Real-time Streaming Applications on AWS – Patterns and Use Cases

Paul Armstrong - Solutions Architect (AWS)
Tom Seddon - Data Engineering Tech Lead (Deliveroo)

28th June 2017
What to Expect from the session

• Core streaming processing use cases
• Overview of AWS services that can help you solve these use cases including all Amazon Kinesis Streams, Kinesis Analytics, Kinesis Firehose, AWS Lambda, and Amazon EMR
• Deep dive into some core stream processing using: streaming ingest-transform-load, continuous metric generation, and responsive analysis
# It’s All About the Pace

<table>
<thead>
<tr>
<th>Batch Processing</th>
<th>Stream Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hourly server logs</td>
<td>Real-time metrics</td>
</tr>
<tr>
<td>Weekly or monthly bills</td>
<td>Real-time spending alerts/caps</td>
</tr>
<tr>
<td>Daily web-site clickstream</td>
<td>Real-time clickstream analysis</td>
</tr>
<tr>
<td>Daily fraud reports</td>
<td>Real-time detection</td>
</tr>
</tbody>
</table>
Simple Pattern for Streaming Data

**Data Producer**
- Continuously creates data
- Continuously writes data to a stream
- Can be almost anything

**Streaming Storage**
- Durably stores data
- Provides temporary buffer
- Supports very high-throughput

**Data Consumer**
- Continuously processes data
- Cleans, prepares, & aggregates
- Transforms data to information

- Mobile Client
- Amazon Kinesis
- Amazon Kinesis app
Why streaming?

Decouple producers & consumers
Persistent buffer
Collect multiple streams

Preserve ordering
Parallel consumption
Streaming MapReduce

Amazon Kinesis stream

Producer A: Key = orange
Producer B: Key = blue
Producer C: Key = green
Producer D: Key = yellow

Consumer App 1: Counts
Consumer App 2: Sums
Important Services for Streaming Data Applications
Amazon Kinesis: Streaming Data Made Easy

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

Amazon Kinesis Streams
Amazon Kinesis Firehose
Amazon Kinesis Analytics

SQL
Amazon Kinesis Streams

- Easy administration
- Build real time applications with framework of choice
- Low cost
Amazon Kinesis Client Library

• Build applications with Kinesis Client Library (KCL) in Java, Ruby, Python, or Node.JS
• Deploy on your EC2 instances
• Three primary components:
  1. **Worker** – Processing unit that maps to each application instance
  2. **Record Processor Factory** – Creates the record processor
  3. **Record Processor** – Processor unit that processes data from a shard in Amazon Kinesis Streams
State Management with Kinesis Client Library

- One record processor maps to one shard and processes data records from that shard
- One worker maps to one or more record processors
- Balances shard-worker associations when worker / instance counts change
- Balances shard-worker associations when shards added / removed
Amazon Kinesis Firehose

- Zero administration and seamless elasticity
- Direct-to-data store integration
- Continuous data transformations

Capture and submit streaming data
Firehose loads streaming data continuously into Amazon S3, Redshift and Elasticsearch
Analyze streaming data using your favorite BI tools
Easily capture and deliver data

- Write data to a Firehose delivery stream from a variety of sources
- Transform, encrypt, and/or compress data along the way
- Buffer and aggregate data by time and size before it is written to destination
- Elastically scales with no resource provisioning

AWS Platform SDKs Mobile SDKs Kinesis Agent AWS IoT

Amazon Kinesis Firehose

Amazon S3 Amazon Redshift Amazon Elasticsearch Service
Amazon Kinesis Analytics

- Apply SQL on streams
- Build real-time, stream processing applications
- Easy scalability

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
Use SQL to build real-time applications

Connect to streaming source

Easily write SQL code to process streaming data

Continuously deliver SQL results
AWS Lambda

- Function code triggered from newly arriving events
- Simple event-based processing of records
- Serverless processing with low administration

