



A person's hands are shown holding a glowing digital globe. The globe is composed of a network of blue and red lines, with a bright yellow and orange light source on the right side. The background is a dark, blue-tinted image of a person's torso and arms, with a blurred background of trees.

Kinesis

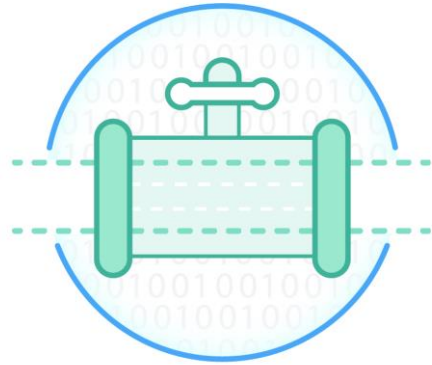
Data Stream Processing Services

Roger S. Barga, Ph.D.
General Manager
Amazon Web Services



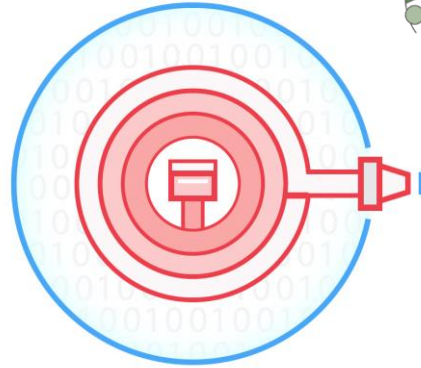
Amazon Kinesis

Services that make it easy to work with real-time data streams on AWS



Amazon Kinesis Streams

Build your own custom applications that process or analyze streaming data



Amazon Kinesis Firehose

Easily load massive volumes of streaming data into Amazon S3 and Redshift



Amazon Kinesis Analytics

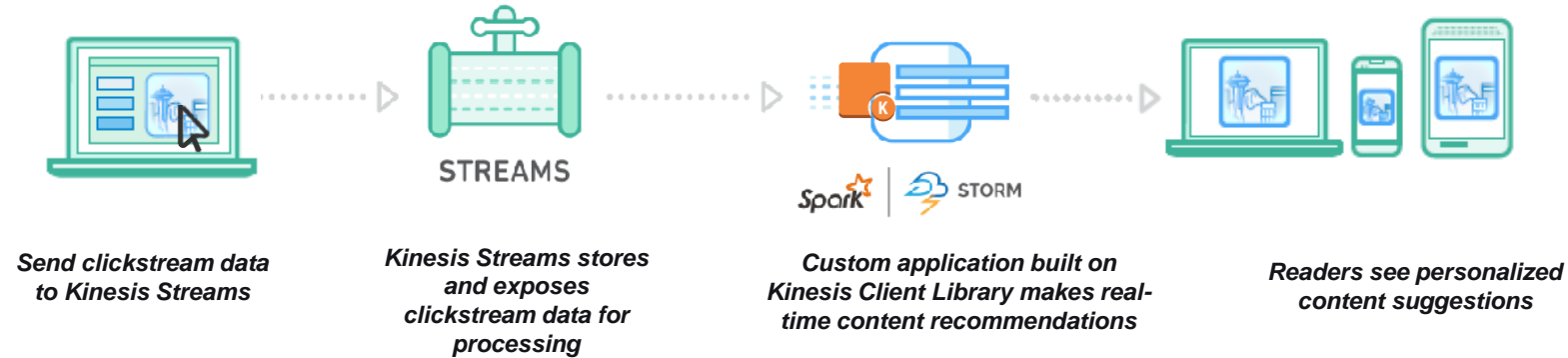
Easily analyze data streams using standard SQL queries



Amazon Kinesis Streams

Amazon Kinesis Streams

Build your own data streaming applications



Easy Administration: Create a new stream, set desired capacity and partitioning to match your data throughput rate and volume.

Build real-time applications: Perform custom record processing on streaming data using Kinesis Client Library, Apache Spark/ Storm, AWS Lambda, and more.

Low cost: Cost-efficient for workloads of any scale.



