Analyzing Big Data on the Fly

Shawn Gandhi, Solutions Architect
@shawnagram

amazon web services


18

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

Metering Record

```
127.0.0.1 user-identifier frank [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326
```

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"]
```

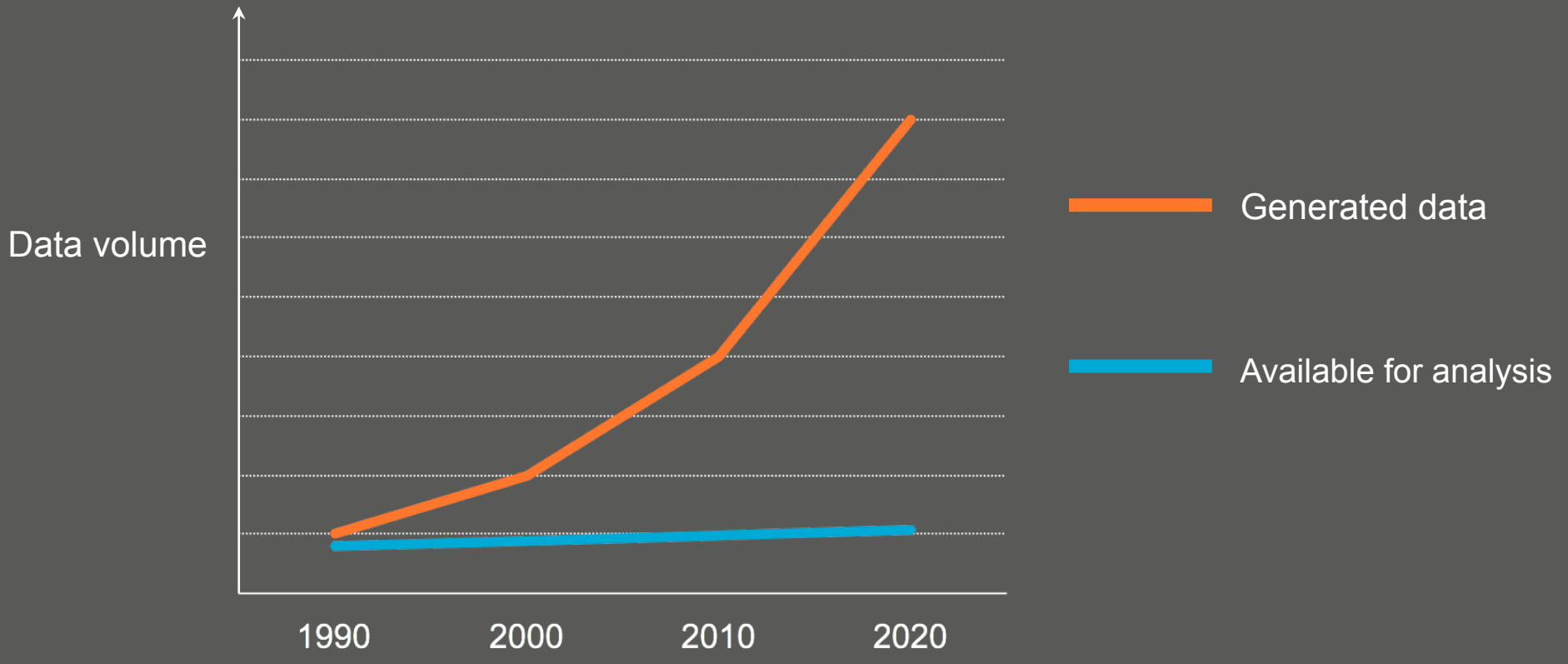
Syslog Entry

