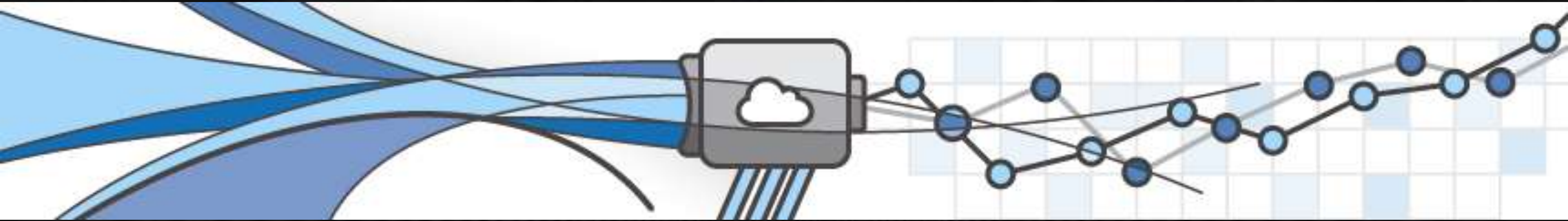
Real-Time Event Processing

Ben Snively

Solutions Architect



Herndon, VA

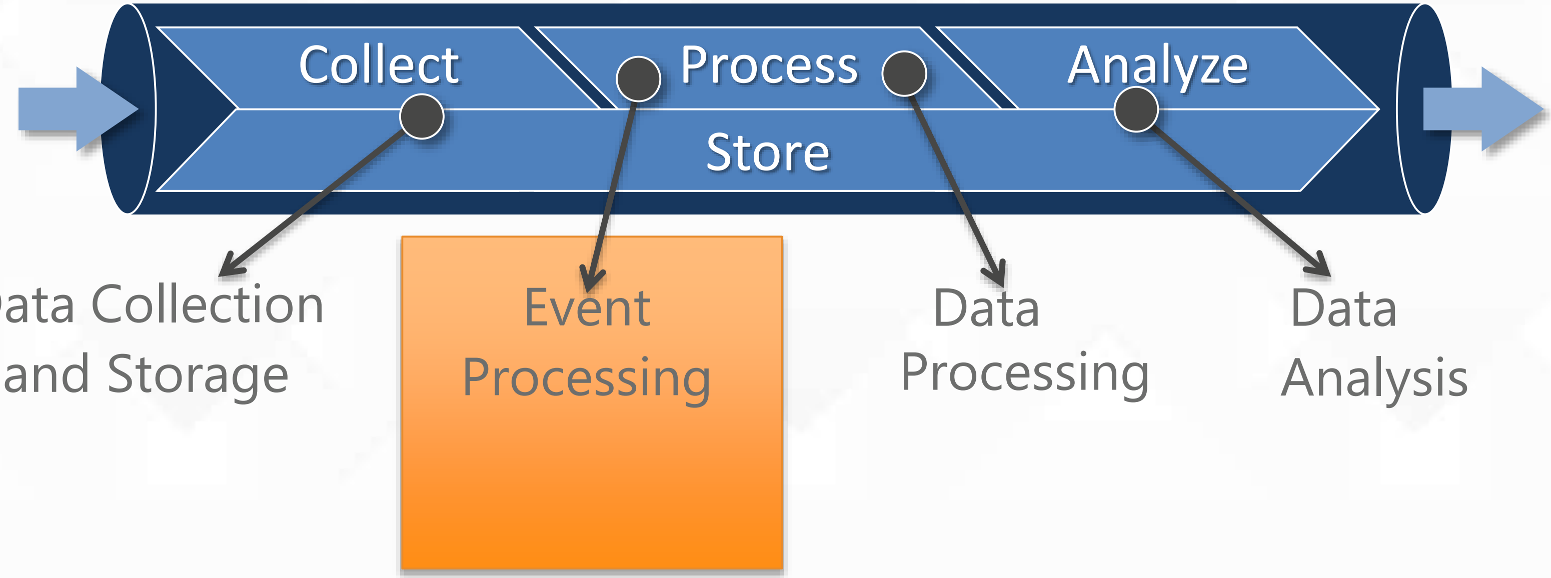
November 10th, 2015

Agenda Overview

8:30 AM	Welcome
8:45 AM	**Big Data in the Cloud**
10:00 AM	Break
10:15 AM	**Data Collection and Storage**
11:30 AM	Break
11:45 AM	**Real-time Event Processing**
1:00 PM	Lunch
1:30 PM	**HPC in the Cloud**
2:45 PM	Break
3:00 PM	**Processing and Analytics**
4:15 PM	Close

amazon web services

Primitive Patterns



Real-Time Event Processing

Reminder

- Event-driven programming
- Trigger action based on real-time input

Examples:

- ❑ Proactively detect errors in logs and devices
- ❑ Identify abnormal activity
- ❑ Monitor performance SLAs
- ❑ Notify when SLAs/performance drops below a threshold

Two main processing patterns

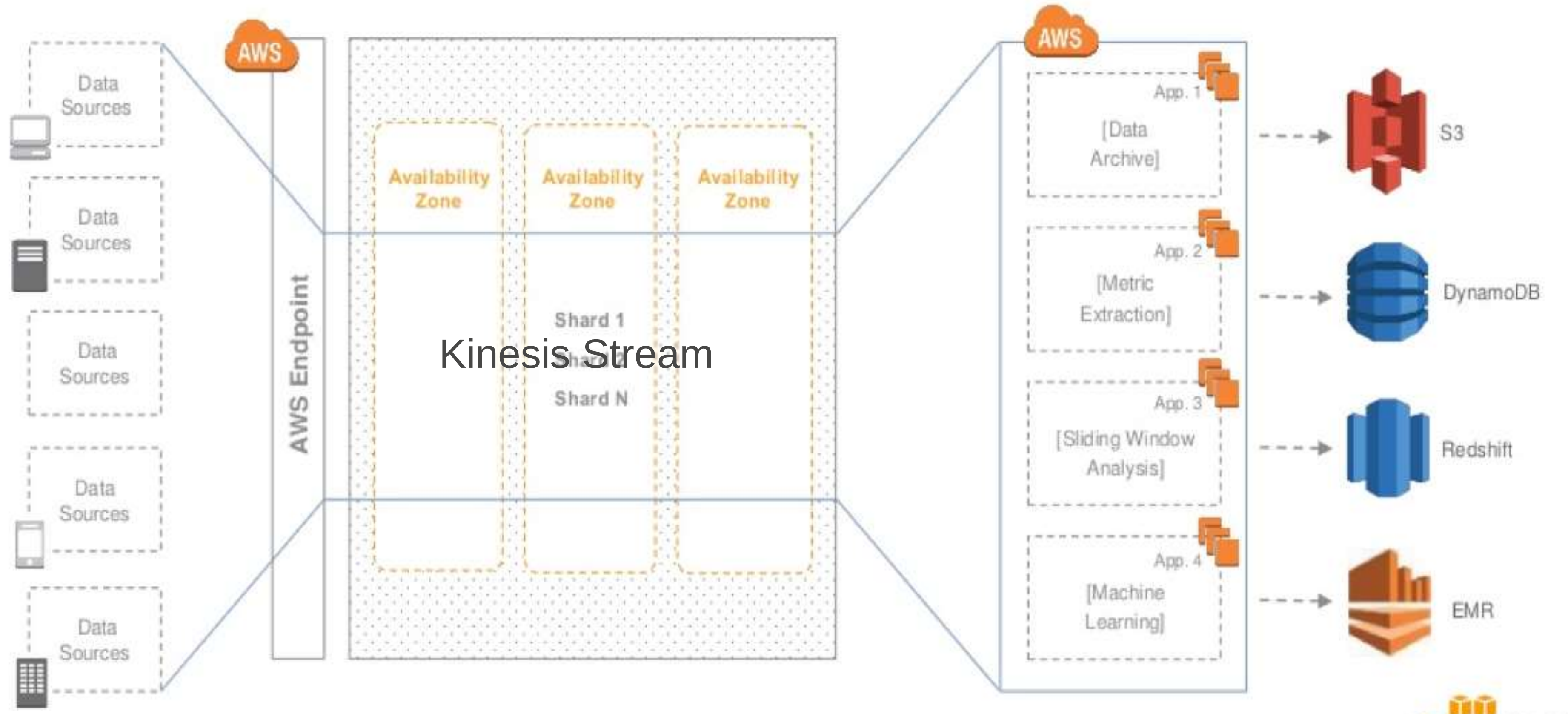
- Stream processing (real time)
  - ❑ Real-time response to events in data streams
  - ❑ Relatively simple data computations (aggregates, filters, sliding window)

- Micro-batching (near real time)
  - ❑ Near real-time operations on small batches of events in data streams
  - ❑ Standard processing and query engine for analysis