1 TB+/day game data
Analyzed in Real-Time



1 Billion Events/week from
Connected Devices



17 PB of Game Data
Per Season



80 Billion Ad Impressions per day
with 30 ms response time



100 GB/day Click
Streams from 250 sites



60 Billion Ad Impressions per
day sub-50 ms responses



17 million events day



1 Billion transaction per day

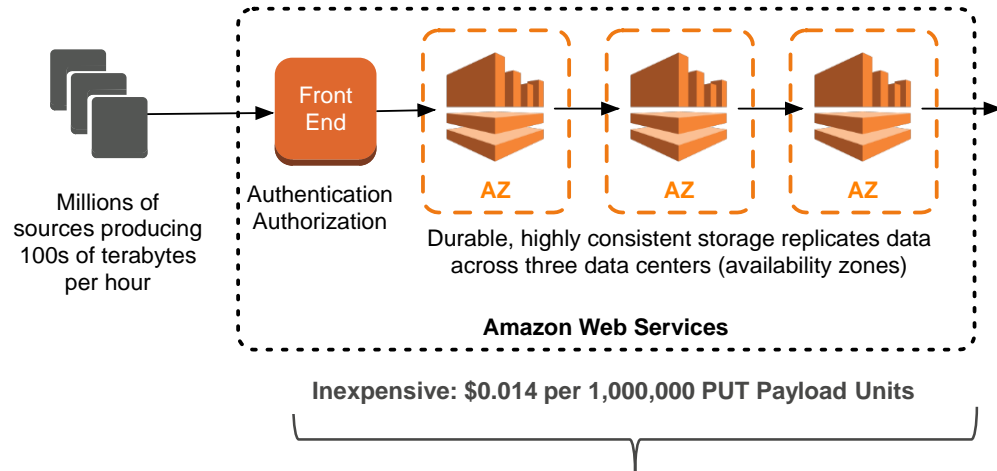


Amazon Kinesis Streams

Streaming Data Ingest and Storage

Amazon Kinesis Streams (re:Invent 2013)

Fully managed service for real-time processing of streaming data



Real-Time Streaming Data Ingestion

Putting Data into Kinesis

Simple Put* interface to capture and store data in Streams

A provisioned entity called a Stream composed of Shards

- Each shard provides 1MB/sec or 1,000 RPS of data ingress and provides 2MB/sec or 5 GetRecords TPS of data egress

Producers use a PUT call to store data in a Stream.

- Each record, up to 1 MB payload via **PutRecord** API call
- ~20ms latency (three copies, each in its own availability zone)

A partition key is supplied by producer and used to distribute (MD5 hash) the PUTs across (hash key range) of Shards

- Unique Sequence# returned upon successful PUT call
- Approximate arrival timestamp affixed to each record

Supports Different Data Ingestion Models

Workload determines partition key strategy

Managed Buffer

- Simply care about a reliable, scalable way to capture data
- Defer all processing to a generic consumer application
- Generate **random partition keys**
- Ensure a high cardinality for Partition Keys with respect to shards, to spray data evenly across available shards

Streaming Map-Reduce

- Streaming Map-Reduce: leverage partition keys as a natural way to aggregate data
 - e.g. partition key per customer, per Device_Id, per stock symbol, etc
- Implement specific consumer applications to process (reduce) data per partition key range.
- Design partition keys to scale and guard against “hot partition keys or shards”

Kinesis PutRecords API

High throughput API for efficient writes to Kinesis

- PutRecords {Records {Data,PartitionKey}, StreamName}
 - Supports 500 records.
 - Record can be up to 1 MB and up to 5 MB for whole request
 - Can include records with different partition keys
 - Ordering not guaranteed
- Successful response includes ShardID and SeqNumber values
- Unsuccessful Response

```
{
  "FailedRecordCount": number,
  "Records": [
    {
      "ErrorCode": "string",
      "ErrorMessage": "string",
      "SequenceNumber": "string",
      "ShardId": "string"
    }
  ]
}
```

Kinesis Producer Library

Highly configurable library to write to Kinesis

- Collects records and uses PutRecords for high throughput writes

```
KinesisProducerConfiguration config = new KinesisProducerConfiguration()
    .setRecordMaxBufferedTime(3000)
    .setMaxConnections(1)
    .setRequestTimeout(60000)
    .setRegion("us-west-1");

final KinesisProducer kinesisProducer = new KinesisProducer(config);
```

- Integrates seamlessly with the Amazon Kinesis Client Library (KCL) to de-aggregate batched records
- Submits Amazon CloudWatch metrics on your behalf to provide visibility into producer performance
- <https://github.com/aws-labs/amazon-kinesis-producer>

Extended Retention in Kinesis

New!


Default 24 Hours but configurable to 7 days

- 2 New APIs
 - `IncreaseStreamRetentionPeriod(String StreamName, int RetentionPeriodHours)`
 - `DecreaseStreamRetentionPeriod(String StreamName, int RetentionPeriodHours)`
- Use it in one of two-modes
 - As always-on extended retention
 - Raise retention period in response to operational event or planned maintenance of record processing application to extend retention.

