Processing Big Data in Motion
Streaming Data Ingestion and Processing

Roger Barga, General Manager, Kinesis Streaming Services, AWS

April 7, 2016
Riding the Streaming Rapids

Microsoft Research

2007 & 2008
2009
2010
2011
2012
2013
2015
2016

Amazon Kinesis Streams
Amazon Kinesis Firehose
Amazon Kinesis Analytics

Streaming Map Reduce & Machine Learning over Streams

Azure Stream Analytics

Relational Semantics and Implementation

Complex Event Processing over Streaming Data

Microsoft SQL Server StreamInsight

CIDR 2009

Over Streaming Data

SQL
Interest in and demand for stream data processing is rapidly increasing...
Most data is *produced continuously*.


```json
{
  "payerId": "Joe",
  "productCode": "AmazonS3",
  "clientProductCode": "AmazonS3",
  "usageType": "Bandwidth",
  "operation": "PUT",
  "value": "22490",
  "timestamp": "1216674828"
}
```

**Common Log Entry**

```
<165>1 2003-10-11T22:14:15.003Z mymachine.example.com evntslog - ID47 [exampleSDID@32473 iut="3" eventSource="Application" eventID="1011"] [examplePriority@32473 class="high"]
```

**MQTT Record**

```
<R,AMZN,T,G,R1>
```

**NASDAQ OMX Record**

```
"SeattlePublicWater/Kinesis/123/Realtime" – 412309129140
```

**Syslog Entry**
Why?

Time is money

• *Perishable Insights (Forrester)*

  - Hourly server logs: how your systems were misbehaving an hour ago
  - Weekly / Monthly Bill: What you spent this past billing cycle?
  - Daily fraud reports: tells you if there was fraud yesterday
  - CloudWatch metrics: what just went wrong now
  - Real-time spending alerts/caps: guaranteeing you can’t overspend
  - Real-time detection: blocks fraudulent use now
Why?

Time is money

- *Perishable Insights (Forrester)*
- *A more efficient implementation*
- *Most ‘Big Data’ deployments process continuously generated data (batched)*
Why?

Availability

Variety of stream data processing systems, active ecosystem but still early days…
Why?

Disruptive

Foundational for business critical workflows

Enable new class of applications & services that process data continuously.
Need to begin thinking about applications & services in terms of **streams of data** and continuous processing.

*A change in perspective is worth 80 IQ points...*  
– Alan Kay
Agenda

• Scalable & Durable Data Ingest
  ▪ A quick word on our motivation
  ▪ Kinesis Streams, through a simple example

• Continuous Stream Data Processing
  ▪ Kinesis Client Library (KCL)
  ▪ One select design challenge: dynamic resharding
  ▪ How customers are using Kinesis Streams today

• Building on Kinesis Streams
  ▪ Kinesis Firehose
  ▪ AWS Event Driven Computing
Our Motivation for Continuous Processing

AWS Metering service
• 100s of millions of billing records per second
• Terabytes++ per hour
• Hundreds of thousands of sources
• For each customer: gather all metering records & compute monthly bill
• Auditors guarantee 100% accuracy at months end

Seem perfectly reasonable to run as a batch, but relentless pressure for realtime…

With a Data Warehouse to load
• 1000s extract-transform-load (ETL) jobs every day
• Hundreds of thousands of files per load cycle
• Thousands of daily users, hundreds of queries per hour
Our Motivation for Continuous Processing

AWS Metering service
• 100s of millions of billing records per second
• Terabytes++ per hour
• Hundreds of thousands of sources
• For each customer: gather all metering records & compute monthly bill
• Auditors guarantee 100% accuracy at months end

Other Service Teams, Similar Requirements
• CloudWatch Logs and CloudWatch Metrics
• CloudFront API logging
• ‘Snitch’ internal datacenter hardware metrics
Real-time Ingest
- Highly Scalable
- Durable
- Replayable Reads

Continuous Processing
- Support multiple simultaneous data processing applications
- Load-balancing incoming streams, scale out processing
- Fault-tolerance, Checkpoint / Replay