Social media stream is loaded into Kinesis in real-time
Lambda runs code that generates hashtag trend data and stores it in DynamoDB
Social media trend data immediately available for business users to query
Amazon Elastic Map Reduce (EMR)

- Ingest streaming data from many sources
- Easily configure clusters with latest versions of open source frameworks
- Less underlying performance management

*Ingest streaming data through Amazon Kinesis Streams*

*Your choice of stream data processing engine, Spark Streaming or Apache Flink*

*Send processed data to S3, HDFS, or a custom destination using an open source connector*
Streaming App
Patterns & Use Cases
## Stream Processing Use Cases

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Streaming</td>
<td>• Ingest and store raw data at high volume</td>
</tr>
<tr>
<td>Ingest-Transform-Load</td>
<td>• Atomic transformations</td>
</tr>
<tr>
<td></td>
<td>• Simple data enrichment</td>
</tr>
<tr>
<td>Continuous Metric</td>
<td>• Windowed analytics (count X over 5 minutes)</td>
</tr>
<tr>
<td>Generation</td>
<td>• Event correlation like sessionization</td>
</tr>
<tr>
<td></td>
<td>• Visualization</td>
</tr>
<tr>
<td>Actionable Insights</td>
<td>• Act on the data by triggering events or alerts</td>
</tr>
<tr>
<td></td>
<td>• Machine learning</td>
</tr>
<tr>
<td></td>
<td>• Real-time feedback loops</td>
</tr>
</tbody>
</table>
Streaming Ingest-Transform-Load

Use Cases
• Ingest raw log data and store in S3 for later analysis
• Ingest and clean clickstream data, persist to a data warehouse
• Capture and validate IoT Sensor, device telemetry data

Key Requirements
• Durably ingest and buffer large volumes of small events
• Perform simple transformations
• Persist and store data efficiently
Deep Dive: Analyzing VPC Flow Logs

• VPC Flow Logs enable you to capture information about the IP traffic going to and from network interfaces in your VPC
• Solution provides both cost-effective storage of logs as well as near real-time analysis
Best Practices for VPC Flow Log Ingestion

Buffering data
• Amazon Kinesis provides makes it easy to aggregate data in one or more steps
• Further buffering on producer is sometimes used for efficiency

Reduce latency
• VPC Flow Logs buffers data locally which adds latency
• Use Amazon Kinesis Streams or Firehose directly to buffer if you need to react closer to real-time
Best Practices for VPC Flow Log Ingestion

Storage optimization

- Data is delivered and stored in GZIP format in Amazon S3
- Athena is faster with Parquet or partitioning
- Perform secondary transform using:
  - Spark cluster on Amazon EMR to transform to Parquet
  - AWS Lambda function to apply different partitions as data arrives on S3
Continuous Metric Generation

Examples
• Operational metrics and dashboards (e.g. API error counts)
• Massively multiplayer online game (MMOG) live dashboard (e.g. top 10 players)
• Clickstream analytics like impressions and page views

Key Requirements
• Produce accurate results with late and out of order data
• Combine results with historical data
• Quickly provide data to tech and non-tech end users
Deep Dive: Time-Series Analytics

- Ingest IoT data via MQTT and forward to Amazon Kinesis Stream
- Compute important metrics like average temperature every 10 seconds with Amazon Kinesis Analytics
- Persist time series analytics to RDS MySQL database that are immediately ready to serve via queries and BI tools
Best Practices for Time-Series Analytics

Data arrives out of order data
• Data is arriving from sensors intermittently and often out of order
• Sort the data using:
  • ORDER BY clause in Kinesis Analytics
  • Pre-process data using AWS Lambda

Event time versus process time
• Real-time use cases – processing time is OK
• Data is stored for usage later – aggregate data using event time as a key
Responsive Analysis

Examples
• Recommendation engines for retail websites
• Device operational intelligence and alerts
• Detecting trends and anomalies on user behavior

Key Requirements
• Ability to notify users and machines with low latency
• Long running, stateful operations over streams
Deep Dive: Real-Time Anomaly Detection