Dealing with Provisioned Throughput Exceeded Metrics and Re-sharding (SplitShard/ MergeShard)

- Keep track of your stream metrics
- Monitor CloudWatch metrics: PutRecord.Bytes + GetRecords.Bytes metrics keep track of shard usage
- **Retry** if rise in input rate is temporary
- **Reshard** to increase number of shards
 - **SplitShard** – Adds more shards
 - **MergeShard** – Removes shards
 - Use the Kinesis Scaling Utility - <https://github.com/awslabs/amazon-kinesis-scaling-utils>

Metric	Units
PutRecords.Bytes	Bytes
PutRecords.Latency	Milliseconds
PutRecords.Success	Count
PutRecords.Records	Count
Incoming Bytes	Bytes
Incoming Records	Count



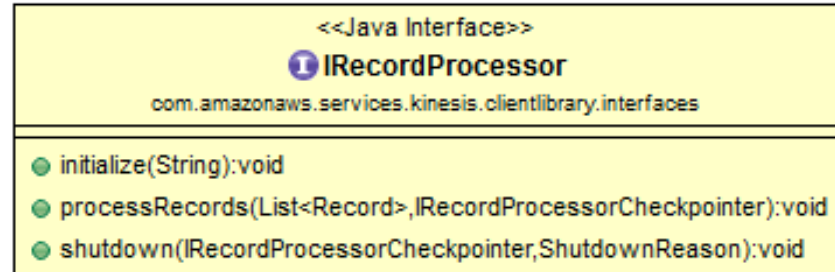
Amazon Kinesis Streams

Build Applications w/ Kinesis Client Library

Building Applications: **Kinesis Client Library**

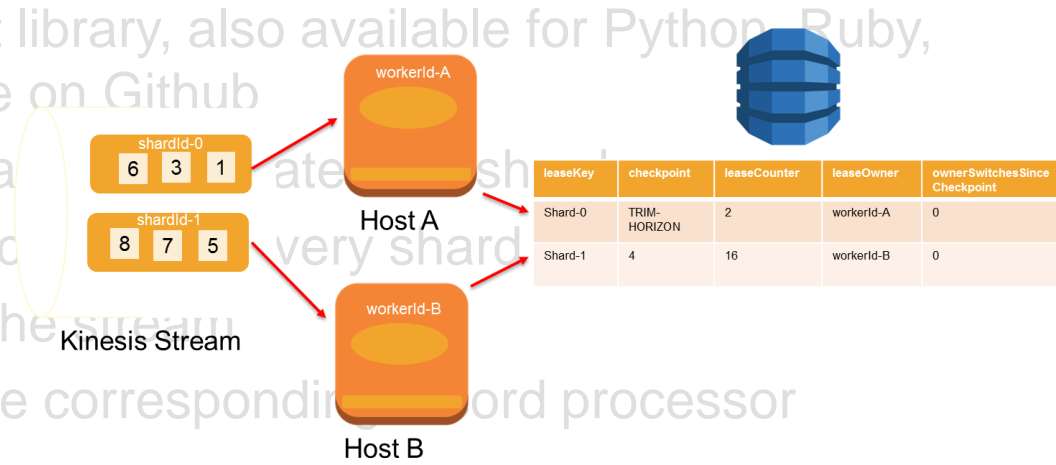
For **fault-tolerant, scalable** stream processing applications

- Open Source Java client library, also available for Python, Ruby, Node.JS, .NET. Available on Github
- Connects to the stream and enumerates the shards
- Instantiates a record processor for every shard it manages
- Pulls data records from the stream
- Pushes the records to the corresponding record processor



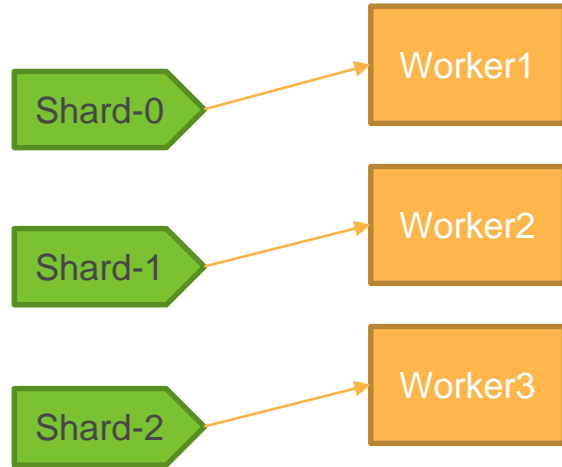
Building Applications: **Kinesis Client Library**