You’re likely already “streaming”

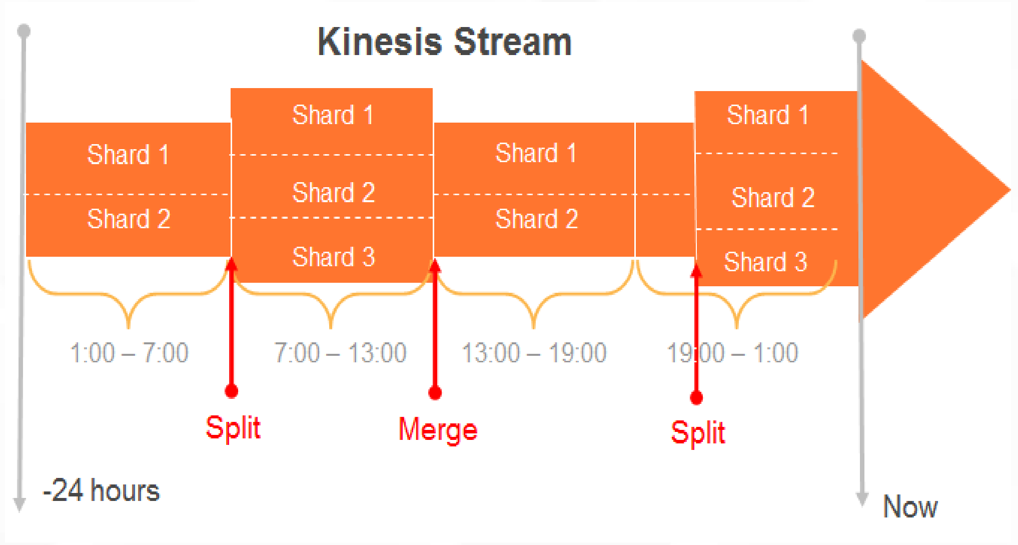
- Sensor networks analytics
- Network analytics
- Log shipping and centralization
- Click stream analysis
- Gaming status
- Hardware and software appliance metrics
- …more…
- Any proxy layer B organizing and passing data from A to C
  - ❑ A to B to C

Amazon Kinesis



amazon web services

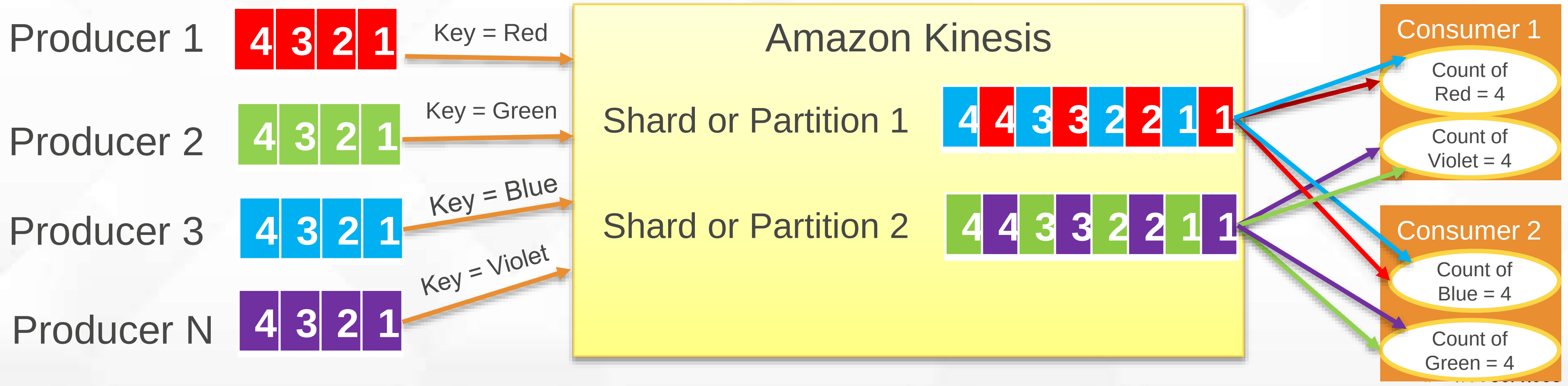
Kinesis Stream & Shards



- Streams are made of **Shards**
- Each Shard ingests data up to 1MB/sec, and up to 1000 TPS
- Each Shard emits up to 2 MB/sec
- All data is stored for **24 hours**
- **Scale** Kinesis streams by splitting or merging Shards
- **Replay** data inside of 24Hr. Window
  - Extensible to up to 7 days

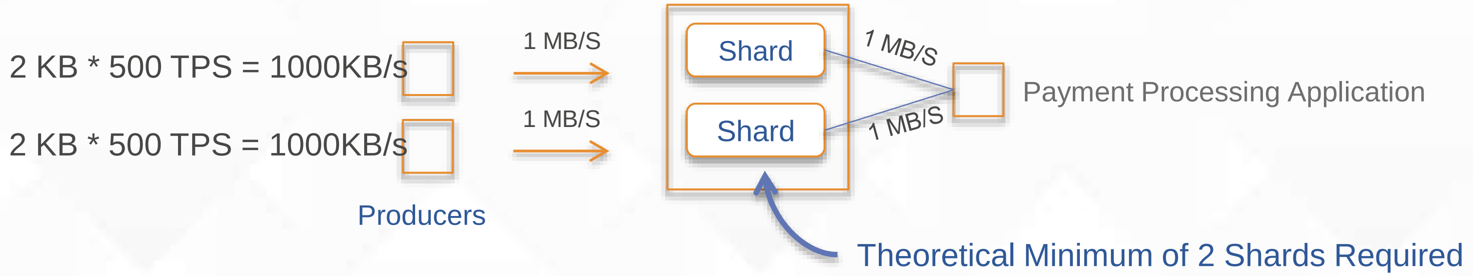
Why Stream Storage?

- Decouple producers & consumers
- Temporary buffer
- Preserve client ordering
- Streaming MapReduce



How to Size your Kinesis Stream - Ingress

Suppose 2 Producers, each producing 2KB records at 500 KB/s:



2 KB * 500 TPS = 1000KB/s
2 KB * 500 TPS = 1000KB/s
Producers
1 MB/S
1 MB/S
Shard
Shard
1 MB/S
1 MB/S
Payment Processing Application
Theoretical Minimum of 2 Shards Required

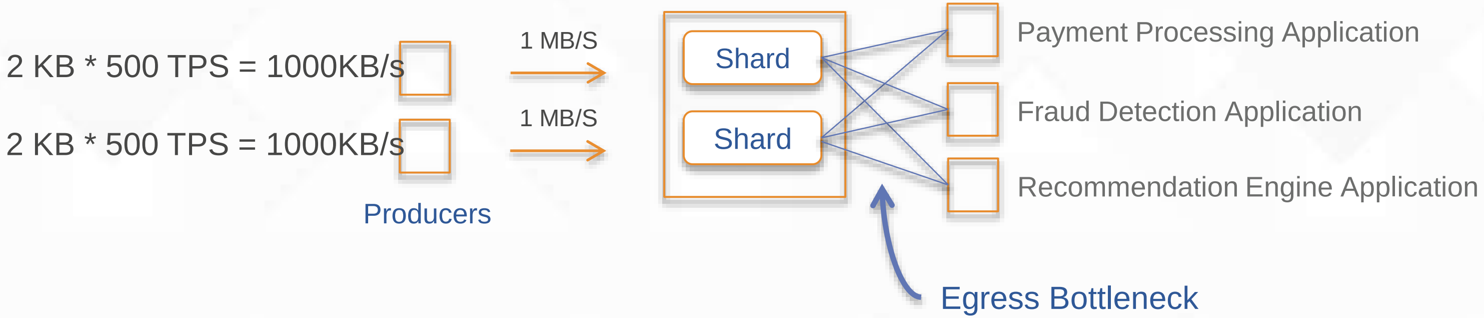
Minimum Requirement: Ingress Capacity of 2 MB/s, Egress Capacity of 2MB/s

A theoretical minimum of **2 shards** is required which will provide an ingress capacity of 2MB/s, and egress capacity 4 MB/s

How to Size your Kinesis Stream - Egress

Records are durably stored in Kinesis for 24 hours, allowing for multiple consuming applications to process the data