```
“SeattlePublicWater/Kinesis/123/Realtime”
— 412309129140
```

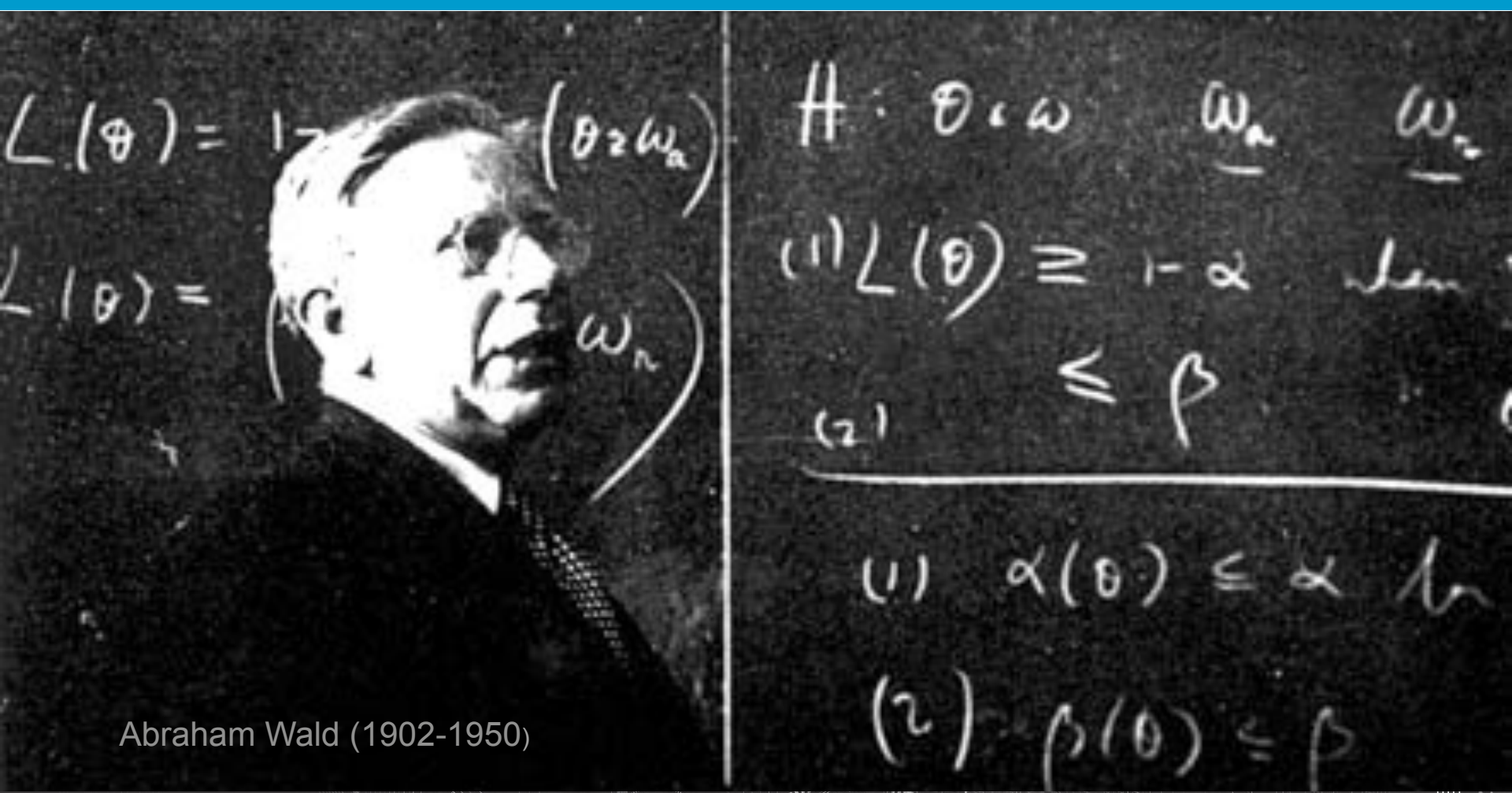
MQTT Record

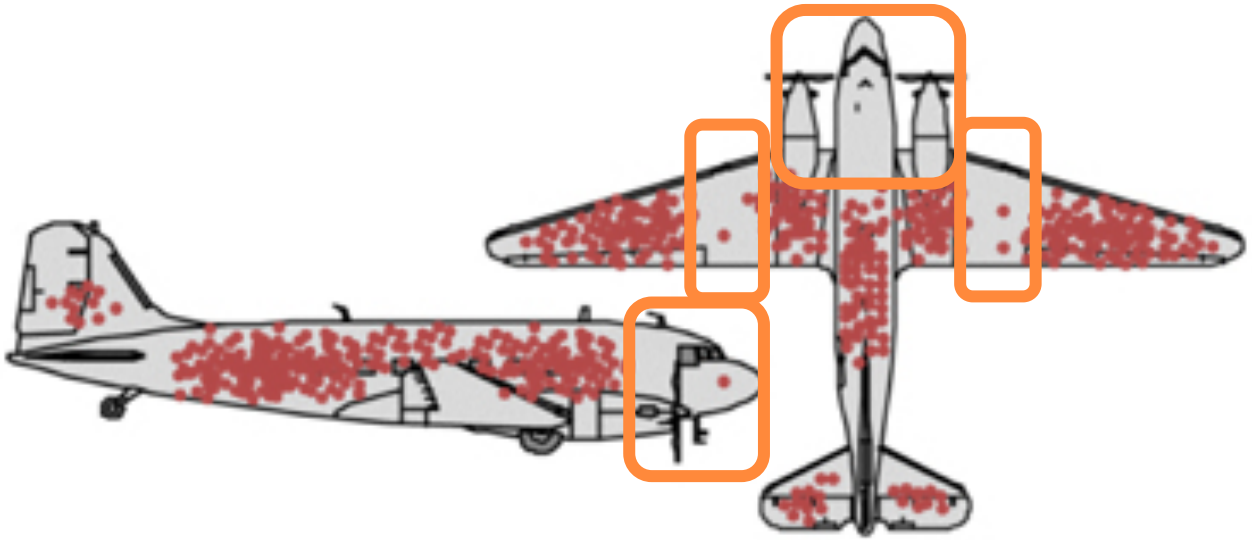
```
<R,AMZN    ,T,G,R1>
```

NASDAQ OMX Record



Gartner: User Survey Analysis: Key Trends Shaping the Future of Data Center Infrastructure Through 2011
IDC: Worldwide Business Analytics Software 2012–2016 Forecast and 2011 Vendor Shares


Abraham Wald (1902-1950)



Big Data: Best Served Fresh

Big Data

- Hourly server logs:
  were your systems misbehaving 1hr ago
- Weekly / Monthly Bill:
  what you spent this billing cycle
- Daily customer-preferences report from your web site's click stream:
  what deal or ad to try next time
- Daily fraud reports:
  was there fraud yesterday

Real-time Big Data

- Amazon CloudWatch metrics:
  what went wrong *now*
- Real-time spending alerts/caps:
  prevent overspending *now*
- Real-time analysis:
  what to offer the current customer *now*
- Real-time detection:
  block fraudulent use *now*

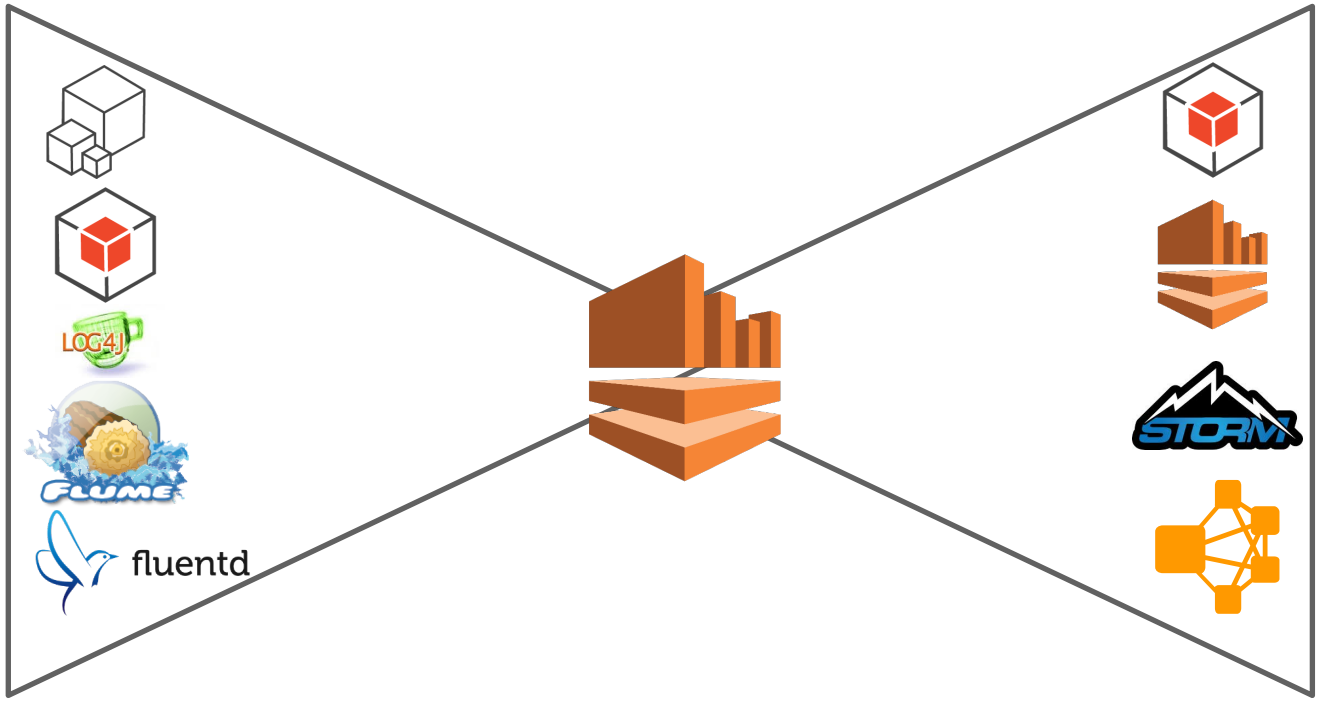
Talk outline

- A tour of Kinesis concepts in the context of a Twitter Trends service
- Implementing the Twitter Trends service
- Kinesis in the broader context of a Big Data eco-system

Why did we make this?

J.M. Turnauckas Photography
Highland Square, Akron