For **fault-tolerant, scalable** stream processing applications

- Open Source Java client library, also available for Python, Ruby, Node.JS, .NET. Available on Github
 - Connects to the stream and creates a worker instance for every shard
 - Instantiates a record processor for every shard
 - Pulls data records from the stream
 - Pushes the records to the corresponding record processor
- 
- The diagram illustrates the Kinesis Client Library architecture. On the left, a 'Kinesis Stream' contains two shards: 'shardId-0' with records [6, 3, 1] and 'shardId-1' with records [8, 7, 5]. Two worker instances, 'workerId-A' and 'workerId-B', are shown on 'Host A' and 'Host B' respectively. Red arrows indicate that workerId-A is processing shardId-0 and workerId-B is processing shardId-1. To the right, a table provides details for each shard's lease.
- | leaseKey | checkpoint | leaseCounter | leaseOwner | ownerSwitchesSince Checkpoint |
|----------|--------------|--------------|------------|-------------------------------|
| Shard-0 | TRIM-HORIZON | 2 | workerId-A | 0 |
| Shard-1 | 4 | 16 | workerId-B | 0 |

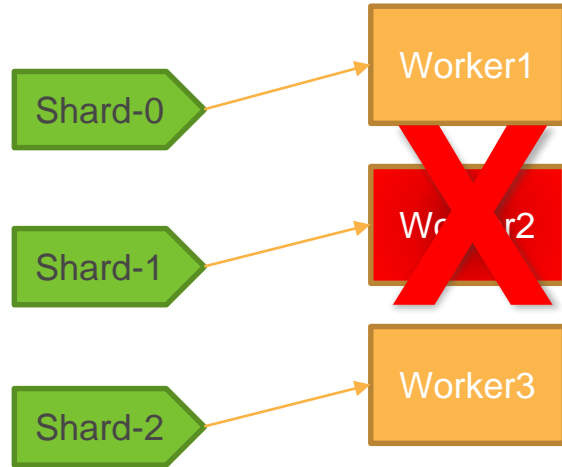
- Checkpoints processed records
- Balances shard-worker associations when the worker instance count changes
- Balances shard-worker associations when shards are split or merged

Worker Fail Over



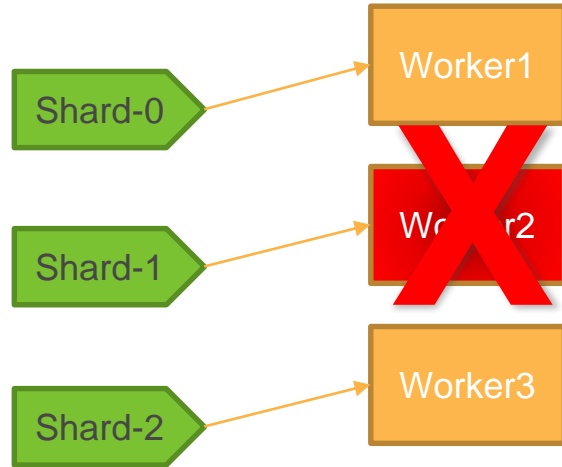
LeaseKey	LeaseOwner	LeaseCounter
Shard-0	Worker1	85
Shard-1	Worker2	94
Shard-2	Worker3	76

Worker Fail Over



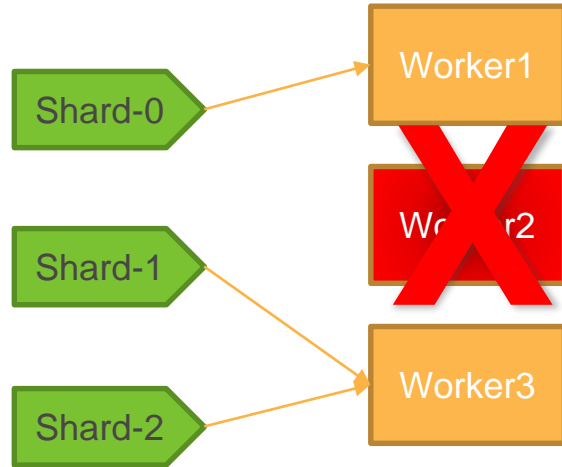
LeaseKey	LeaseOwner	LeaseCounter
Shard-0	Worker1	85 86
Shard-1	Worker2	94
Shard-2	Worker3	76 77

Worker Fail Over



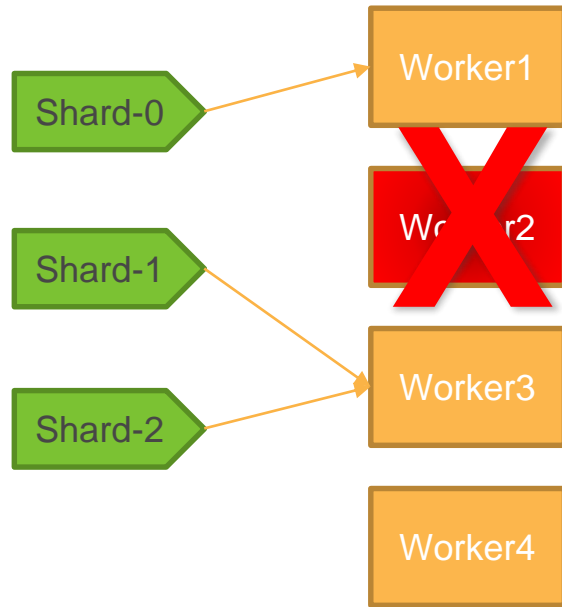
LeaseKey	LeaseOwner	LeaseCounter
Shard-0	Worker1	85 86 87
Shard-1	Worker2	94
Shard-2	Worker3	76 77 78