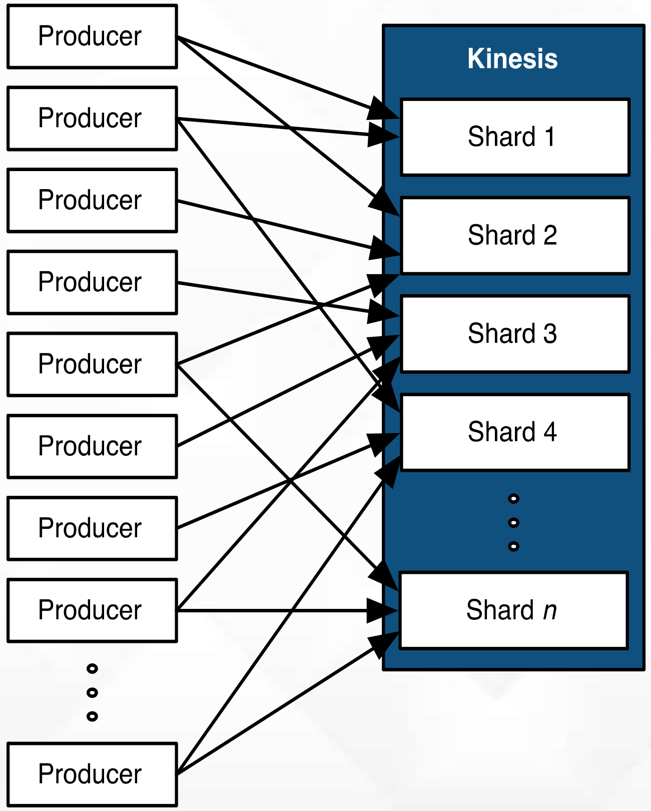
Let's extend the same example to have 3 consuming applications:


2 KB * 500 TPS = 1000KB/s
2 KB * 500 TPS = 1000KB/s
1 MB/S
1 MB/S
Producers
Shard
Shard
Payment Processing Application
Fraud Detection Application
Recommendation Engine Application
Egress Bottleneck

If all applications are reading at the ingress rate of 1MB/s per shard, an aggregate read capacity **of 6 MB/s** is required, exceeding the shard's egress limit of 4MB/s

Solution: Simple! ***Add*** another shard to the stream to spread the load

Putting Data into Kinesis
Simple Put interface to store data in Kinesis



- Producers use **`PutRecord`** `or` **`PutRecords`** call to store data in a Stream.
- **`PutRecord`** `{Data, StreamName, PartitionKey}`
- A **Partition Key** is supplied by producer and used to distribute the PUTs across Shards
- Kinesis **MD5 hashes** supplied partition key over the hash key range of a Shard
- A unique **Sequence #** is returned to the Producer upon a successful call

Real-time event processing frameworks

Kinesis Client Library

AWS Lambda

Spark Streaming

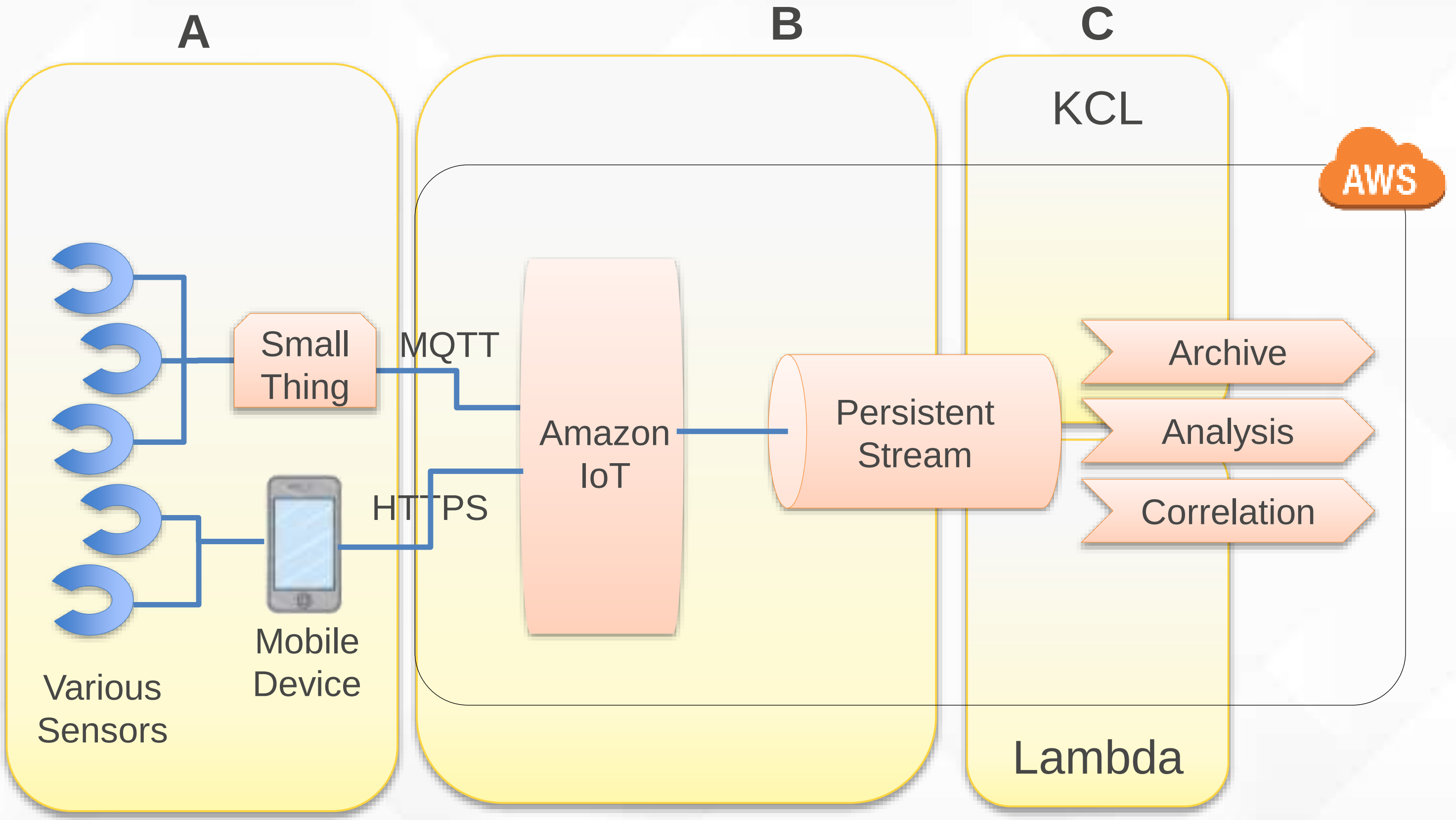
STORM

amazon web services

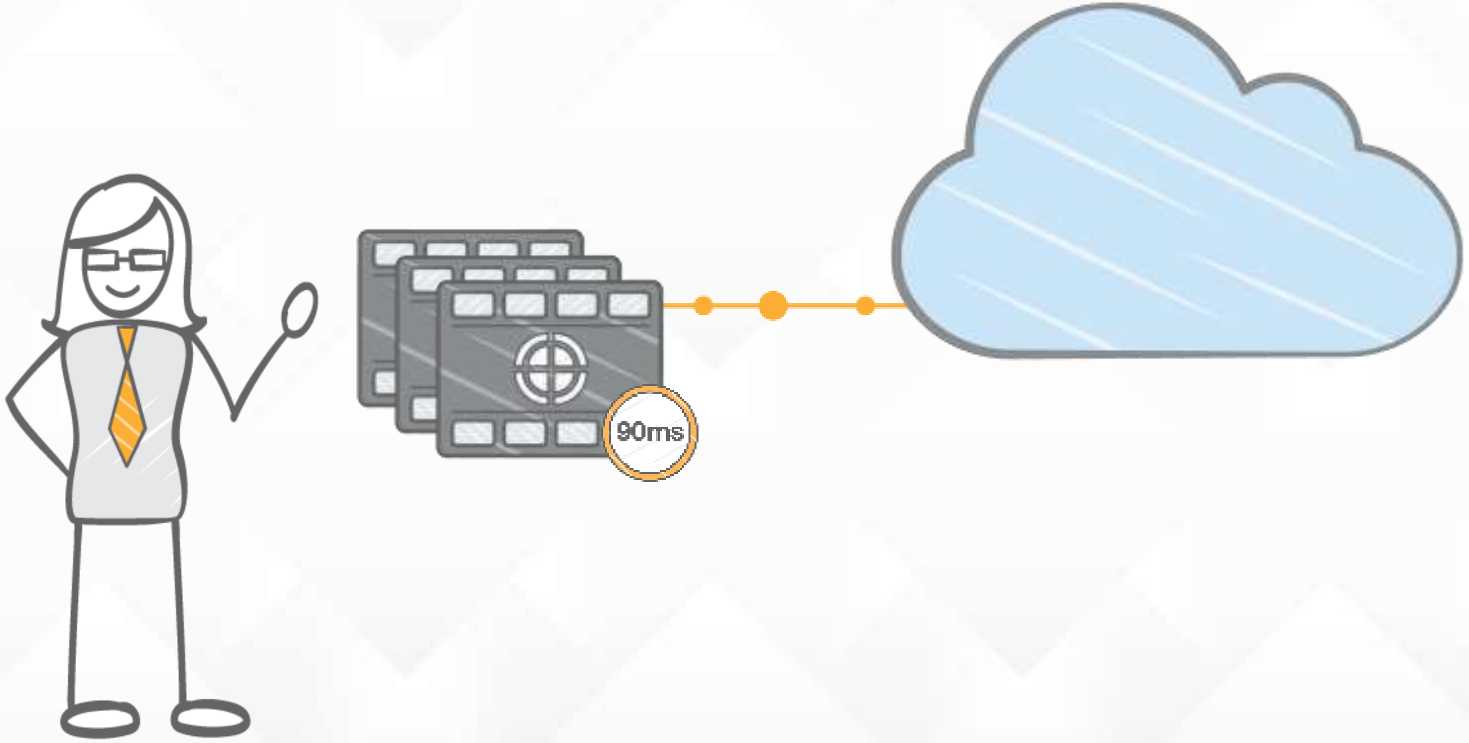
Use Case: IoT Sensors

Remotely determine what a device senses.