Amazon Kinesis
Example application
twitter-trends.com website

Elastic Beanstalk
twitter-trends.com
Too big to handle on one box
The solution: streaming map/reduce
Core concepts

- **Data record**
- **Stream**
- **Partition key**
- **Shard**
- **Worker**

**Twitter Trends**

- **My top-10**
- **Global top-10**

**Sequence number**

**Shard:** 14, 17, 18, 21, 23
How this relates to Kinesis
Kinesis Streaming Data Ingestion

- Streams are made of **Shards**
- Each Shard ingests data up to 1MB/sec, and up to 1000 TPS
- Producers use a PUT call to store data in a Stream: `PutRecord {Data, PartitionKey, StreamName}`
- Each Shard emits up to 2 MB/sec
- All data is stored for **24 hours, 7 days** if extended retention is ‘ON’
- **Scale** Kinesis streams by adding or removing Shards
- **Replay** data from retention period
Amazon Web Services

- Durable, highly consistent storage replicates data across three data centers (availability zones)

Real-Time Streaming Data Ingestion

- Inexpensive: $0.028 per million PUTs
- $0.014 per 1,000,000 PUT Payload Units

- Millions of sources producing 100s of terabytes per hour

- 25–40ms
- 100–150ms

- Ordered stream of events supports multiple readers

- Aggregate and archive to S3
- Real-time dashboards and alarms
- Machine learning algorithms or sliding window analytics
- Aggregate analysis in Hadoop or a data warehouse

Custom-built Streaming Applications (KCL)
Kinesis Client Library
Using the Kinesis API directly

twitter-trends.com
Using the Kinesis API directly

```java
iterator = getShardIterator(shardId, LATEST);
while (true) {
    [records, iterator] =
        getNextRecords(iterator, maxRecsToReturn);
    process(records);
}

process(records): {
    for (record in records) {
        updateLocalTop10(record);
    }
    if (timeToDoOutput()) {
        writeLocalTop10ToDDB();
    }
}

while (true) {
    localTop10Lists =
        scanDDBTable();
    updateGlobalTop10List(
        localTop10Lists);
    sleep(10);
}
```
Challenges with using the Kinesis API directly

- Manual creation of workers and assignment to shards

Kinesis application

twitter-trends.com
Challenges with using the Kinesis API directly

How many workers per EC2 instance?

Kinesis application

twitter-trends.com
Challenges with using the Kinesis API directly

Kinesis application

How many EC2 instances?

twitter-trends.com
Using the Kinesis Client Library
Elasticity and Load Balancing
Elasticity and Load Balancing

KINESIS

Auto scaling Group

Shard mgmt table

twitter-trends.com
Elasticity and Load Balancing
Elasticity and Load Balancing
Fault Tolerance Support

Availability Zone 1

Availability Zone 3

Shard mgmt table

twitter-trends.com
Fault Tolerance Support

KINESIS

Availability Zone 1

Availability Zone 3

Shard mgmt table

twitter-trends.com
Fault Tolerance Support

[Diagram showing Kinesis with availability zones 1 and 3, Shard mgmt table, twitter-trends.com, and arrows indicating data flow]
Fault Tolerance Support
Worker Fail Over

| Shard-0  | Worker1 |
| Shard-1  | Worker2 |
| Shard-2  | Worker3 |

<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>Shard</th>
<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>
<td></td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76 77 78</td>
<td></td>
</tr>
</tbody>
</table>
Worker Fail Over

Shard-0 → Worker1

Shard-1 → Worker2

Shard-2 → Worker3

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

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>88</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker3</td>
<td>96</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker4</td>
<td>79</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>Shard</th>
<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></td>
<td>SHARD_END</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>2</td>
<td></td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker2</td>
<td>3</td>
<td></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>
Putting this into production

**Cost & Scale**

<table>
<thead>
<tr>
<th>Tweet Volume</th>
<th>Cost Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>500MM tweets/day = ~ 5,800 tweets/sec</td>
<td>$0.015/hour per shard, $0.014/million PUTS</td>
</tr>
<tr>
<td>2k/tweet is ~12MB/sec (~1TB/day)</td>
<td>Kinesis cost is $0.47/hour</td>
</tr>
<tr>
<td></td>
<td>Redshift cost is $0.850/hour (for a 2TB node)</td>
</tr>
<tr>
<td></td>
<td>Total: $1.32/hour</td>
</tr>
</tbody>
</table>
Design Challenge(s)