Worker Fail Over



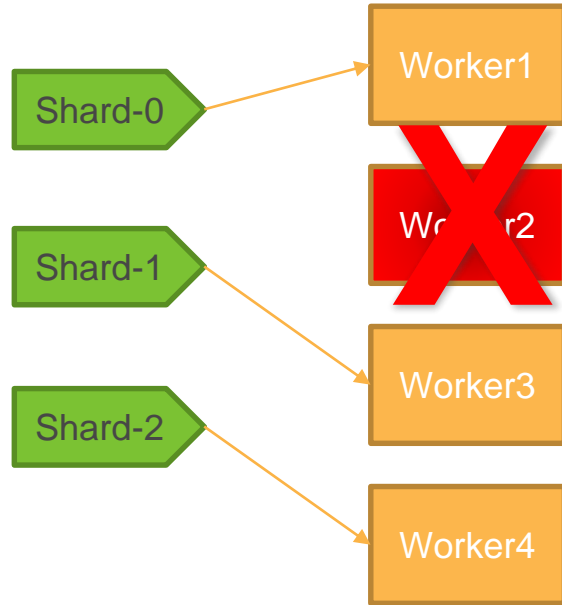
LeaseKey	LeaseOwner	LeaseCounter
Shard-0	Worker1	85 86 87 88
Shard-1	Worker3	94 95
Shard-2	Worker3	76 77 78 79

Worker Load Balancing



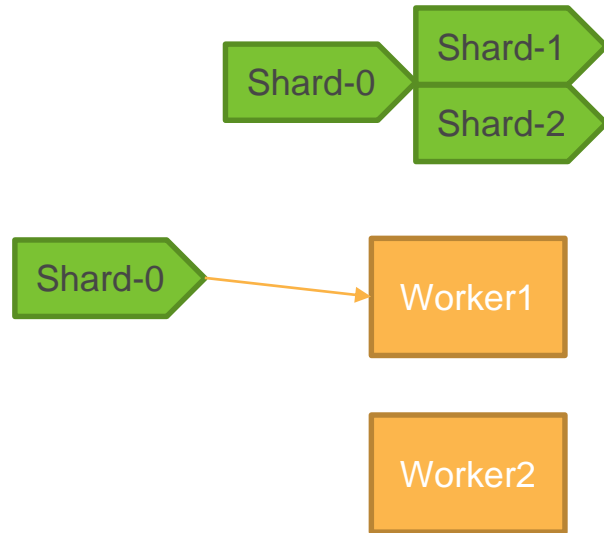
LeaseKey	LeaseOwner	LeaseCounter
Shard-0	Worker1	88
Shard-1	Worker3	96
Shard-2	Worker3	78

Worker Load Balancing



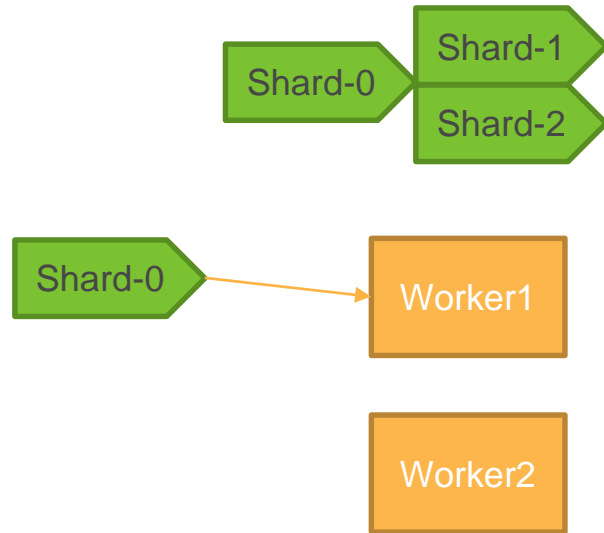
LeaseKey	LeaseOwner	LeaseCounter
Shard-0	Worker1	88
Shard-1	Worker3	96
Shard-2	Worker4	79