- Ingest data from website through API Gateway and Amazon Kinesis Streams
- Use Amazon Kinesis Analytics to produce an anomaly score for each data point and identify trends in data
- Send users and machines notifications through Amazon SNS
Best Practices for Anomaly Detection

Scalable ingest and processing is important
- Real time apps become significant less valuable when you fall behind
- Architecture is highly scalable but must be closely monitored

Your data is different
- Different data in different domains produce different results in ML algorithms
- Algorithms are a tool for scientists, not a replacement
- Must understand your data, its patterns (logarithmic, circadian, etc.), and algorithmic results to make decisions
Stream Processing for Analytics
Deliveroo Data Engineering

Provide scalable hosting and data crunching services for analytics, data science, and product applications
Analytics at Deliveroo

• Just like any other business, we need to move data from the applications that produce it into systems that are optimised for analytics.

• Aggregation queries

• Joined datasets

• Lookups from external systems

• There’s significant work required to manipulate data that’s optimised for the application into data that’s useful for answering a wide range of questions
Analytics at Deliveroo

Our data volumes:

• 20TB in our data warehouse
• ~10 million event records per day
• Enough to warrant reasonably heavy distributed processing when we process this in batch mode
Batch Data Processing for Analytics

• Take a window of data (typically 1 whole day), and append this to your existing data in your data warehouse
Batch Data Processing for Analytics

- Take a window of data (typically 1 whole day), and append this to your existing data in your data warehouse
- Simple right?
Batch Data Processing for Analytics

- Take a window of data (typically 1 whole day), and append this to your existing data in your data warehouse
- Simple right?

... Batch 23  Batch 24  Batch 25
Batch Data Processing for Analytics

• Take a window of data (typically 1 whole day), and append this to your existing data in your data warehouse

• Simple right?

• But what if something is wrong with this latest batch?
Batch Data Processing for Analytics

• Take a window of data (typically 1 whole day), and append this to your existing data in your data warehouse

• Simple right? 4 hours to process

• But what if something is wrong with this latest batch?

• And this batch is not cheap to compute
Batch Data Processing for Analytics

- Take a window of data (typically 1 whole day), and append this to your existing data in your data warehouse
- Simple right?
- But what if something is wrong with this latest batch?
- And this batch is not cheap to compute
- Answer Reprocess the whole thing
Batch Data Processing for Analytics

- Another approach
- Smaller batches 10 minutes to process

Batch 23  Batch 24  Batch 25  Batch 26  Batch 27  Batch 28
Batch Data Processing for Analytics

- Another approach
- Smaller batches
- And now we discover a problem from an old batch

10 minutes to process

Batch 23  Batch 24  Batch 25  Batch 26  Batch 27  **Batch 28**  Batch 28  Batch 29
Batch Data Processing for Analytics

- Another approach
- Smaller batches
- And now we discover a problem from an old batch

10 minutes to process

- Now we need to roll back our batches and re-process them.
Another approach

Smaller batches

And now we discover a problem from an old batch

Now we need to roll back our batches and re-process them.

What happens if we apply the same batches again? This brings up the topic of data pipeline idempotency.
Idempotency with Batched Datasets

• With bounded datasets pipeline logic for loading data into the warehouse needs to have full knowledge of what’s in the destination tables already

• Logic needs to be built in to avoid duplication of data, and to ensure the effect of applying the batch more than once is not detrimental for data integrity
Other Batch Processing Challenges

- The bigger the processing window, the slower the data is to process.
- Compute costs can be significantly higher.
- It’s an all or nothing approach (within the context of the batch you’re interested in).
- Late arriving data may necessitate re-processing and subsequent expensive updates to already processed tables.
Stream Data Processing
Stream Data Processing

Up to 7 Days History

...
Stream Data Processing

Up to 7 Days History

LATEST
Stream Data Processing

Up to 7 Days History

Start at timestamp 1498501427293

Process all records up to latest

LATEST
Stream Data Processing

Up to 7 Days History

Start at timestamp 1498501427293

Process all records up to latest

Document Store

LATEST
Stream Data Processing

If the destination of the data is a document store, you can get a lot of data transformation conveniences with less effort

- De-duplication
- Denormalisation (through partial updates)
- Idempotency
- Other effects that can simulate MapReduce over batches
Stream Data Processing

• Stream data is the first class citizen

• At the same time (from the same stream) persist the raw event data (can use Kinesis Firehose for this). This persisted data could form part of a data lake.

