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

Inexpensive: $0.028 per million PUTs

Inexpensive: $0.014 per 1,000,000 PUT Payload Units

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
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

```json
{
  "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

```java
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/awslabs/amazon-kinesis-producer](https://github.com/awslabs/amazon-kinesis-producer)
Extended Retention in Kinesis
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
- **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 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.
- 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

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>85</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
<td>94</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76</td>
</tr>
</tbody>
</table>
Worker Fail Over

<table>
<thead>
<tr>
<th>Shard-0</th>
<th>Worker1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>85 86</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
<td>94</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76 77</td>
</tr>
</tbody>
</table>
Worker Fail Over

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>85 86 87</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
<td>94</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76 77 78</td>
</tr>
</tbody>
</table>
Worker Fail Over

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>85 86 87 88</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker3</td>
<td>94 95</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76 77 78 79</td>
</tr>
</tbody>
</table>
Worker Load Balancing

<table>
<thead>
<tr>
<th>Shard</th>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td></td>
<td>Worker1</td>
<td>88</td>
</tr>
<tr>
<td>Shard-1</td>
<td></td>
<td>Worker3</td>
<td>96</td>
</tr>
<tr>
<td>Shard-2</td>
<td></td>
<td>Worker3</td>
<td>78</td>
</tr>
</tbody>
</table>

Worker1
Worker2
Worker3
Worker4
Worker Load Balancing

<table>
<thead>
<tr>
<th>Shard-0</th>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worker1</td>
<td>Shard-0</td>
<td>Worker1</td>
<td>88</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Shard-1</td>
<td>Worker3</td>
<td>96</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Shard-2</td>
<td>Worker4</td>
<td>79</td>
</tr>
</tbody>
</table>

Worker1 and Worker2 are marked with an 'X' to indicate that they are currently handling the load of Shard-0.
Resharding

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>88</td>
<td>100</td>
</tr>
</tbody>
</table>
Resharding

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>90</td>
<td>SHARD_END</td>
</tr>
</tbody>
</table>
Resharding

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>90</td>
<td>SHARD_END</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>0</td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker1</td>
<td>0</td>
<td>TRIM_HORIZON</td>
</tr>
</tbody>
</table>
Resharding

<table>
<thead>
<tr>
<th>ShardKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>90</td>
<td>SHARD_END</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>2</td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker2</td>
<td>3</td>
<td>TRIM_HORIZON</td>
</tr>
</tbody>
</table>
Resharding

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>2</td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker2</td>
<td>3</td>
<td>TRIM_HORIZON</td>
</tr>
</tbody>
</table>

1. Shard-0
2. Shard-1
3. Shard-2

Worker1

Worker2
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

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

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

Amazon Kinesis Analytics ➔ Create New Application

Step 1: Name Application

Name Application

Name: myApplication
Description: This application rocks!

Next
Configure Stream Source

Choose an input source for your Amazon Kinesis Analytics application.

Input source type:

- Amazon Kinesis Streams
- Amazon Kinesis Firehose
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

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

Input View

Json schema for the input format:

```json
{
    "entry": {
        "DATETIME`: "2015-09-24 15:18:51.321", "VEHICLEID": "fbc9afa8-14ad-41b0-89c7-751446e0824ae", "LATITUDE": 47.69211959838867, "LONGITUDE": -122.11260223388672, "DESTINATIONLATITUDE": 47.539215087890025, "DESTINATIONLONGITUDE": -122.09043804277344, "FUELEFFICIENCY": 0.09.099942432682125, "SPEED": 74.439453125, "PARTITION_ID": "0.2599220474611321", "SEQUENCE_NUMBER": "495547258581873888689966634253502759895155788828639234", "SHARD_ID": "shardId-000000000000", "ENTRY": {
            "DATETIME": "2015-09-24 15:18:51.321", "VEHICLEID": "fbc9afa8-14ad-41b0-89c7-751446e0824ae", "LATITUDE": 47.69211959838867, "LONGITUDE": -122.11260223388672, "DESTINATIONLATITUDE": 47.539215087890025, "DESTINATIONLONGITUDE": -122.09043804277344, "FUELEFFICIENCY": 0.09.099942432682125, "SPEED": 74.439453125, "PARTITION_ID": "0.2599220474611321", "SEQUENCE_NUMBER": "495547258581873888689966634253502759895155788828639234", "SHARD_ID": "shardId-000000000000", "ENTRY": {
```
```
**Configure Source – Schema Discovery**

**Configure Input View**

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

<table>
<thead>
<tr>
<th>DATETIME STAMP</th>
<th>VEHICLEID (512)</th>
<th>LATITUDE DOUBLE</th>
<th>LONGITUDE DOUBLE</th>
<th>DESTINATION DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2015-09-24 15:18:51:321</td>
<td>fbeafa8-141b0-89c7-75144ae024ae</td>
<td>47.6921195983887</td>
<td>-122.112602233867</td>
<td>47.53921</td>
</tr>
<tr>
<td>2015-09-24 15:18:51:322</td>
<td>f230f85c-439e-4232-8a56-a4f12e191ab5</td>
<td>47.6710715332031</td>
<td>-122.0933074951172</td>
<td>47.57791</td>
</tr>
<tr>
<td>2015-09-24 15:18:51:321</td>
<td>24601b4a-b6ae-478d-8617-9dbb0ad52f03</td>
<td>47.50821304321289</td>
<td>-122.0143681490234</td>
<td>47.61925</td>
</tr>
<tr>
<td>2015-09-24 15:18:51:321</td>
<td>4240ed3c-2ae4-448f-95c3-9ad4c5f399c5</td>
<td>47.65561676053906</td>
<td>-122.06053161261094</td>
<td>47.53152</td>
</tr>
<tr>
<td>2015-09-24 15:18:51:321</td>
<td>c0452f21-918d-4727-810-3c560533341</td>
<td>47.57186584472656</td>
<td>-122.28516387093453</td>
<td>47.59902</td>
</tr>
<tr>
<td>2015-09-24 15:18:51:321</td>
<td>35366f8f-ca9e-4414-b61a-6d1e17c12a11</td>
<td>47.50307846069336</td>
<td>-122.2349853516625</td>
<td>47.51128</td>
</tr>
<tr>
<td>2015-09-24 15:18:51:322</td>
<td>d29e6995-9cc5-4ee2-ac84-d0fdcceeb5803</td>
<td>47.552000767822256</td>
<td>-122.03602600097656</td>
<td>47.66856</td>
</tr>
</tbody>
</table>
Configure Application

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

- **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

```
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

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).

<table>
<thead>
<tr>
<th>Output destination type</th>
<th>Amazon Kinesis Firehose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firehose DeliveryStream</td>
<td>kinesis-deliverystream-test</td>
</tr>
<tr>
<td>Role</td>
<td>streamanalytics-deliverystream-role</td>
</tr>
<tr>
<td>Output format</td>
<td>CSV</td>
</tr>
</tbody>
</table>

Sample output: Sample output records go here...

Test output
# Review your Application

## Review Application

**Step 1: Name Application**
- **Name**: myApplication
- **Description**: This application rocks!

**Step 2: Configure Stream Source**
- **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**
- **Blueprint**: Running Record Count

**Step 4: Configure Destination**
- **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