Resharding



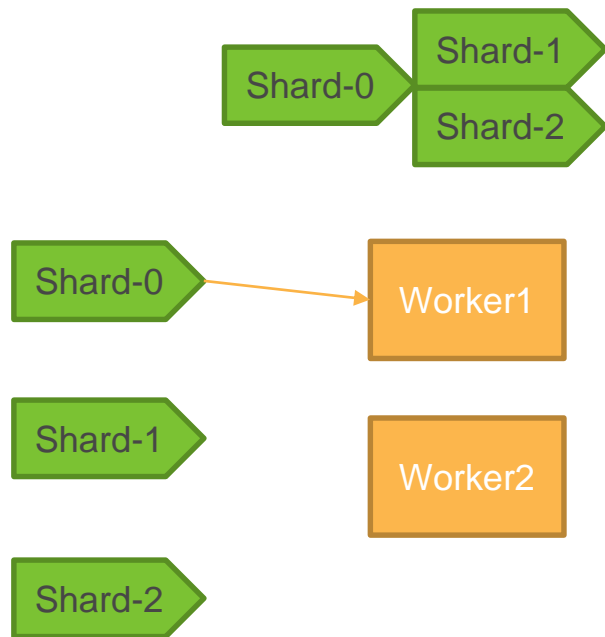
LeaseKey	LeaseOwner	LeaseCounter	checkpoint
Shard-0	Worker1	88	100

Resharding



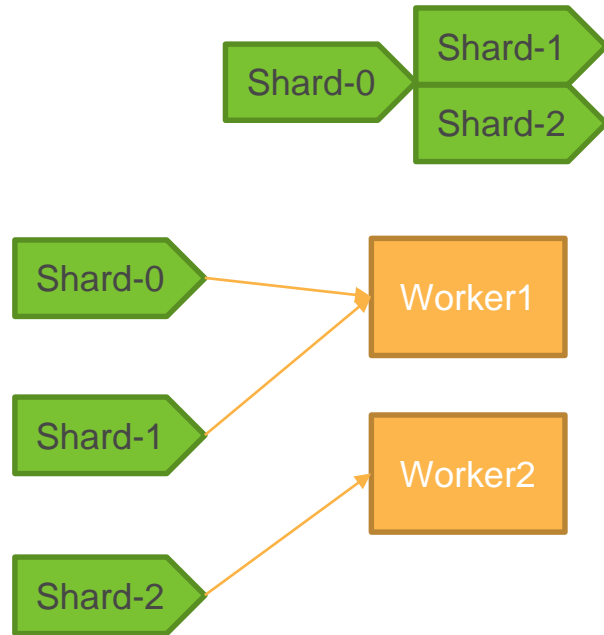
LeaseKey	LeaseOwner	LeaseCounter	checkpoint
Shard-0	Worker1	90	SHARD_END

Resharding



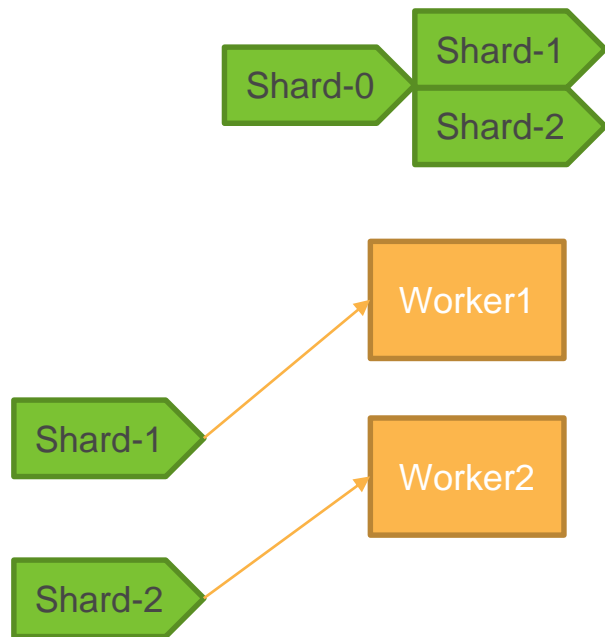
LeaseKey	LeaseOwner	LeaseCounter	checkpoint
Shard-0	Worker1	90	SHARD_END
Shard-1		0	TRIM_HORIZON
Shard-2		0	TRIM_HORIZON

Resharding



LeaseKey	LeaseOwner	LeaseCounter	checkpoint
Shard-0	Worker1	90	SHARD_END
Shard-1	Worker1	2	TRIM_HORIZON
Shard-2	Worker2	3	TRIM_HORIZON

Resharding



LeaseKey	LeaseOwner	LeaseCounter	checkpoint
Shard-1	Worker1	2	TRIM_HORIZON
Shard-2	Worker2	3	TRIM_HORIZON

Sending & Reading Data from Kinesis Streams

Sending

AWS SDK



Kinesis
Producer
Library



AWS Mobile
SDK



LOG4J



Flume



Fluentd



Consuming

Get* APIs



Kinesis Client Library
+
Connector Library



AWS Lambda



Amazon Elastic
MapReduce



Apache
Storm



Apache
Spark



Amazon Kinesis Firehose

Load massive volumes of streaming data into Amazon S3 and Amazon Redshift



Capture and submit streaming data to Firehose

Firehose loads streaming data continuously into S3 and Redshift

Analyze streaming data using your favorite BI tools

Zero administration: Capture and deliver streaming data into S3, Redshift, and other destinations [without writing an application](#) or [managing infrastructure](#).