Sending & Reading Data from Kinesis Streams



http://bit.ly/1prQaWb

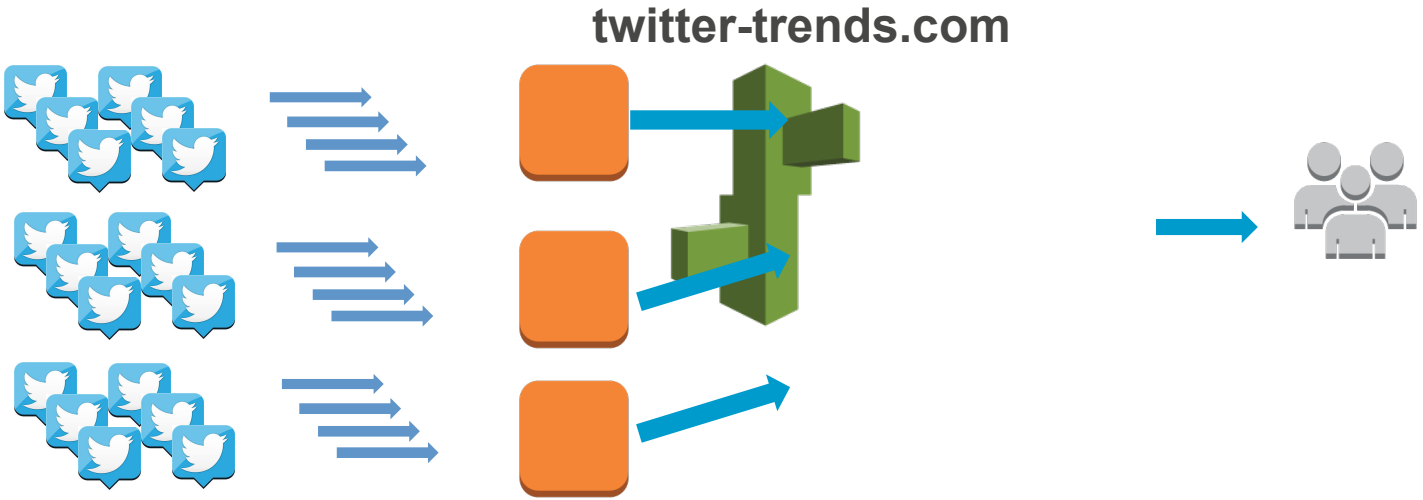
A tour of Kinesis concepts in the context of a Twitter Trends service

twitter-trends.com website

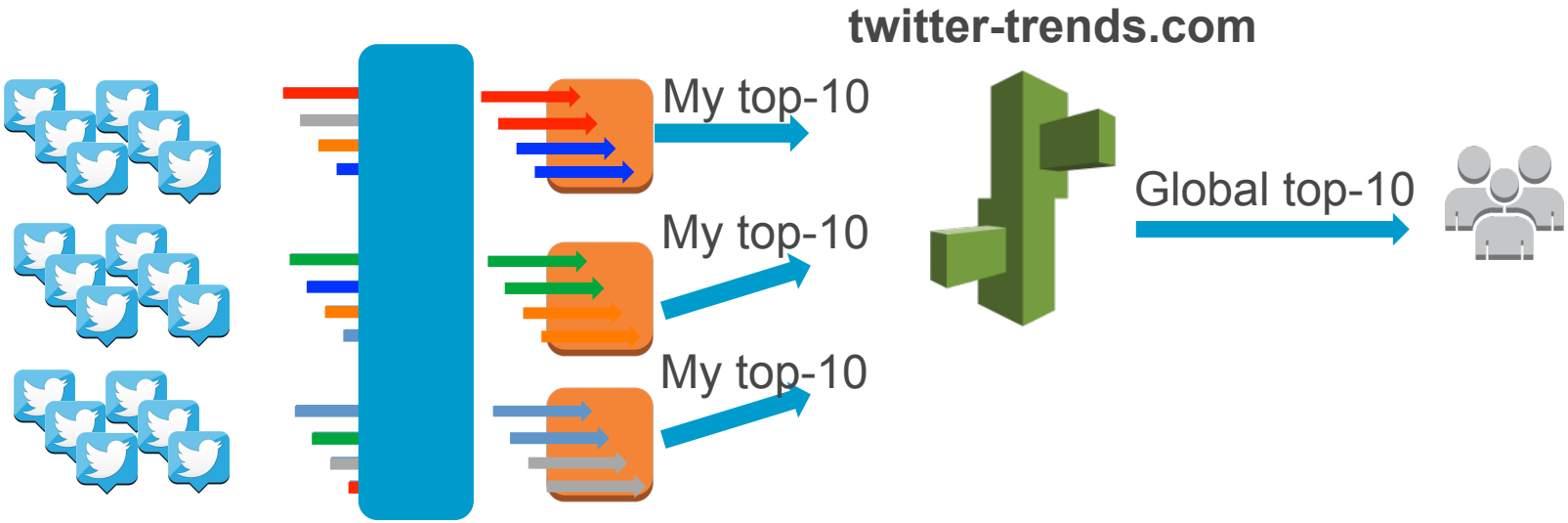
Elastic Beanstalk

twitter-trends.com

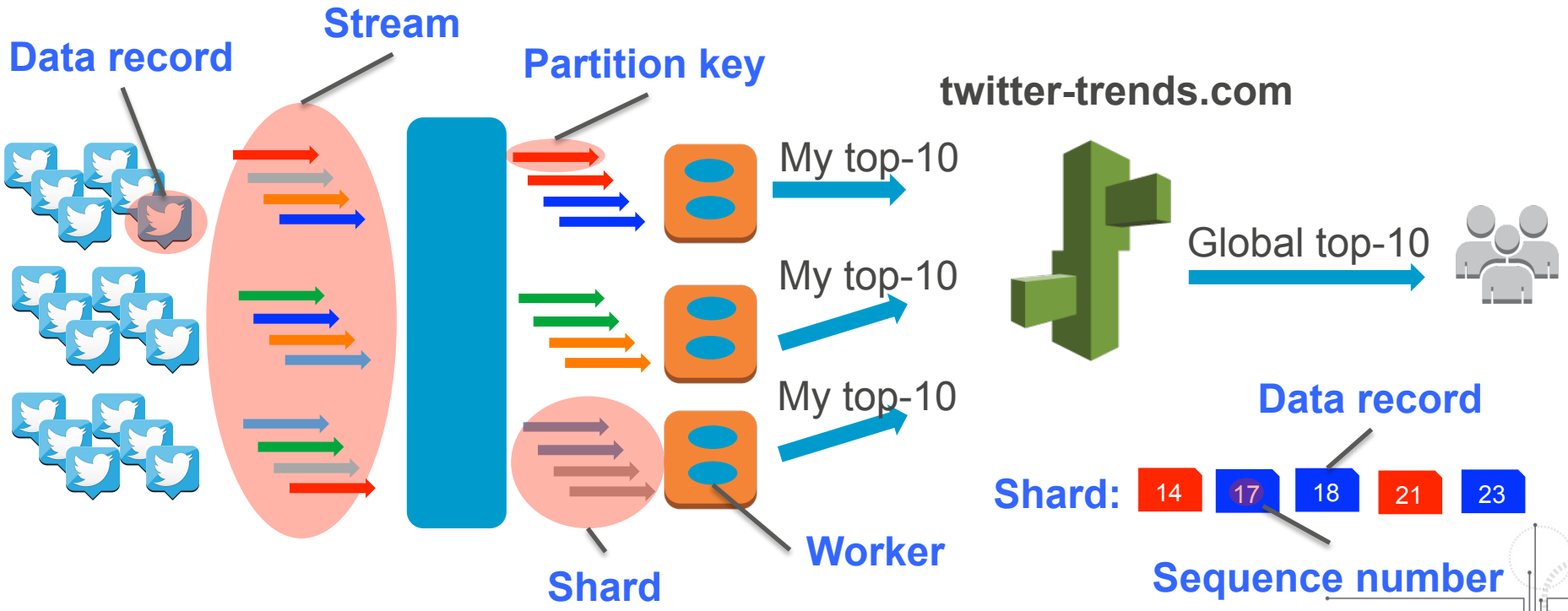
Too big to handle on one box



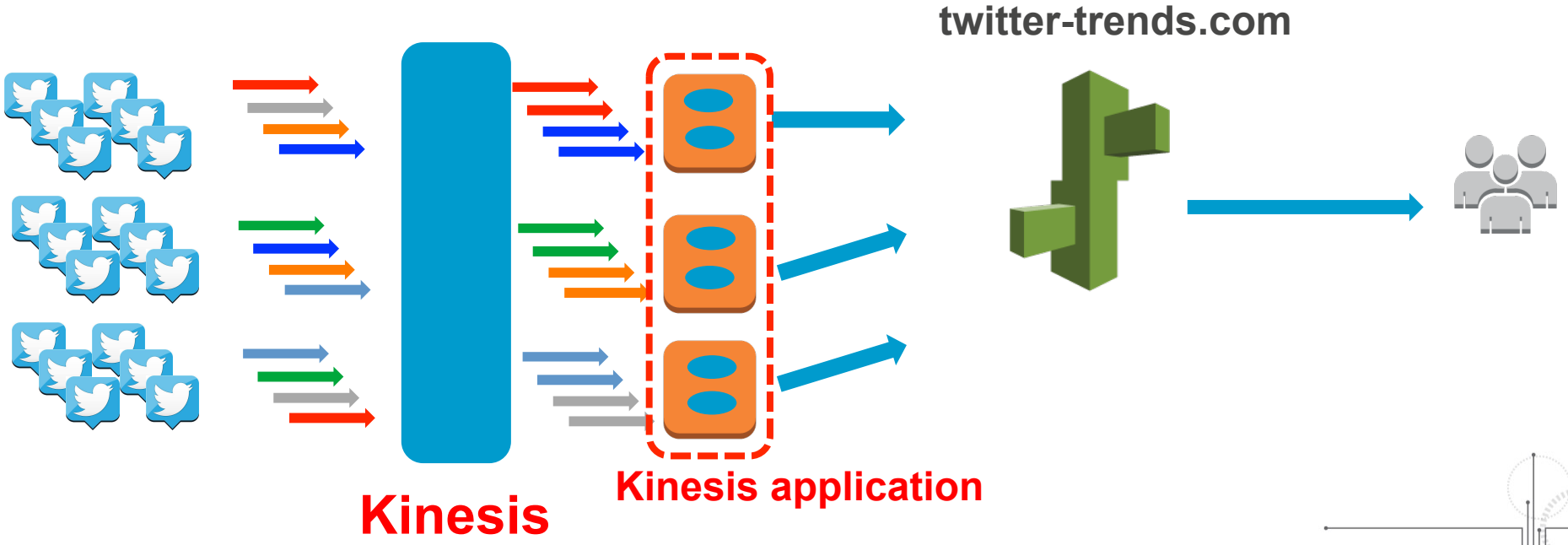
The solution: streaming map/reduce



Core concepts



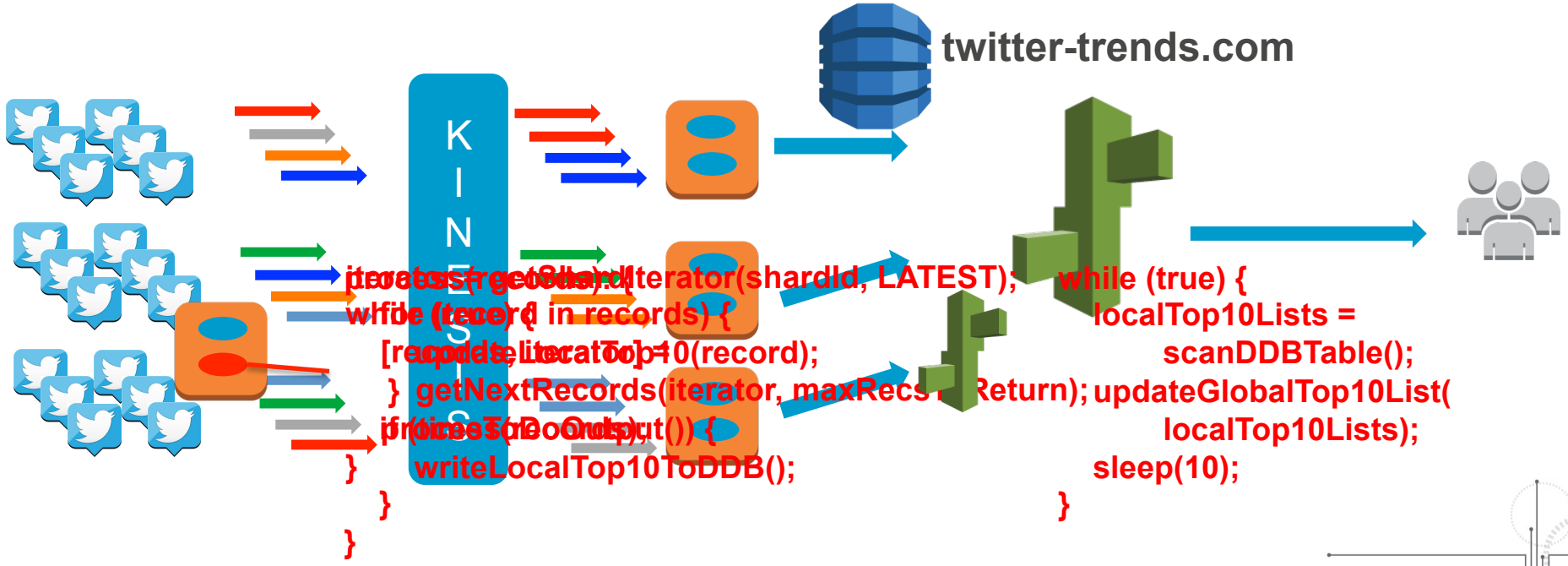
How this relates to Kinesis



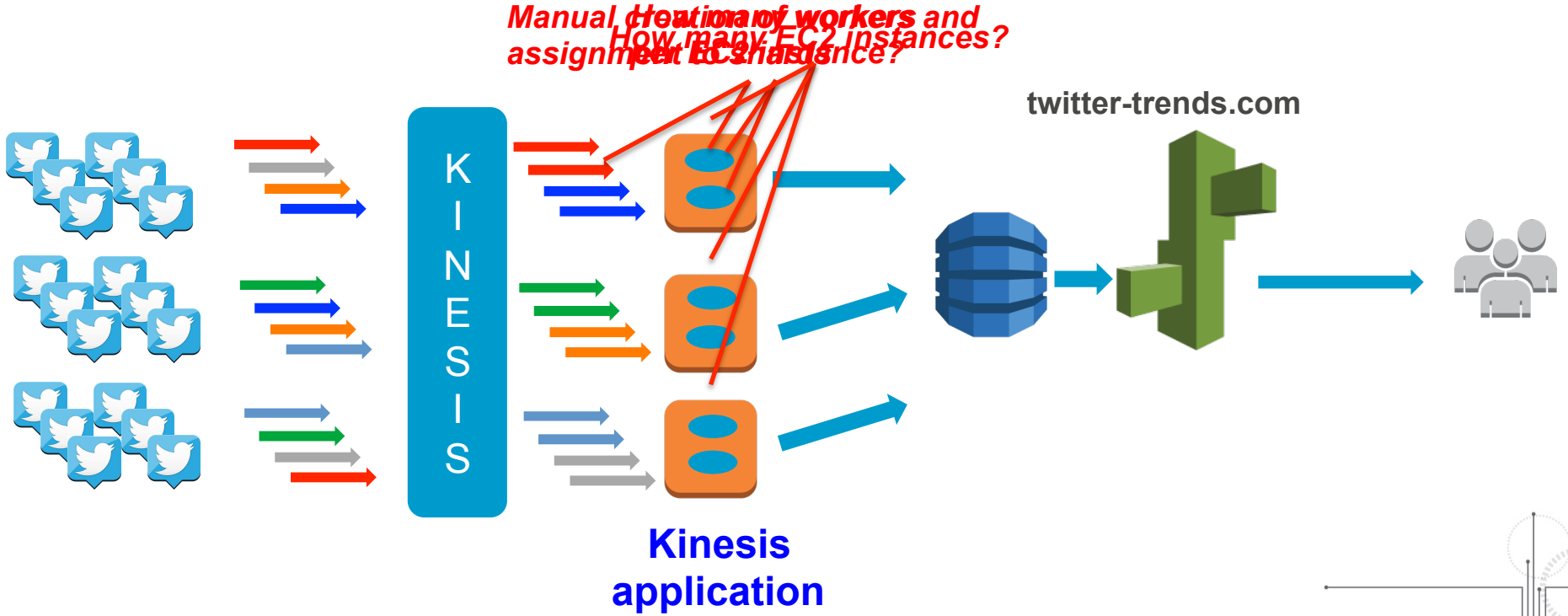
Core Concepts recapped

- **Data record** ~ tweet
- **Stream** ~ all tweets (the Twitter Firehose)