- Dynamic Resharding & Scale Out
- Enforcing Quotas (think proxy fleet with 1Ks servers)
- Distributed Denial of Service Attack (unintentional)
- Dynamic Load Balancing on Storage Servers
- Heterogeneous Workloads (tip of stream vs 7 day)
- Optimizing Fleet Utilization (proxy, control, data planes)
- Avoid Scaling Cliffs
- …
Kinesis Streams: Streaming Data the AWS Way

- Easy to provision, deploy, and manage
- Elastically scalable
- Real-time latencies
- Choose the service, or combination of services, for your specific use cases.
- Pay as you go, no up front costs
Sushiro: Kaiten Sushi Restaurants

380 stores stream data from sushi plate sensors and stream to Kinesis
Sushiro: Kaiten Sushi Restaurants

380 stores stream data from sushi plate sensors and stream to Kinesis
Real-Time Streaming Data with Kinesis Streams

1 TB+/day game data analyzed in real-time | Gaming

5 billion events/wk from connected devices | IoT

17 PB of game data per season | Entertainment

100 billion ad impressions/day, 30 ms response time | Ad Tech

100 GB/day click streams
250+ sites | Enterprise

50 billion ad impressions/day sub-50 ms responses | Ad Tech

17 million events/day | Technology

1 billion transactions per day | Bitcoin
Streams provide a foundational abstraction on which to build higher level services.
Amazon Kinesis Firehose

- **Zero Admin**: 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 S3, and other destinations in as little as 60 secs, set up in minutes.

- **Seamless elasticity**: Seamlessly scales to match data throughput.
Amazon Kinesis Firehose
Fully Managed Service for Delivering Data Streams into AWS Destinations

Data Sources
- AWS IoT
- Data Sources
- All AWS Platform SDKs
- Mobile SDKs
- Kinesis Agent

AWS Endpoint
- No Partition Keys
- No Provisioning
- End to End Elastic

[Batch, Compress, Encrypt]

AWS

S3

Redshift
AWS Event-Driven Computing

• Compute in response to recently occurring events
• Newly arrived/changed data
  – **Example**: generate thumbnail for an image uploaded to S3
• Newly occurring system state changes
  – **Example**: EC2 instance created
  – **Example**: DynamoDB table deleted
  – **Example**: Auto-scaling group membership change
  – **Example**: RDS-HA primary fail-over occurs
Event Driven Computing in AWS Today

S3 event notifications

[Diagram showing S3 buckets and SQS queue]
Event Driven Computing in AWS Today

DynamoDB Update Streams
Event Driven Computing in AWS Today

Cloudtrail event log for API calls

- S3
- Customer 1
- Customer 2
- Customer 3
Event Driven Computing in AWS Tomorrow

Single Event logs for asynchronous service events

Customer 1
Customer 2
Customer 3
Event Driven Computing in AWS Tomorrow

Event logs for asynchronous service events

Event logs from other data storage services
A Unified Event Log Approach

SQS
(Unordered Events)

Kinesis
(Ordered Events)
Ordered Event Log Using Kinesis Streams and the Kinesis Client Library

Use of the KCL
Mostly writing business logic

EDC Rules Language
Simple CloudWatch actions in response to matching rules.
Event Logs for Customers’ Services

**Vision:** customers’ services and applications leverage the AWS event log infrastructure

- Customers 2593, 7302, 3826
- Widget A, Widget B, Widget C
- www.widget.com

Per-customer control plane events sent to customer’s unified control plane log

Per-entity data plane event logs
Streaming data is highly prevalent and relevant; Stream data processing is on the rise; A key part of business critical workflows today, a powerful abstraction for building a new class of applications & data intensive services tomorrow. A rich area for distributed systems, programming model, IoT, and new service(s) research.
Questions