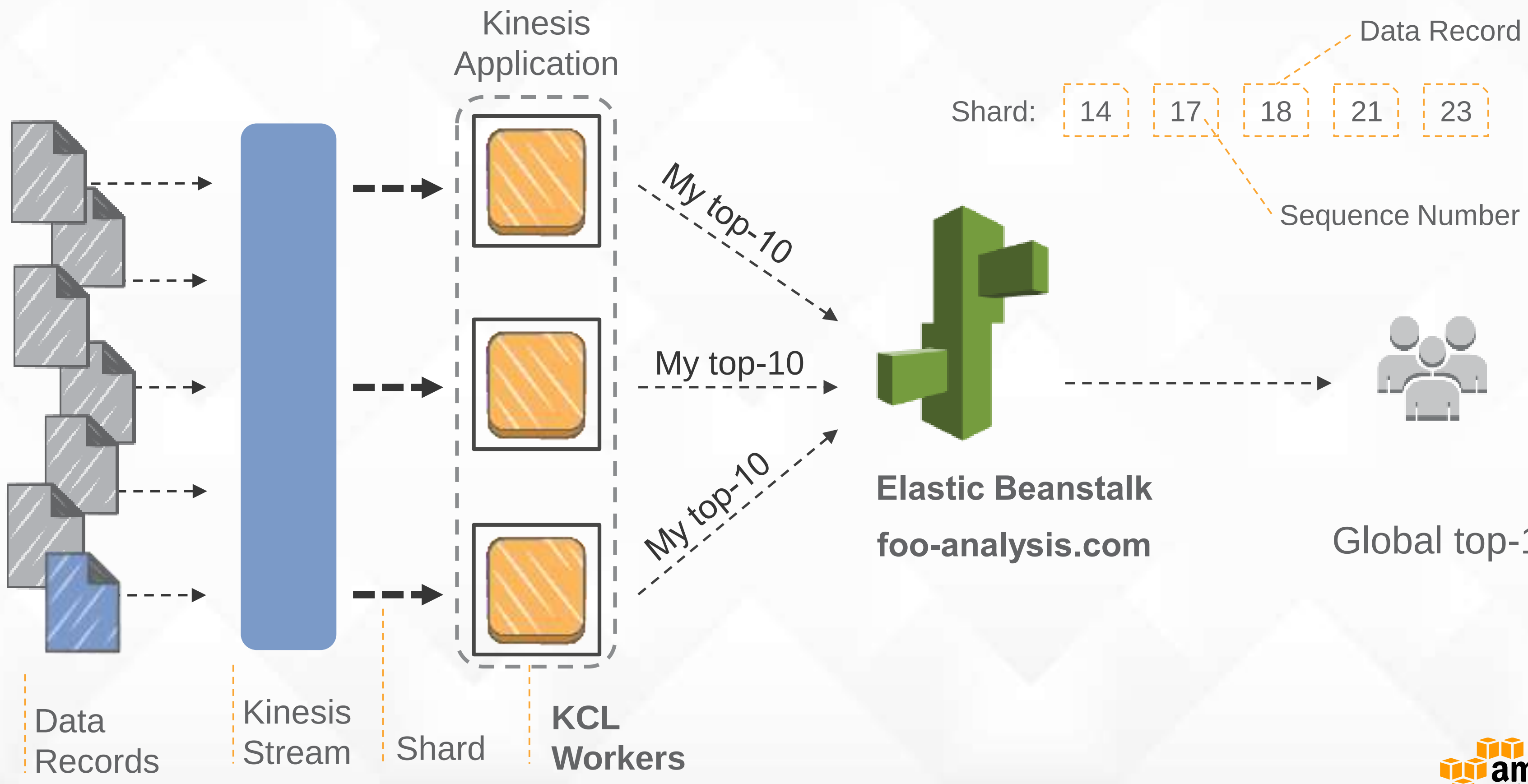
IoT Sensors - Trickles become a Stream



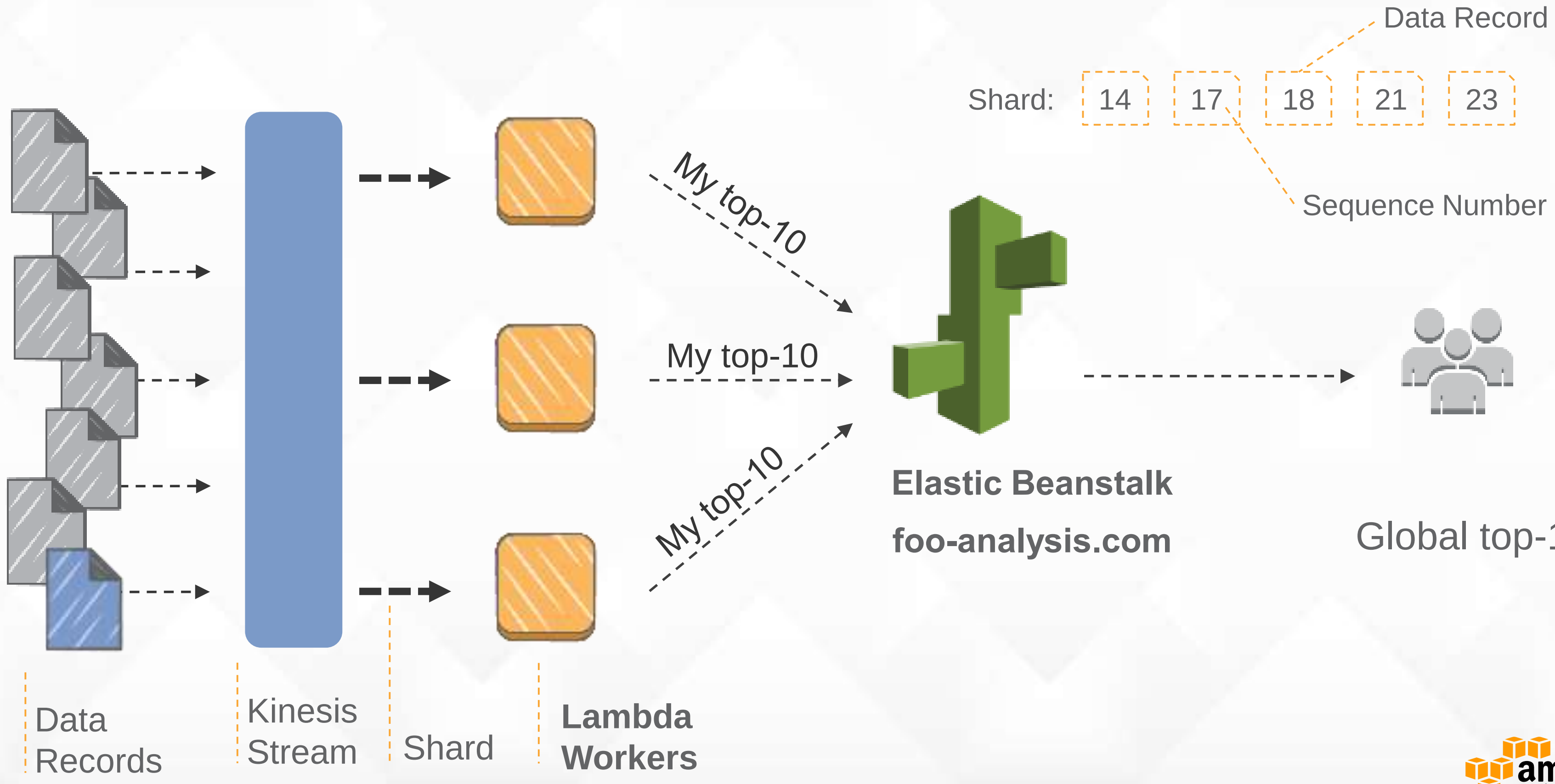
Use Case: Trending Top Activity



Ad Network Logging Top-10 Detail - KCL



Ad Network Logging Top-10 Detail - Lambda

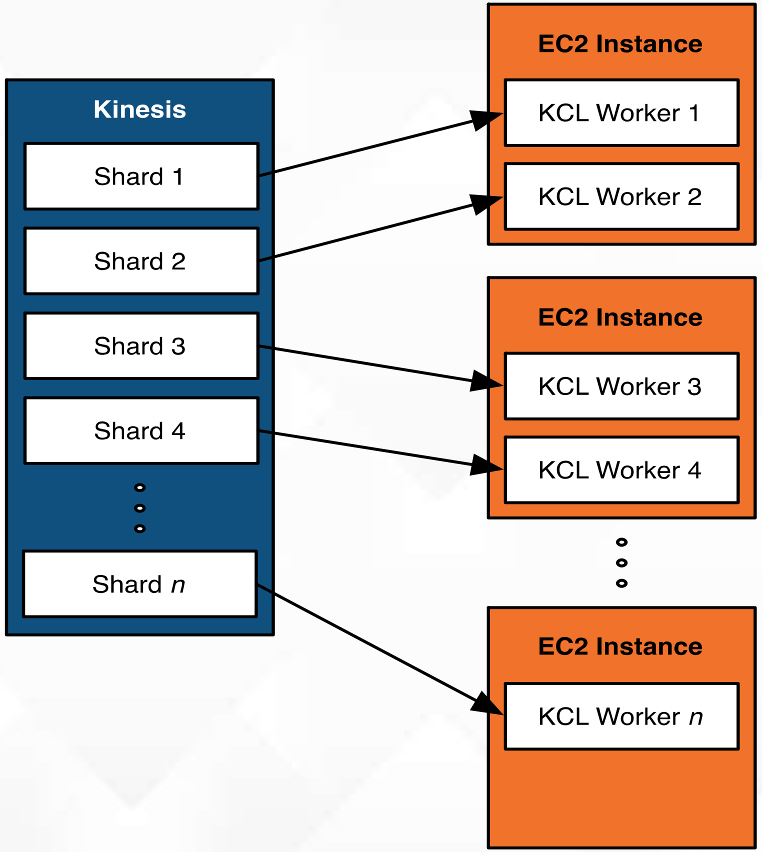


Amazon KCL

Kinesis Client Library

Kinesis Client Library (KCL)

- Distributed to handle multiple shards
- Fault tolerant
- Elastically adjust to shard count
- Helps with distributed processing
- Develop in Java, Python, Ruby, Node.js, .NET



KCL Design Components

- Worker:- Processing unit that maps to each application instance
- Record processor:- The processing unit that processes data from a shard of a Kinesis stream
- Check-pointer: Keeps track of the records that have already been processed in a given shard
- KCL restarts the processing of the shard at the last known processed record if a worker fails

KCL restarts the processing of the shard at the last known processed record if a worker fails

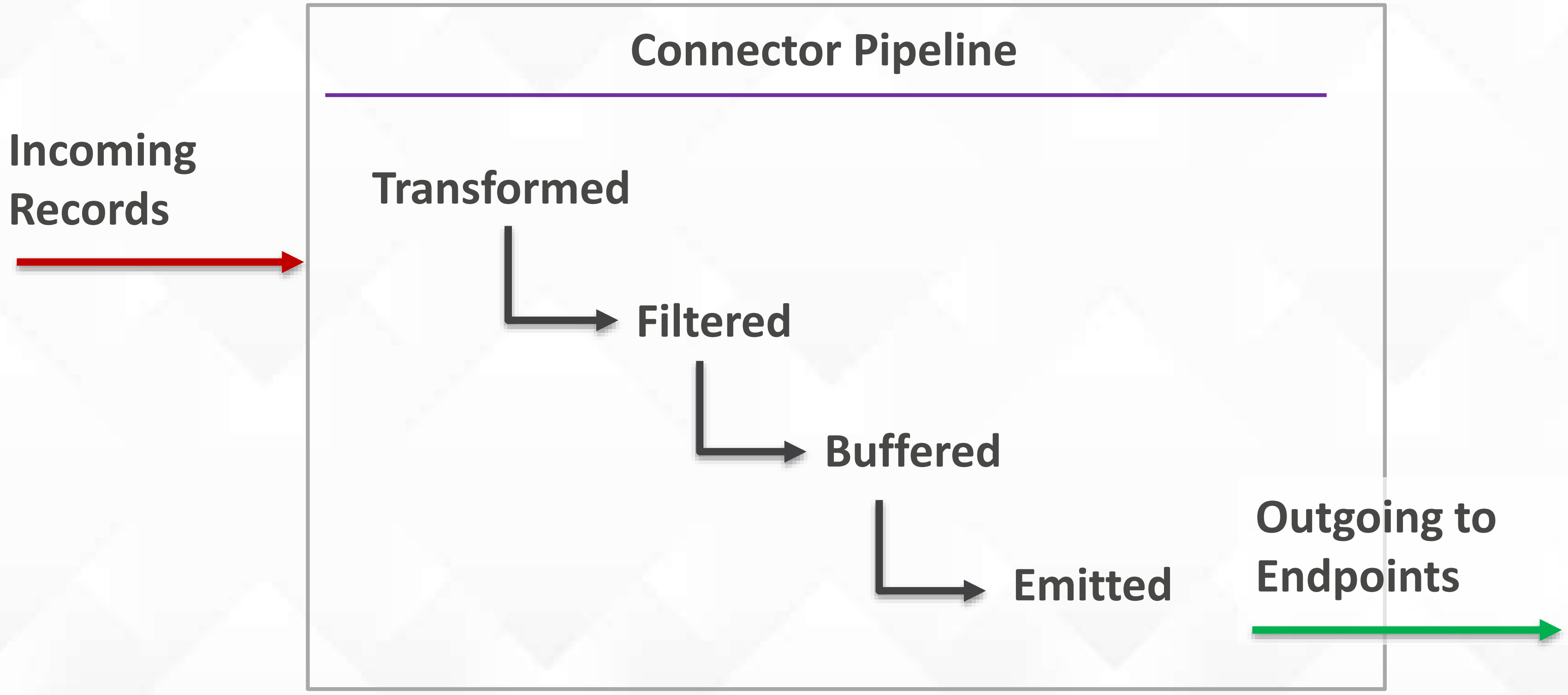
Processing with Kinesis Client Library

- Connects to the stream and enumerates the shards
- Instantiates a record processor for every shard managed