- **Partition key** ~ Twitter topic (every tweet record belongs to exactly one of these)
- **Shard** ~ all the data records belonging to a set of Twitter topics that will get grouped together
- **Sequence number** ~ each data record gets one assigned when first ingested.
- **Worker** ~ processes the records of a shard in sequence number order

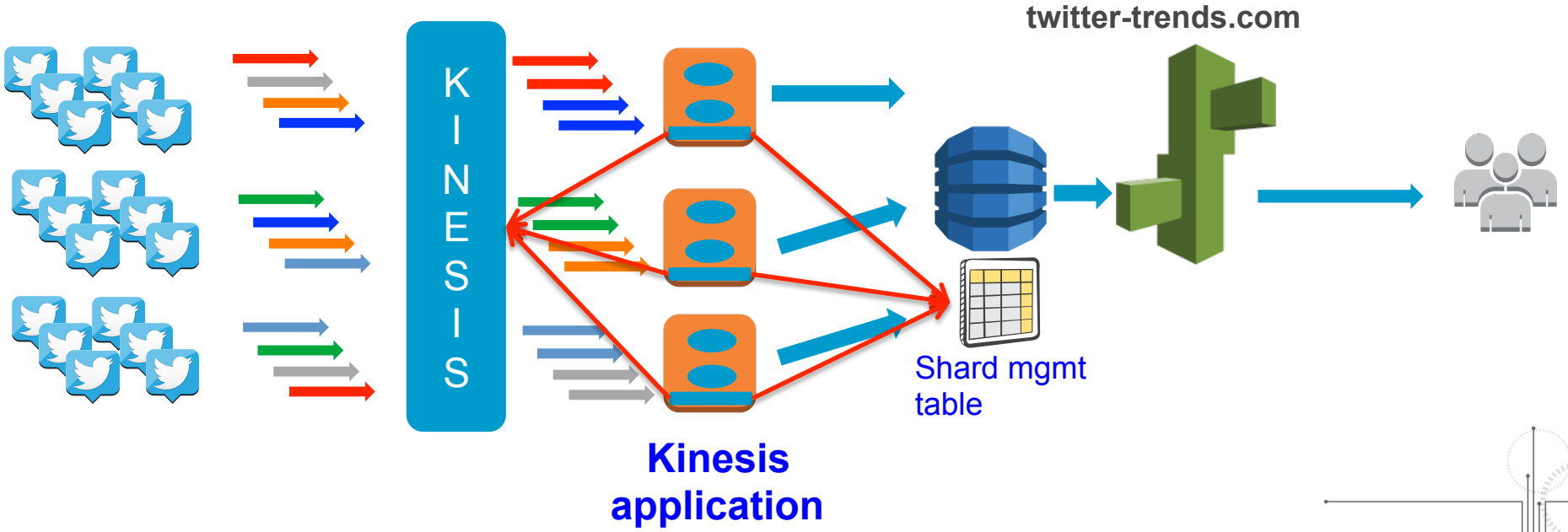
Using the Kinesis API directly



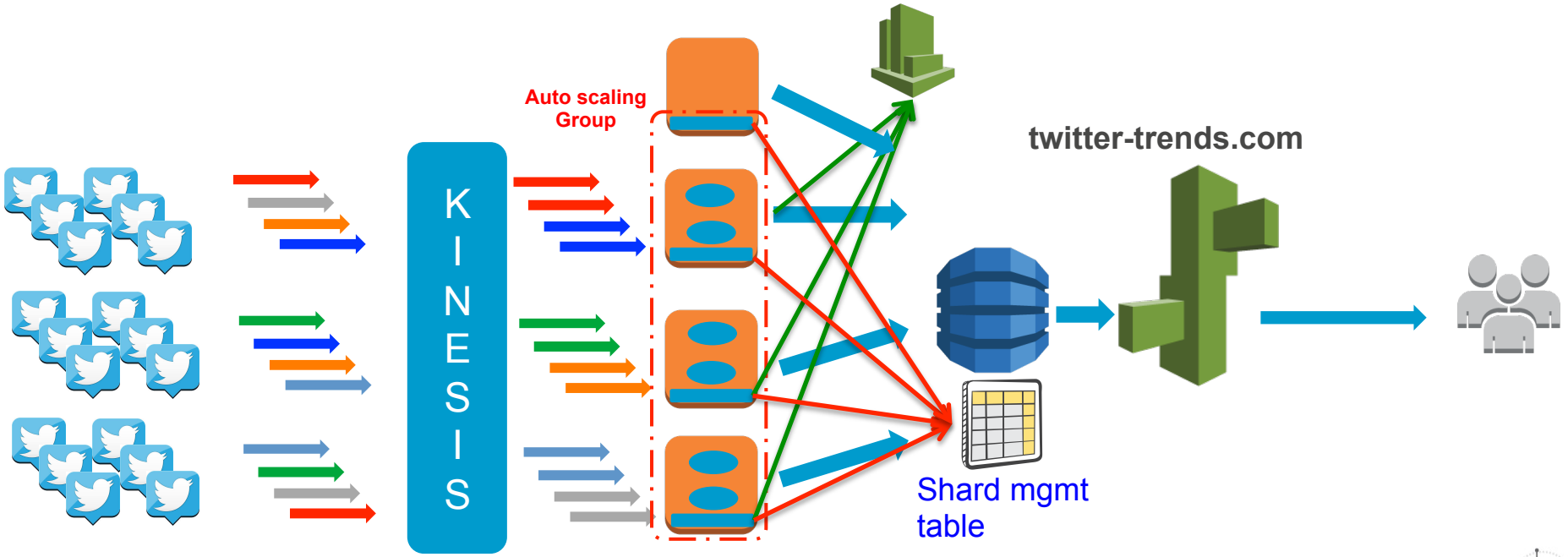
Challenges with using the Kinesis API directly



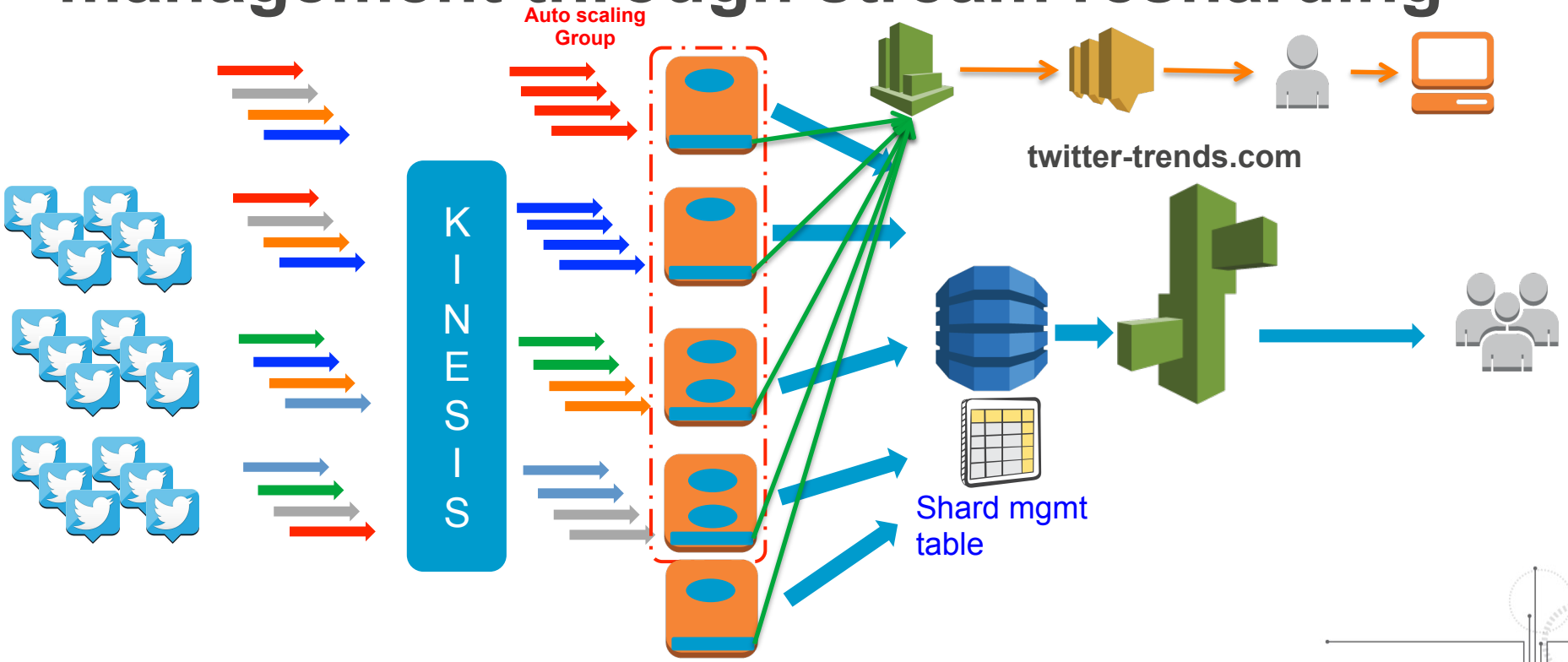
Using the Kinesis Client Library



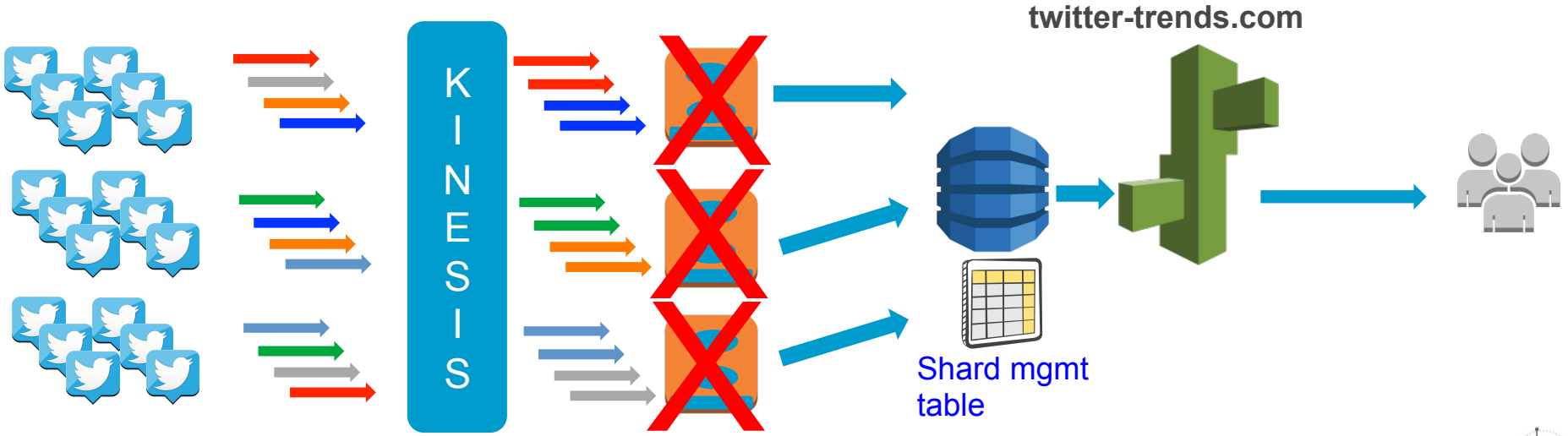
Elasticity and load balancing using Auto-Scaling Groups