Direct-to-data store integration: [Batch](#), [compress](#), and [encrypt](#) streaming data for delivery into data destinations [in as little as 60 secs](#) using simple configurations.

Seamless elasticity: Seamlessly scales to match data throughput w/o intervention

Amazon Kinesis Analytics *(preannounced, beta)*

Analyze data streams continuously with standard SQL



*Connect to Kinesis streams,
Firehose delivery streams*

*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*

Apply SQL on streams: Easily connect to data streams and apply existing SQL skills.

Build real-time applications: Perform continual processing on streaming big data with sub-second processing latencies

Scale elastically: Elastically scales to match data throughput without any operator intervention.

Create New Application



AWS

Kinesis

Amazon Kinesis Analytics > Create New Application

Step 1: Name Application

Step 2: Configure Stream Source

Step 3: Configure Application

Step 4: Configure Destination

Step 5: Review Application

Name Application

Name*

Description

Next

Select Source



AWS

Kinesis

Amazon Kinesis Analytics > Create New Application

Step 1: Name Application

Step 2: Configure Stream Source

Step 3: Configure Application

Step 4: Configure Destination

Step 5: Review Application

Configure Stream Source

Choose an input source for your Amazon Kinesis Analytics application.

Input source type*

- Amazon Kinesis Streams
- Amazon Kinesis Firehose

Previous

Next

Configure Source

Configure Input View

Your data will be turned into a schema called "INPUTVIEW" for you to write SQL statements against.

Input format* [JSON](#) [edit](#)

We have detected your data format to be JSON. Edit this setting if it is incorrect.

Input View

FORMATTED **RAW**

```
{
  "entry": {
    "DATATIMESTAMP": "2015-09-24 15:18:51.321",
    "VEHICLEID": "fbeafaf8-14ad-41b0-89c7-75144ee024ae",
    "LATITUDE": 47.69211959838867,
    "LONGITUDE": -122.11260223388672,
    "DESTINATIONLATITUDE": 47.539215087890625,
    "DESTINATIONLONGITUDE": -122.09043884277344,
    "FUELEFFICIENCY": 69.00909423828125,
    "SPEED": 74.439453125,
    "PARTITION_ID": "0.2359220474611321",
    "SEQUENCE_NUMBER": "49554725858818738486899606234253502759895155788828639234",
    "SHARD_ID": "shardId-000000000000"},
  "entry": {
    "DATATIMESTAMP": "2015-09-24 15:18:51.322",
    "VEHICLEID": "f23f08c6-438e-4232-8a56-a412ea191ab5",
    "LATITUDE": 47.67170715332031,
    "LONGITUDE": -122.00933074951172,
    "DESTINATIONLATITUDE": 47.577919006347656,
    "DESTINATIONLONGITUDE": -122.00635528564453,
    "FUELEFFICIENCY": 38.60463333129883,
    "SPEED": 7.285689353942871,
    "PARTITION_ID": "0.2359220474611321",
    "SEQUENCE_NUMBER": "49554725858818738486899606234253502759895155788828639234",
    "SHARD_ID": "shardId-000000000000"},
  "entry": {
    "DATATIMESTAMP": "2015-09-24 15:18:51.321",
    "VEHICLEID": "24601b4a-b6ae-478d-86f7-9db0ad5f2d03",
    "LATITUDE": 47.50821304321289,
    "LONGITUDE": -122.01436614990234,
    "DESTINATIONLATITUDE": 47.619258880615234,
    "DESTINATIONLONGITUDE": -122.20710754394531,
    "FUELEFFICIENCY": 42.48281478881836,
    "SPEED": 140.18995666503906,
    "PARTITION_ID": "0.2359220474611321",
    "SEQUENCE_NUMBER": "49554725858818738486899606234253502759895155788828639234",
    "SHARD_ID": "shardId-000000000000"},
  "entry": {
    "DATATIMESTAMP": "2015-09-24 15:18:51.321",
    "VEHICLEID": "4240ed0c-
```

Configure Source – Schema Discovery

Configure Input View

Your data will be turned into a schema called "INPUTVIEW" for you to write SQL statements against.

Input format* **JSON** [edit](#)

We have detected your data format to be JSON. Edit this setting if it is incorrect.