- Checkpoints processed records in Amazon DynamoDB
- Balances shard-worker associations when the worker instance count changes
- Balances shard-worker associations when shards are split or merged

Best practices for KCL applications

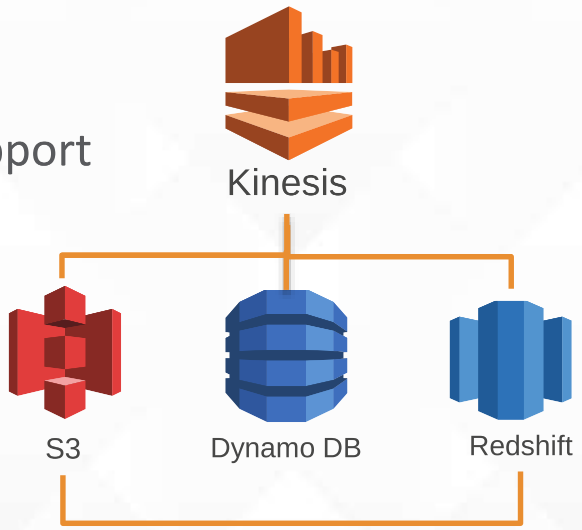
- Leverage EC2 Auto Scaling Groups for your KCL Application
- Move Data from an Amazon Kinesis stream to S3 for long-term persistence
  - ❑ Use either Firehose or Build an “archiver” consumer application
- Leverage durable storage like DynamoDB or S3 for processed data prior to check-pointing
- Duplicates: Ensure the authoritative data repository is resilient to duplicates
- Idempotent processing: Build a deterministic/repeatable system that can achieve idempotence processing through check-pointing

Amazon Kinesis connector application



Amazon Kinesis Connector

- Amazon S3
  - ❑ Batch Write Files for Archive into S3
  - ❑ Sequence Based File Naming
- Amazon Redshift
  - ❑ Micro-batching load to Redshift with manifest support
  - ❑ User Defined message transformers
- Amazon DynamoDB
  - ❑ BatchPut append to table
  - ❑ User defined message transformers
- Elasticsearch
  - ❑ Put to Elasticsearch cluster
  - ❑ User defined message transforms