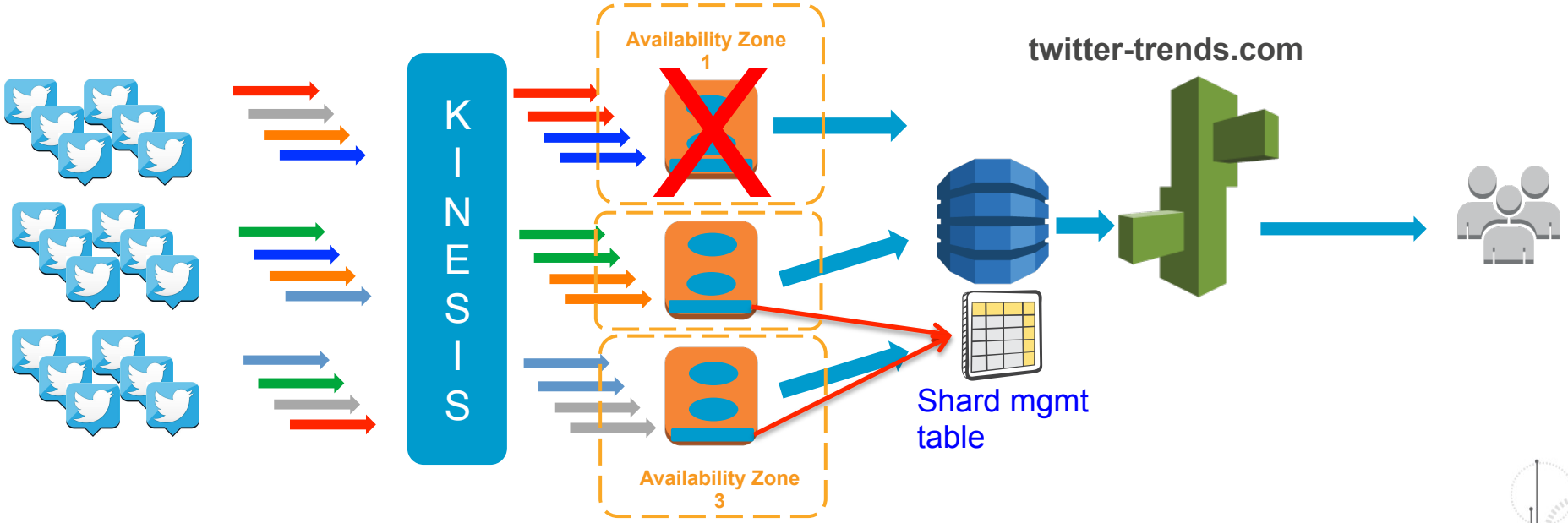
Dynamic growth and hot spot management through stream resharding



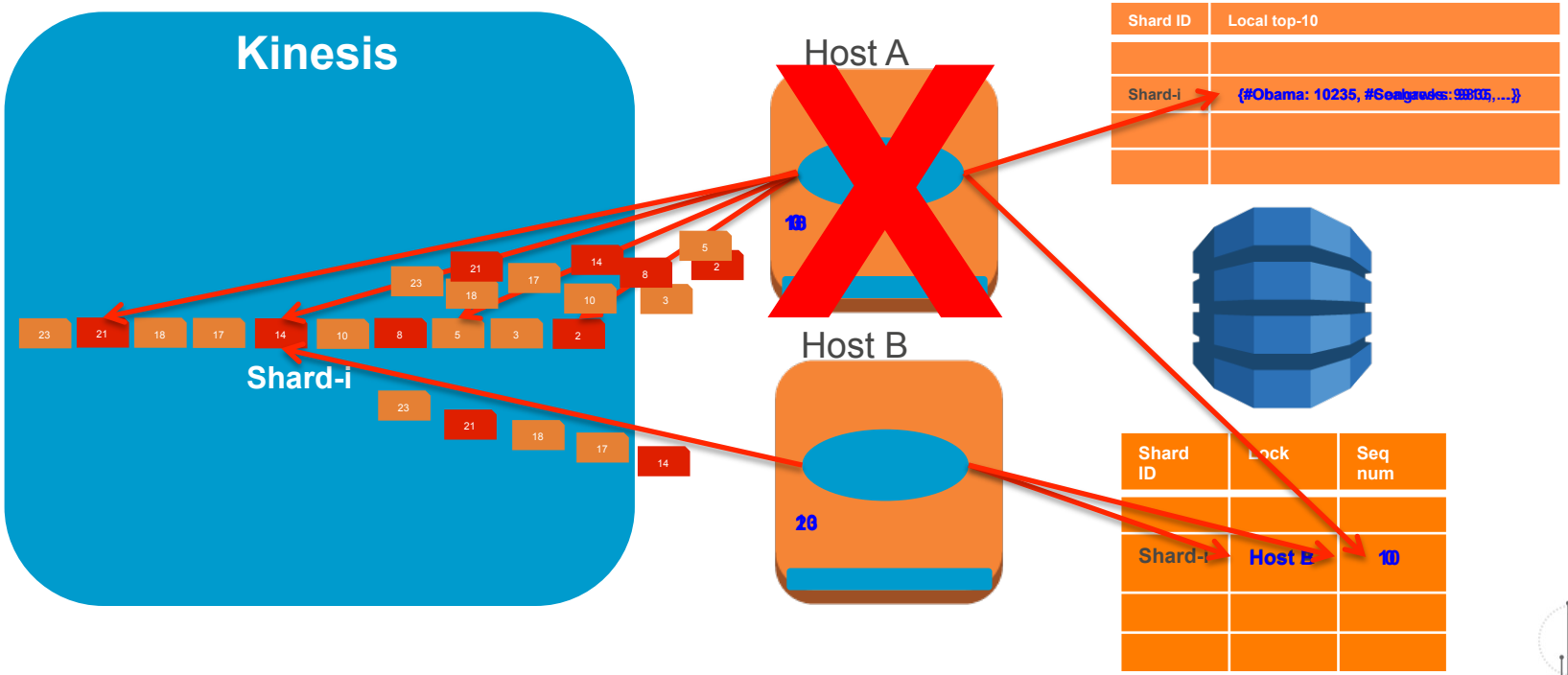
The challenges of fault tolerance



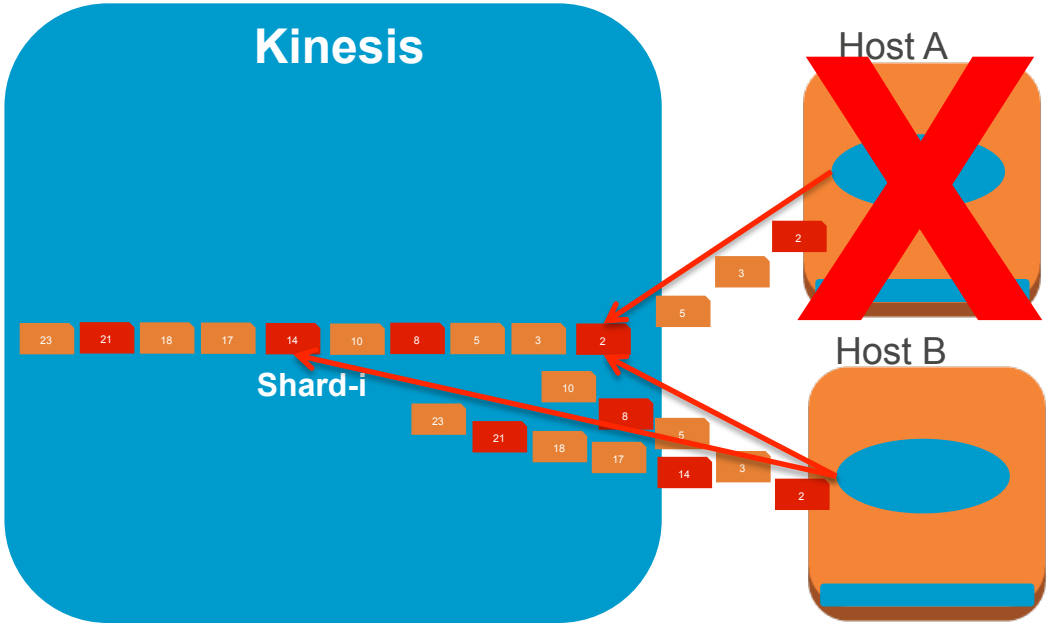
Fault tolerance support in KCL



Checkpoint, replay design pattern



Catching up from delayed processing



How long until we've caught up?

Shard throughput SLA:
- **1 MBps in**
- **2 MBps out**

=>

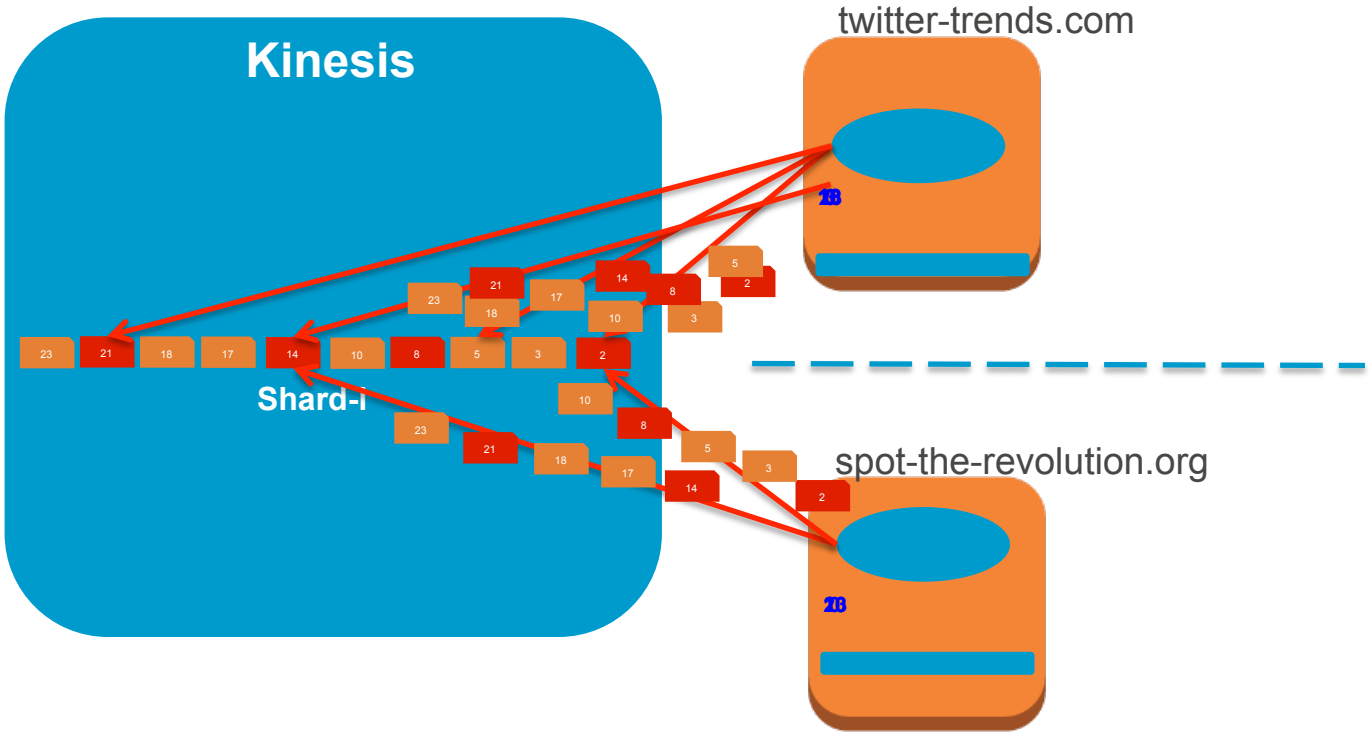
Catch up time ~ down time

Unless your application can't process 2 MBps on a single host

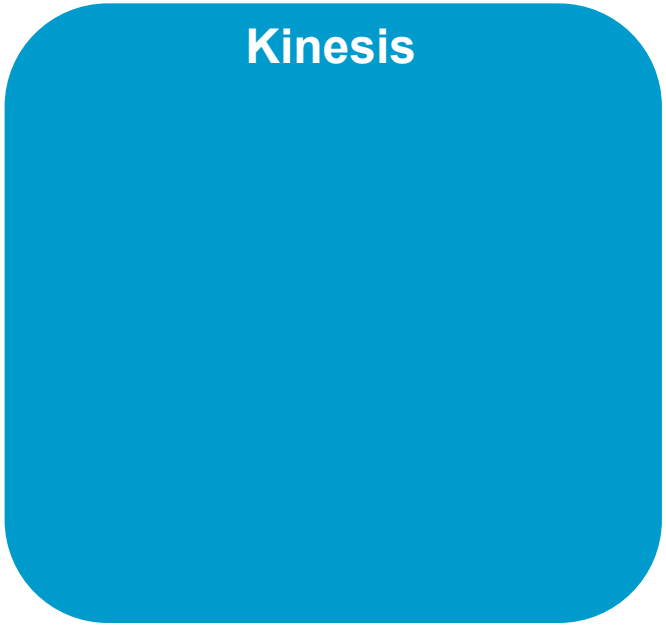
=>

Provision more shards

