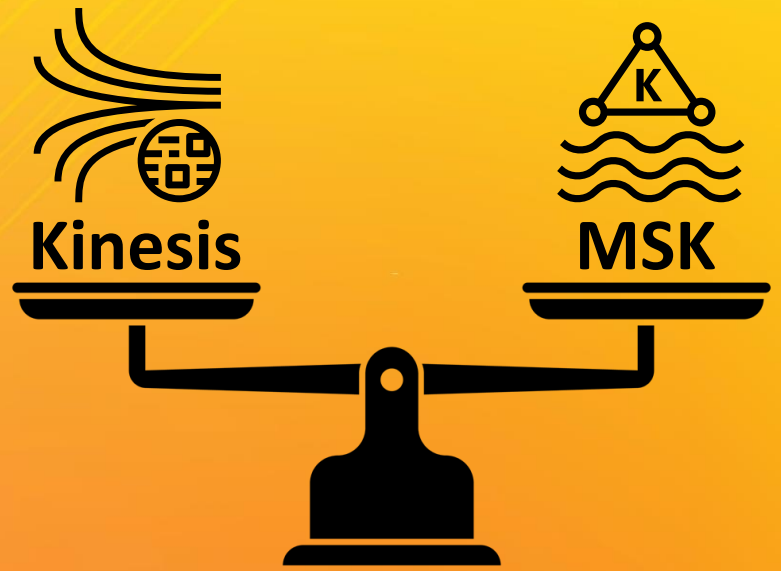
	Amazon MSK Provisioned Cluster	Amazon MSK Serverless
Creating a new cluster	Select the number of brokers, size of the instances, storage capacity, broker instance type.	No capacity configuration needed.
Workload type and use-case	Predictable workload. Stable, have more control, large workloads.	Unpredictable workload. Offload capacity management.
Scaling	Manually adjust the capacity configuration to adhere to the traffic increases.	Automatically provisions and manages capacity needed during traffic increases.
Monitoring	Manually monitor overload/idle brokers.	Auto-scales on demand so do not need to constantly monitor brokers and storage.
Configurations	Allows broker level configurations	Allows a few topic level configurations
Pricing dimensions	On-demand hourly pricing for brokers and storage	Pay-per-use pricing based on cluster hours, partition hours, storage