AWS Lambda

Event-Driven Compute in the Cloud

- *Lambda functions*: Stateless, request-driven code execution
  - ❑ Triggered by events in other services:
    - PUT to an Amazon S3 bucket
    - Write to an Amazon DynamoDB table
    - Record in an Amazon Kinesis stream
  - ❑ Makes it easy to…
    - Transform data as it reaches the cloud
    - Perform data-driven auditing, analysis, and notification
    - Kick off workflows

No Infrastructure to Manage

- Focus on business logic, not infrastructure
- Upload your code; AWS Lambda handles:
  - ❑ Capacity
  - ❑ Scaling
  - ❑ Deployment
  - ❑ Monitoring
  - ❑ Logging
  - ❑ Web service front end
  - ❑ Security patching

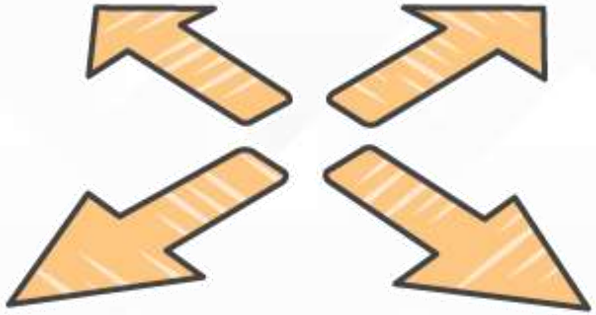
Automatic Scaling

- Lambda scales to match the event rate
- Don’t worry about over or under provisioning
- Pay only for what you use
- New app or successful app, Lambda matches your scale

Bring your own code

- Create threads and processes, run batch scripts or other executables, and read/write files in /tmp

- Include any library with your Lambda function code, even native libraries.



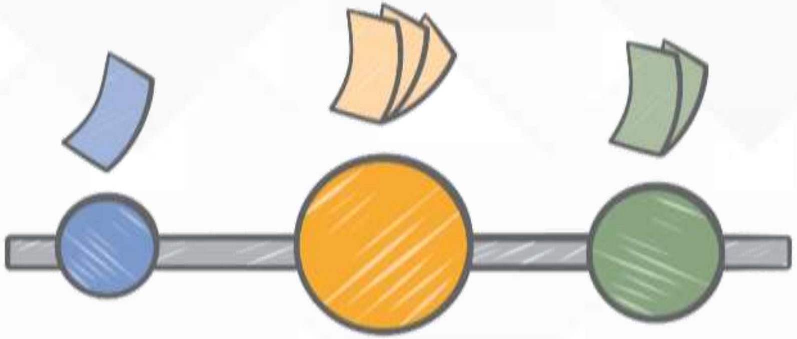
Fine-grained pricing

- Buy compute time in 100ms increments
- Low request charge
- No hourly, daily, or monthly minimums
- No per-device fees

Free Tier

1M requests and 400,000 GB-s of compute

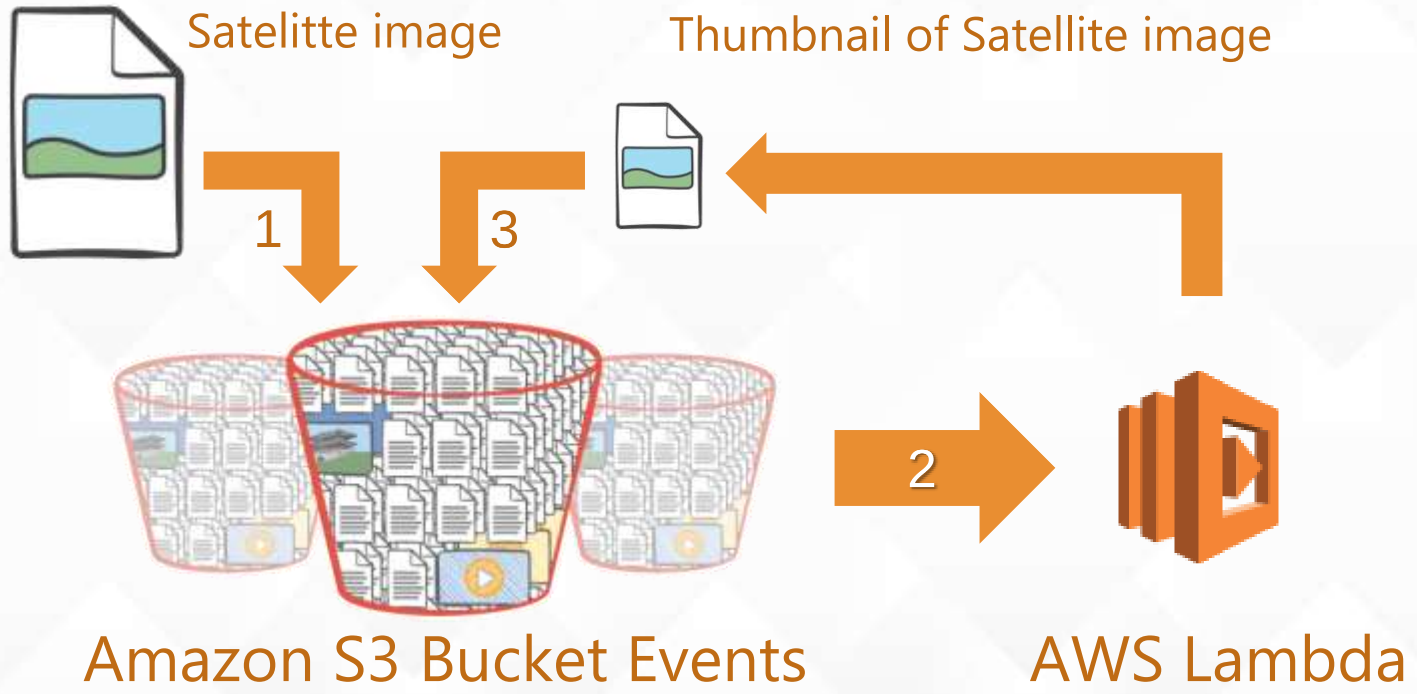
Every month, every customer



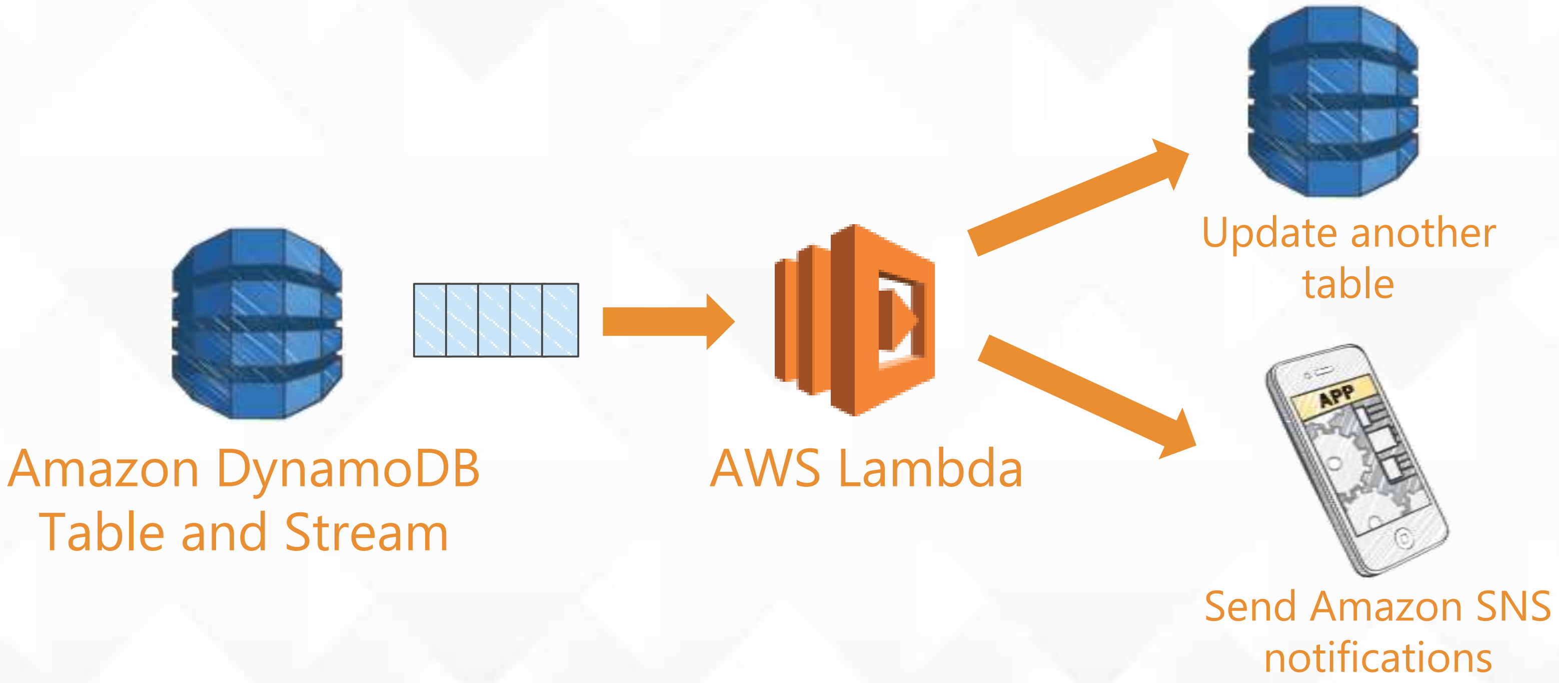
Never pay for idle

amazon web services

Data Triggers: Amazon S3



Data Triggers: Amazon DynamoDB



Calling Lambda Functions

- Call from mobile or web apps
  - Wait for a response or send an event and continue
  - AWS SDK, AWS Mobile SDK, REST API, CLI
- Send events from Amazon S3 or SNS:
  - One event per Lambda invocation, 3 attempts
- Process DynamoDB changes or Amazon Kinesis records as events:
  - Ordered model with multiple records per event
  - Unlimited retries (until data expires)