Input View

FORMATTED **RAW**

Filter by column header

DATATIMESTAMP VARCHAR (512)	VEHICLEID VARCHAR (512)	LATITUDE DOUBLE	LONGITUDE DOUBLE	DESTINA DOUBLE
2015-09-24 15:18:51.321	fbeafaf8-14ad-41b0-89c7-75144ee024ae	47.69211959838867	-122.11260223388672	47.53921
2015-09-24 15:18:51.322	f23f08c6-438e-4232-8a56-a412ea191ab5	47.67170715332031	-122.00933074951172	47.57791
2015-09-24 15:18:51.321	24601b4a-b6ae-478d-86f7-9db0ad5f2d03	47.50821304321289	-122.01436614990234	47.61925
2015-09-24 15:18:51.321	4240ed0c-2ae4-448f-96d3-9ad4c5f399c5	47.655616760253906	-122.06053161621094	47.53152
2015-09-24 15:18:51.321	c0452f21-918d-4727-81f0-3c6609333841	47.577186584472656	-122.28516387939453	47.59902
2015-09-24 15:18:51.321	35365f97-ca9e-4414-b61a-5dae17c12a41	47.50307846069336	-122.2349853515625	47.51128
2015-09-24 15:18:51.322	d29e6995-9cc5-4ee2-ac84-d0fdccceb5803	47.552005767822266	-122.03602600097656	47.66856

Configure the Application

Configure Application

Your application is expressed in ANSI-SQL. Select a sample processing code that will process your data stream.

Filter by blueprint

Tumbling Time Window

Calculates results over a window that does not overlap.

Running Record Count

Counts all records over the previous hour.

Streaming Transformation

Transforms the stream by adding, dropping, recasting, renaming, augmenting, etc., columns.

Test SQL

Counts all records over the previous hour, grouped by colName. For each row, show the number of rows orders in the previous minute to the same column, and the total number of orders in the previous minute.

```
1 CREATE OR REPLACE VIEW "OUTPUTVIEW" AS
2 SELECT STREAM colName, count(*) OVER lastMinute AS requests
3 FROM "INPUTVIEW"
4 WINDOW lastMinute AS (PARTITION BY colName RANGE INTERVAL '1' MINUTE PRECEDING)
```

Configure Destination

Configure Destination

Choose an output destination, role, and format for your Amazon Kinesis Analytics application. The processed stream data will be sent to your destination in the selected format. A role provides your application access to your output destination through AWS Identity and Access Management (IAM).

Output destination type

Firehose DeliveryStream ⓘ

Role* ⓘ

Output format*

Sample output

Sample output records go here...

Test output

Previous

Next

Review your Application

Review Application

Step 1: Name Application

[Edit](#)

Name myApplication

Description This application rocks!

Step 2: Configure Stream Source

[Edit](#)

Input source type Kinesis Stream

Input source kinesis-stream-mobile

Input role streamanalytics-deliverystream-role

Input time Latest

Input format JSON

Step 3: Configure Application

[Edit](#)

Blueprint Running Record Count

Step 4: Configure Destination

[Edit](#)

Output destination type Firehose DeliveryStream

Output destination kinesis-deliverystream-test

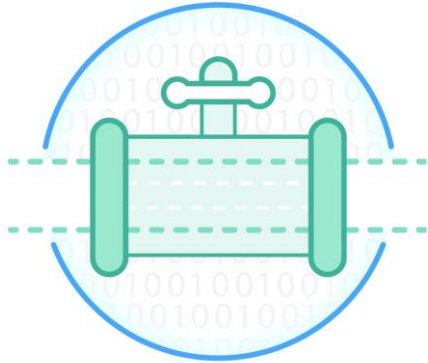
Role streamanalytics-deliverystream-role

Output format CSV

[Save & Run Application](#)[Save Application](#)

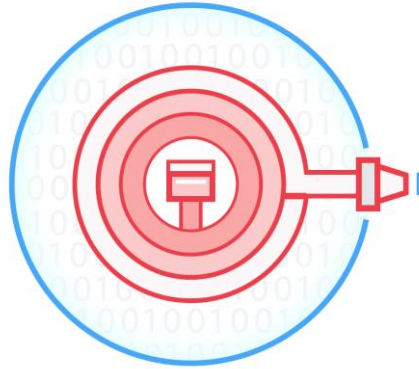
Amazon Kinesis: Streaming data made easy

Services make it easy to capture, deliver, and process streams on AWS



Amazon Kinesis Streams

Build your own custom applications that process or analyze streaming data



Amazon Kinesis Firehose

Easily load massive volumes of streaming data into Amazon S3 and Redshift



Amazon Kinesis Analytics

Easily analyze data streams using standard SQL queries