• Makes use of cost effective storage on AWS (S3)

• If something should change about your understanding of already processed data, the data lake allows you to re-process this

• Makes use of cost effective compute resources available on AWS (EC2/EMR)

• The challenge is in aligning the processing logic between stream and batch
Stream Data Processing

Source Systems → Kinesis (7 days history) → EC2 → Kinesis Firehose → S3 (Raw Data (Full history)) → ElasticSearch (1 month history) → Visualisation Tools
An Example at Deliveroo

Application Events

- Event data is captured from our consumer facing web and mobile apps
- This is useful for:
  - calculating conversion rates (ratio of people who placed an order to those that had some interaction with our site/app)
  - AB testing
  - anomaly/error detection in the app itself

There’s a number of reasons why this event data is not ideal for analysis in its raw form

- Users interact with our site with anonymous ids before they sign in
- Bot traffic skews our understanding of what real users are doing
## Application Events - Getting an accurate visitor count

<table>
<thead>
<tr>
<th>messageId</th>
<th>timestamp</th>
<th>anonymousId</th>
<th>userId</th>
<th>eventType</th>
</tr>
</thead>
<tbody>
<tr>
<td>d4hgfi...334gf</td>
<td>2017-06-23T23:11:17.345Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Homepage</td>
</tr>
<tr>
<td>j349...fh48kdh</td>
<td>2017-06-23T23:11:18.832Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Restaurant List</td>
</tr>
<tr>
<td>f84h...fsl3kfne</td>
<td>2017-06-23T23:11:24.912Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Added Item to Basket</td>
</tr>
<tr>
<td>fkz3...2jf94kd</td>
<td>2017-06-23T23:11:28.143Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Homepage</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kn4d...38fnxk</td>
<td>2017-06-23T23:11:34.483Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Checkout</td>
</tr>
<tr>
<td>ojg4n...3jf9sk</td>
<td>2017-06-23T23:11:35.374Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>23597348</td>
<td>Signed-in</td>
</tr>
</tbody>
</table>
### Application Events - Getting an accurate visitor count

<table>
<thead>
<tr>
<th>messageId</th>
<th>timestamp</th>
<th>anonymousId</th>
<th>userId</th>
<th>eventType</th>
</tr>
</thead>
<tbody>
<tr>
<td>d4hgfi...334gf</td>
<td>2017-06-23T23:11:17.345Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Homepage</td>
</tr>
<tr>
<td>j349...fh48kdh</td>
<td>2017-06-23T23:11:18.832Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Restaurant List</td>
</tr>
<tr>
<td>f84h...fsl3kfne</td>
<td>2017-06-23T23:11:24.912Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Added Item to Basket</td>
</tr>
<tr>
<td>f4z3...2jf94kd</td>
<td>2017-06-23T23:11:28.143Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Homepage</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>kn4d...38fnxk</td>
<td>2017-06-23T23:11:34.483Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Checkout</td>
</tr>
<tr>
<td>ojg4n...3jf9sk</td>
<td>2017-06-23T23:11:35.374Z ...</td>
<td>ag3og9h...3adsw3o</td>
<td>23597348</td>
<td>Signed-in</td>
</tr>
</tbody>
</table>
### An Example at Deliveroo