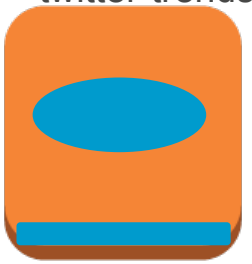
Concurrent processing applications



Why not combine applications?



twitter-trends.com



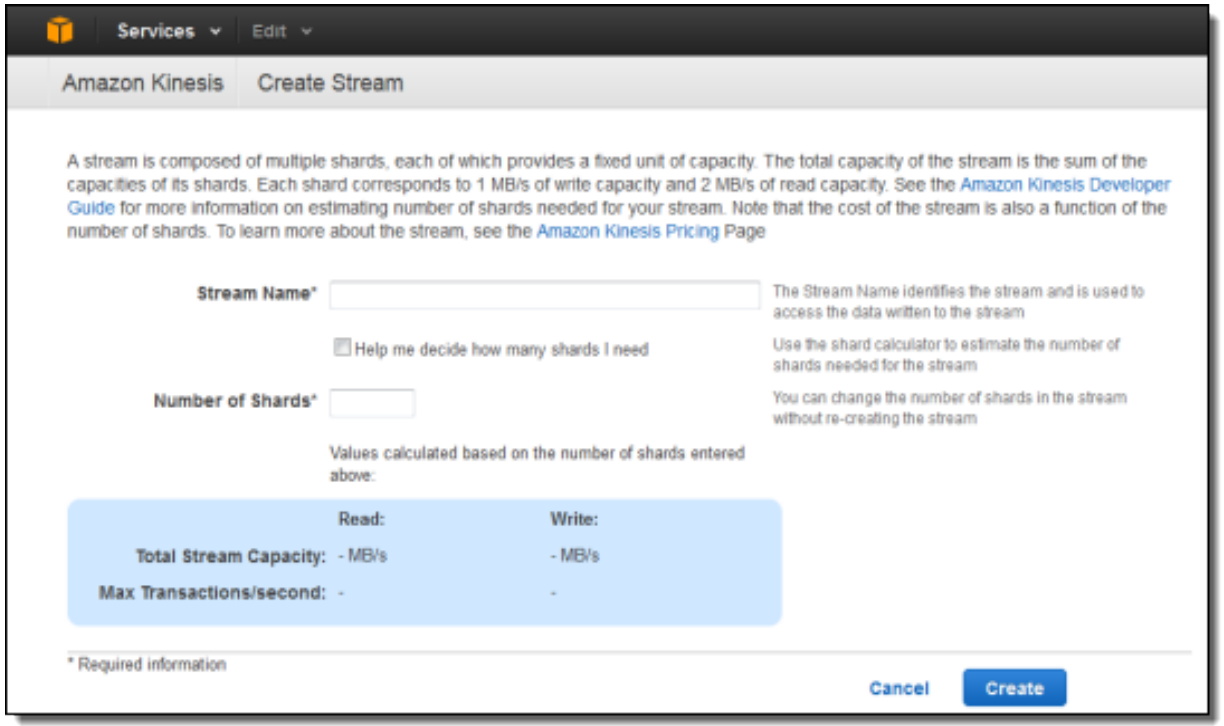
Output state & checkpoint every 10 secs

twitter-stats.com

Output state & checkpoint every hour

Implementing twitter-trends.com

Creating a Kinesis Stream

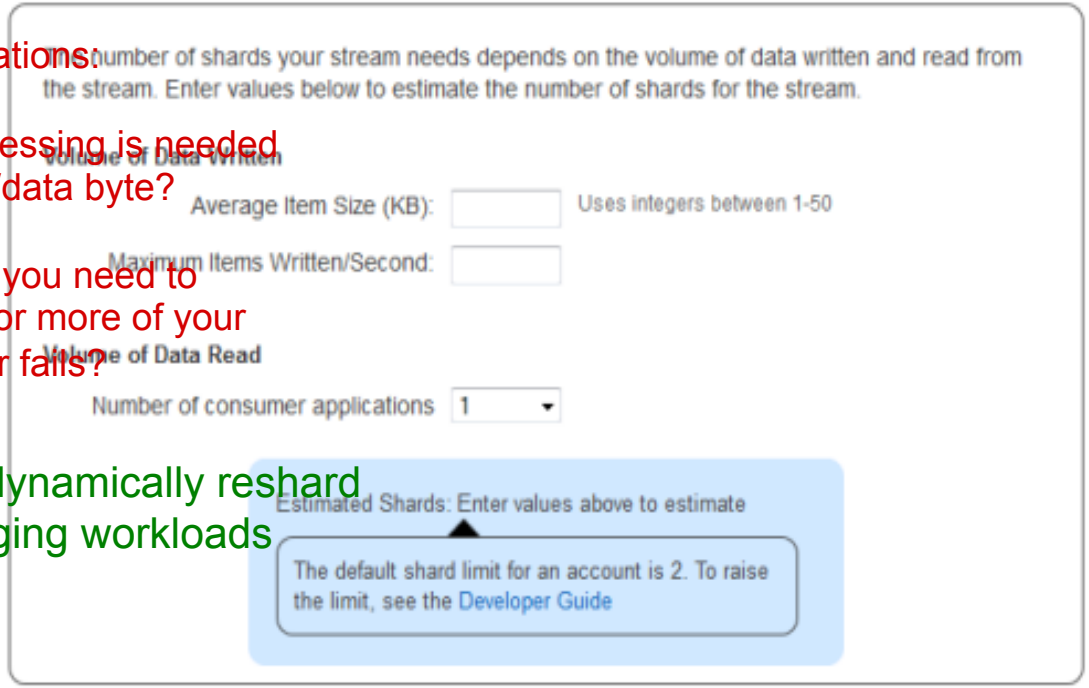


How many shards?

Additional considerations:

- How much processing is needed per data record/data byte?
- How quickly do you need to catch up if one or more of your workers stalls or fails?

You can always dynamically reshard to adjust to changing workloads



Twitter Trends Shard Processing Code