Writing Lambda Functions

- The Basics
  - ❑ Stock Node.js, Java, Python
  - ❑ AWS SDK comes built in and ready to use
  - ❑ Lambda handles inbound traffic
- Stateless
  - ❑ Use S3, DynamoDB, or other Internet storage for persistent data
  - ❑ Don’t expect affinity to the infrastructure (you can’t “log in to the box”)
- Familiar
  - ❑ Use processes, threads, /tmp, sockets, …
  - ❑ Bring your own libraries, even native ones

How can you use these features?

"I want to send customized messages to different users"

SNS + Lambda

"I want to send an offer when a user runs out of lives in my game"

Amazon Cognito + Lambda + SNS

"I want to transform the records in a click stream or an IoT data stream"

Amazon Kinesis + Lambda

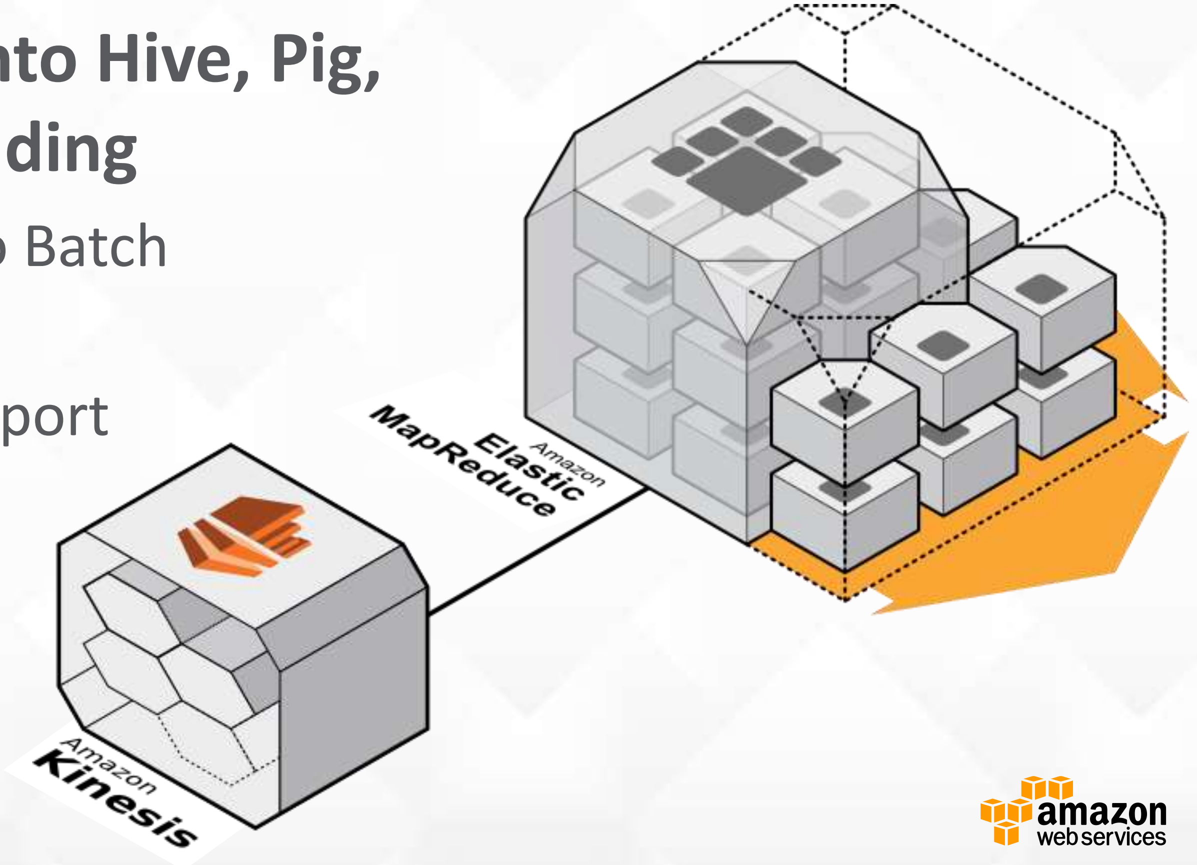
Stream Processing

Apache Spark
Apache Storm
Amazon EMR

Amazon EMR integration

Read Data Directly into Hive, Pig, Streaming and Cascading

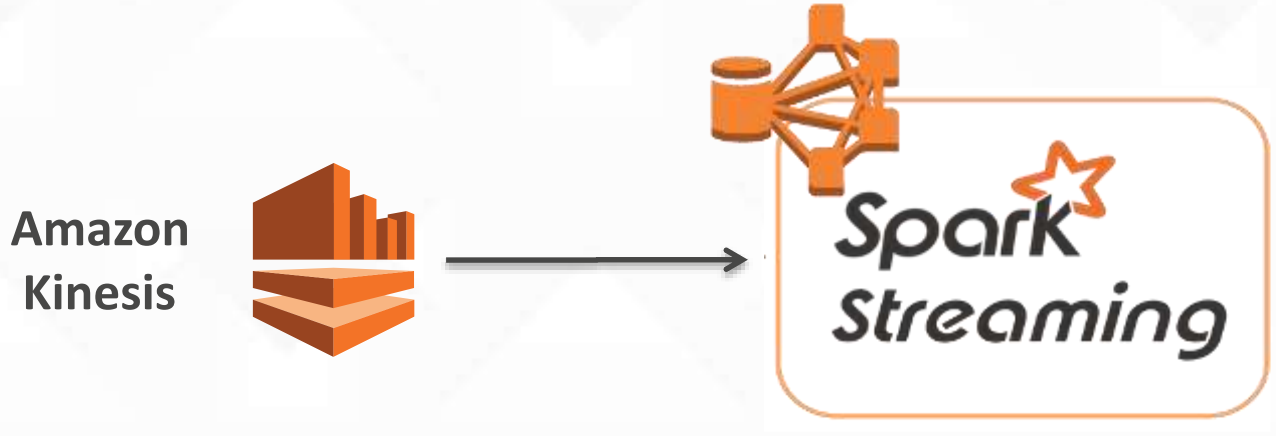
- Real time sources into Batch Oriented Systems
- Multi-Application Support and Check-pointing



Amazon EMR integration: Hive

```
CREATE TABLE call_data_records (
  start_time bigint,
  end_time bigint,
  phone_number STRING,
  carrier STRING,
  recorded_duration bigint,
  calculated_duration bigint,
  lat double,
  long double
)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY ","
STORED BY
'com.amazon.emr.kinesis.hive.KinesisStorageHandler'
TBLPROPERTIES("kinesis.stream.name"=”MyTestStream");
```