**Application Events - Getting an accurate visitor count**

<table>
<thead>
<tr>
<th>messageId</th>
<th>timestamp</th>
<th>anonymousId</th>
<th>userId</th>
<th>eventType</th>
<th>combineid</th>
</tr>
</thead>
<tbody>
<tr>
<td>d4hgfi...334gf</td>
<td>2017-06-23T23:11:17.345Z</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Homepage</td>
<td>23597348</td>
</tr>
<tr>
<td>j349...fh48kdh</td>
<td>2017-06-23T23:11:18.832Z</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Restaurant List</td>
<td>23597348</td>
</tr>
<tr>
<td>f84h...fsl3kfne</td>
<td>2017-06-23T23:11:24.912Z</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Added Item to Basket</td>
<td>23597348</td>
</tr>
<tr>
<td>fkz3...2jf94kd</td>
<td>2017-06-23T23:11:28.143Z</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Homepage</td>
<td>23597348</td>
</tr>
<tr>
<td>kn4d...38fnxk</td>
<td>2017-06-23T23:11:34.483Z</td>
<td>ag3og9h...3adsw3o</td>
<td>null</td>
<td>Viewed Checkout</td>
<td>23597348</td>
</tr>
<tr>
<td>ojg4n...3jf9sk</td>
<td>2017-06-23T23:11:35.374Z</td>
<td>ag3og9h...3adsw3o</td>
<td>23597348</td>
<td>Signed-in</td>
<td>23597348</td>
</tr>
</tbody>
</table>
An Example at Deliveroo

Application Events - How we solve this with batch processing

Using Spark on EMR

• Capture all sign-in events (mapping of anonymousIds to userlIds) and distribute a copy of this dataset to every node on the cluster (using a broadcast variable)

• Map over the dataset to lookup the values into the combinedId field

• To get an accurate picture of back propagated activity, this needs to be quite a big batch (more than one day, more like 30 days)

• Idempotency is achieved through deleting the same dates from the existing dataset as those contained in the incoming batch
An Example at Deliveroo

Application Events - How we solve this with unbounded processing

Using Kinesis into Elasticsearch

• Stream data directly into Elasticsearch

• On detection of a signed-in record, apply an indexed update to the already persisted events that match on the anonymousId to populate the combinedId field

• Idempotency provided by the fact that the events are keyed on messageId, so replaying the stream just overwrites the data that was there originally, with updates applied in the same order
Summary

Stream processing gives us:

• Lighter weight data transformations

• An elegant way to re-process data, through the use of replay from a point in time

• If there’s a use case, we also have the opportunity to consume (analyse) the data in real-time

However, we would not want to ignore batch processing entirely. Rather treat it as a subset of our streaming use case.

Save the heavier distributed processing for when we want to do a large scale replay of data.
Where to go next?
Try these use cases yourself

Many variations of these use cases have sample code on the AWS Big Data Blog. Follow the blog! https://aws.amazon.com/blogs/big-data/

Some good examples:

- Analyzing VPC Flow Logs with Amazon Kinesis Firehose, Amazon Athena, and Amazon QuickSight
- Real-time Clickstream Anomaly Detection with Amazon Kinesis Analytics
- Writing SQL on Streaming Data with Amazon Kinesis Analytics | Part 1, Part 2
Lots of customer examples

Amazon Kinesis as Databus - Migrate from Kafka to Kinesis | Enterprise

1 billion events/wk from connected devices | IoT

17 PB of game data per season | Entertainment

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

100 GB/day click streams from 250+ sites | Enterprise

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

10 million events/day | Retail

Funnel all production events through Amazon Kinesis
Integrate with your current solutions
Get help from partner systems integrators
London Amazon Redshift

Wednesday, July 5, 2017 - 6:00 PM to 8:00 PM
60 Holborn Viaduct, London
http://goo.gl/maps/yMZPT

{1:“Redshift Deep Dive and new features since last Meetup” | 2: “OLX presenting Advanced Analytics and Machine Learning with Redshift” | 3:“Other customer/partner case studies” | 4:“Next steps for the community”}
Thank you

armpaul@amazon.co.uk