Amazon MSK Connect

Amazon MSK Connect

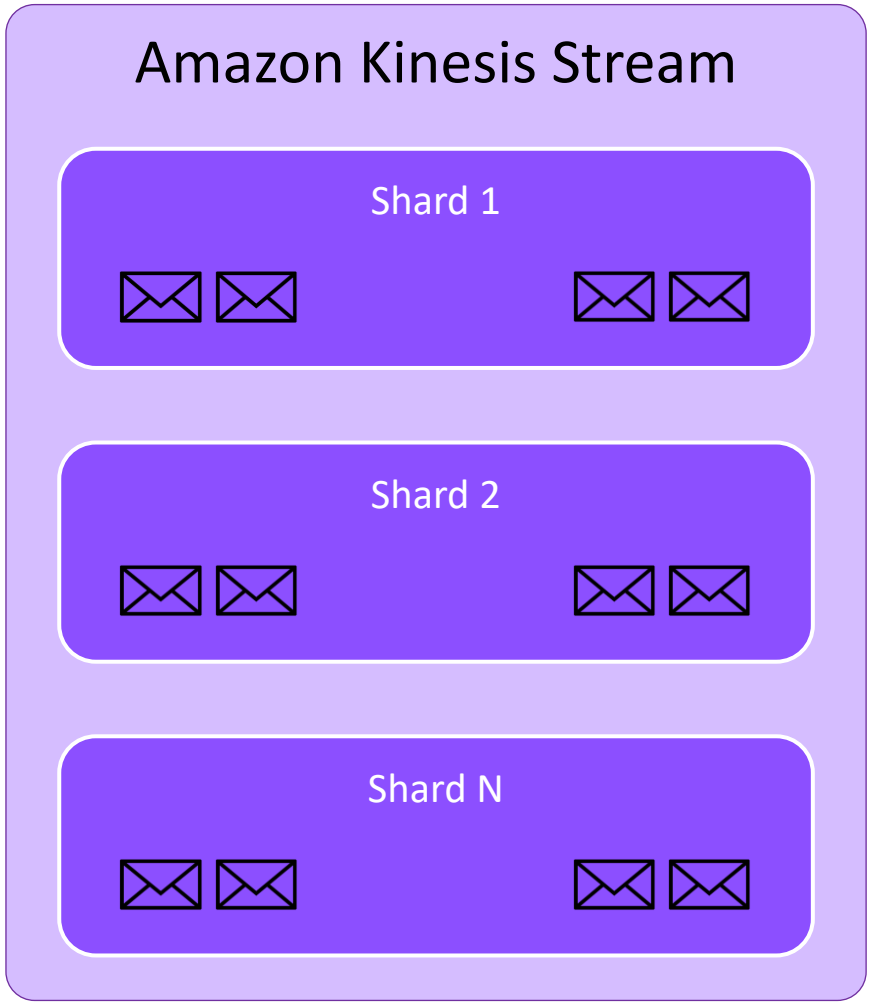


- Run fully managed Kafka Connect clusters with Amazon MSK
- Eliminates the need to provision and maintain cluster infrastructure
- Easily deploy, monitor and scale connectors that move data in and out of Apache Kafka and Amazon MSK
- Connectors scale automatically in response to increases in usage and you pay only for the resources you use
- Fully compatible with Kafka Connect that makes it easy to migrate workloads without code changes.

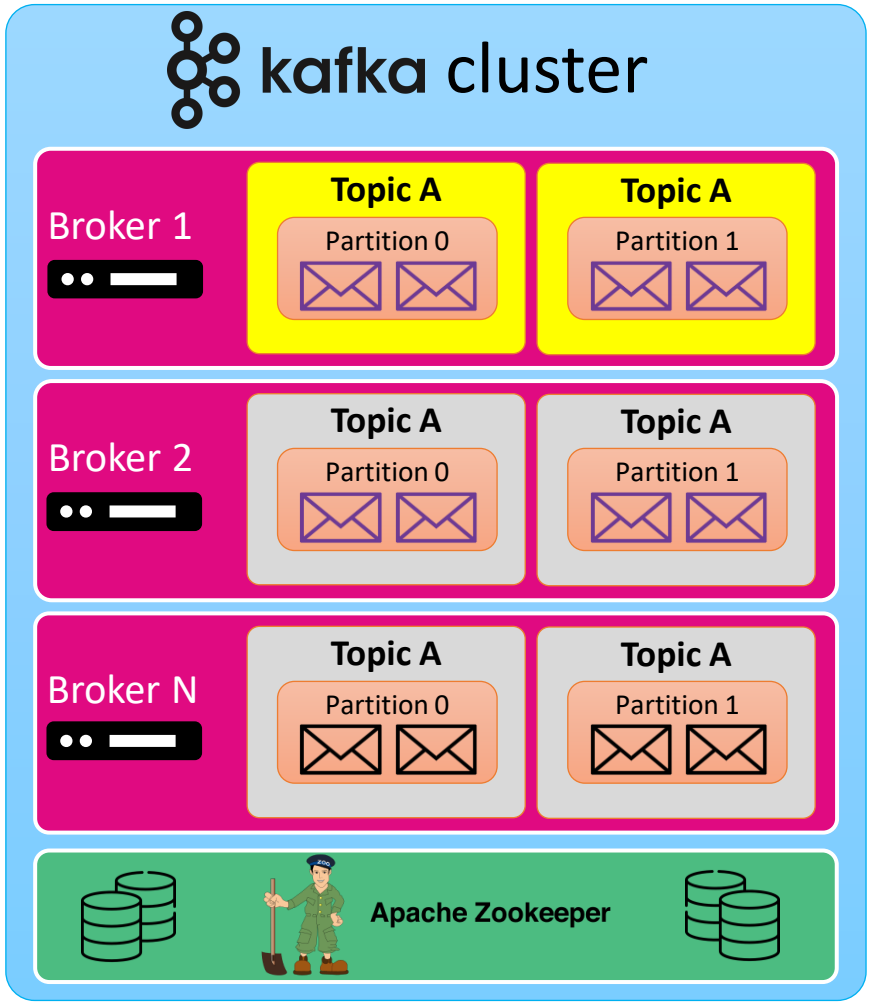


Comparison

Terminology – Kinesis vs. Kafka



Kinesis	Kafka
Stream	Topic
Shard	Partition
Sequence Number	Offset
N/A	Cluster
N/A	Broker



Similarity	AWS managed infrastructure
	Parallel consumption
	Guaranteed ordering: Kinesis – Shard level, MSK – Partition level
	Support Managed Service for Apache Flink

	Amazon Kinesis Data Stream	Amazon MSK
Differences	Serverless	Provisioned and Serverless
	Proprietary to AWS	Based on open source Apache Kafka
	3 AZ replication	Configurable replication
	Suitable for cloud native application	Suitable for on-prem Kafka migration to cloud
	1 MB message/blob size (blob)	Configurable message/blob size
	Integrates natively with AWS services	Supports integration with Kafka ecosystem
	Low (70ms with Enhance Fan Out)	Lowest