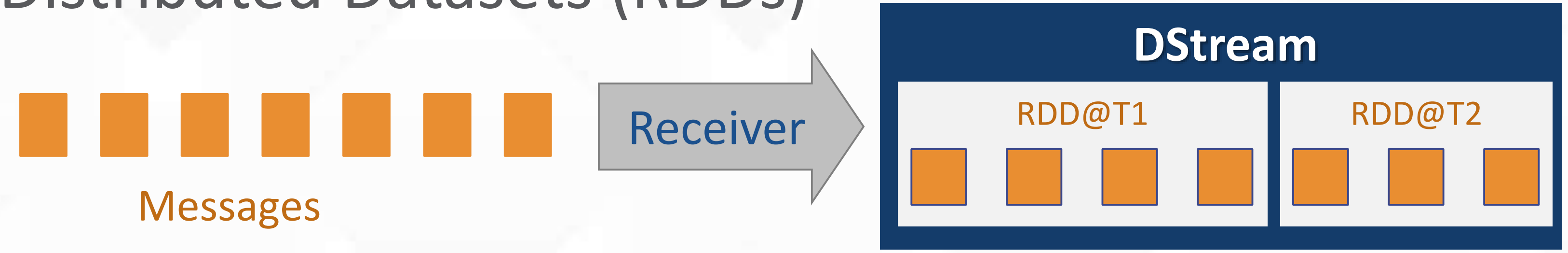
Processing Amazon Kinesis streams



EMR with
Spark Streaming

Spark Streaming – Basic concepts

- Higher level abstraction called Discretized Streams of DStreams
- Represented as a sequence of Resilient Distributed Datasets (RDDs)



http://spark.apache.org/docs/latest/streaming-kinesis-integration.html

Apache Spark Streaming

- Window based transformations
  - ❑ countByWindow, countByValueAndWindow etc.
- Scalability
  - ❑ Partition input stream
  - ❑ Each receiver can be run on separate worker
- Fault tolerance
  - ❑ Write Ahead Log (WAL) support for Streaming
  - ❑ Stateful exactly-once semantics

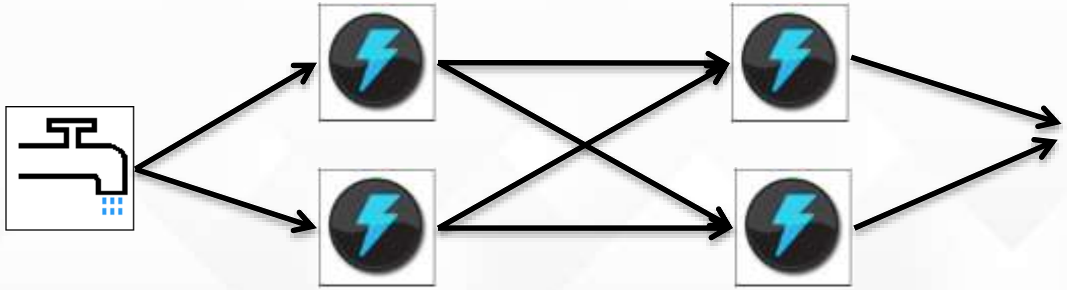
- Flexibility of running what you want
- EC2/etc – wordsmith here

Apache Storm

- Guaranteed data processing
- Horizontal scalability
- Fault-tolerance
- Integration with queuing system
- Higher level abstractions

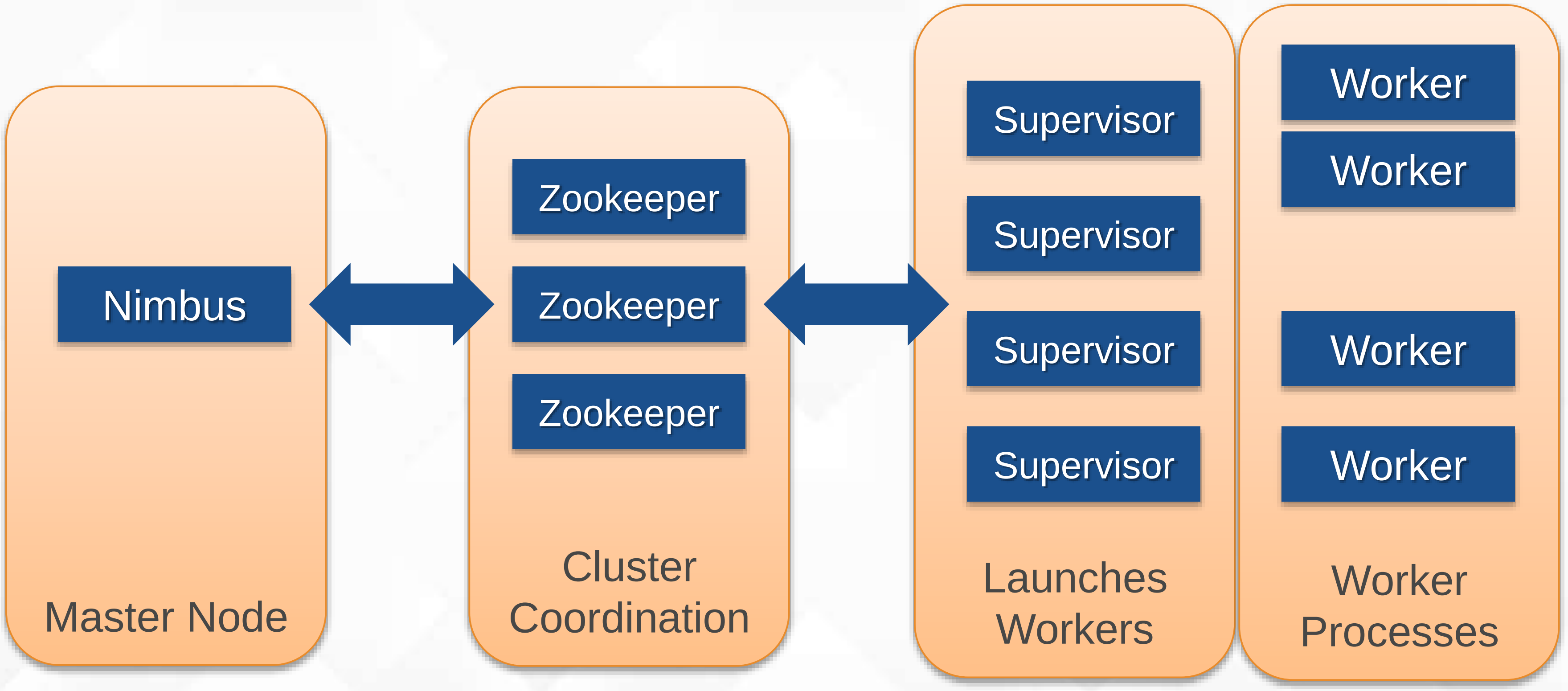
Apache Storm: Basic Concepts

- *Streams*: Unbounded sequence of tuples
- *Spout*: Source of Stream
- *Bolts* :Processes input streams and output new streams
- *Topologies* :Network of spouts and bolts

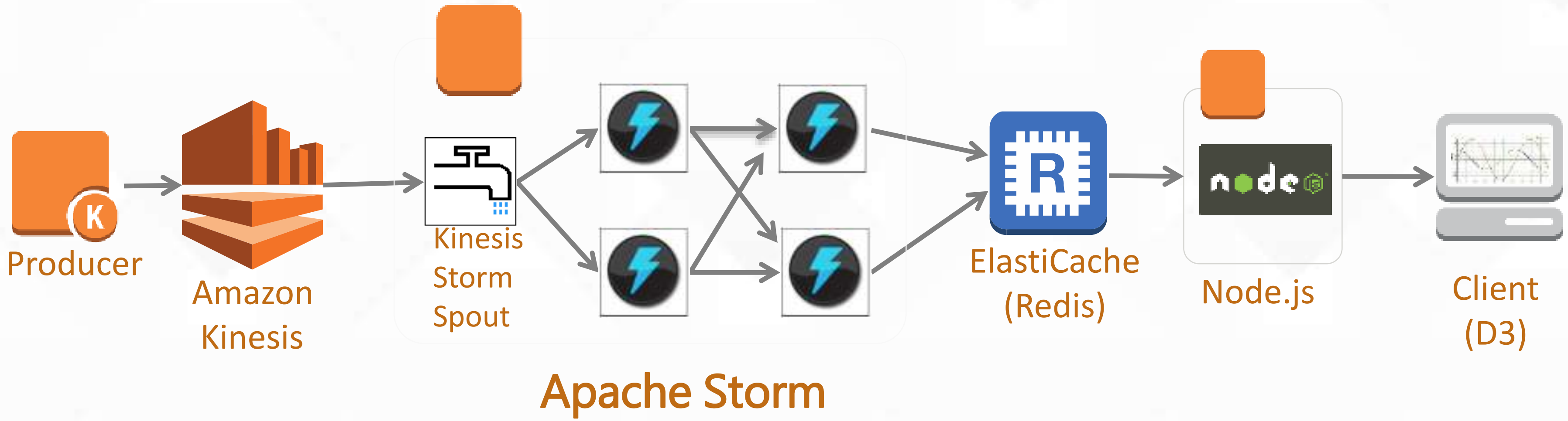


https://github.com/awslabs/kinesis-storm-spout

Storm architecture



Real-time: Event-based processing



http://blogs.aws.amazon.com/bigdata/post/Tx36LYSCY2R0A9B/Implement-a-Real-time-Sliding-Window-Application-Using-Amazon-Kinesis-and-Apache

You’re ~~likely~~ already “streaming”

- Embrace “stream thinking”
- Event Processing tools are available that will help increase your solutions’ functionality, availability and durability

Thank You

Questions?