```
Class TwitterTrendsShardProcessor implements IRecordProcessor {

    public TwitterTrendsShardProcessor() { … }

    @Override
    public void initialize(String shardId) { … }

    @Override
    public void processRecords(List<Record> records,
                               IRecordProcessorCheckpointer checkpointer) { … }

    @Override
    public void shutdown(IRecordProcessorCheckpointer checkpointer,
                         ShutdownReason reason) { … }
}
```

Twitter Trends Shard Processing Code

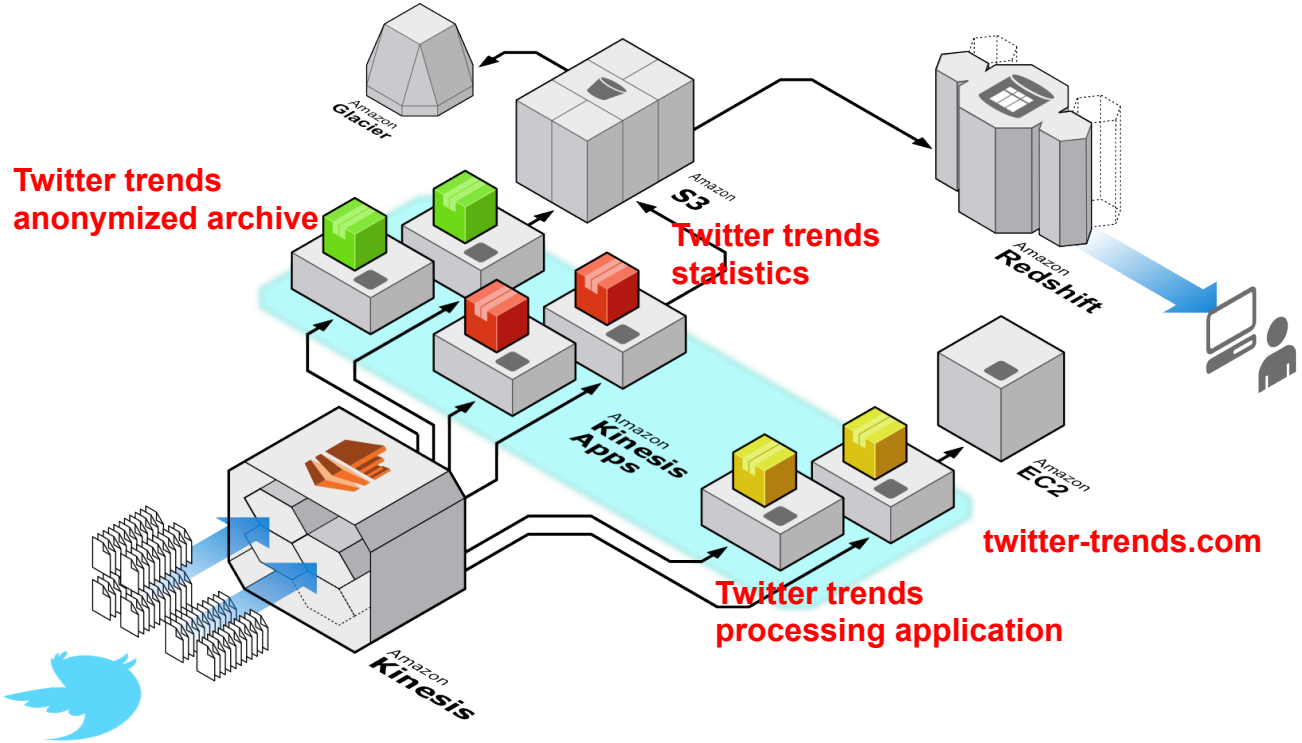
```
Class TwitterTrendsShardProcessor implements IRecordProcessor {

    private Map<String, AtomicInteger> hashCount = new HashMap<>();

    private long tweetsProcessed = 0;


    @Override
    public void processRecords(List<Record> records,
                               IRecordProcessorCheckpointer checkpointer) {
        computeLocalTop10(records);

        if ((tweetsProcessed++) >= 2000) {
            emitToDynamoDB(checkpointer);
        }
    }
}
```

Twitter Trends Shard Processing Code

```
private void computeLocalTop10(List<Record> records) {
    for (Record r : records) {
        String tweet = new String(r.getData().array());
        String[] words = tweet.split(" \t");
        for (String word : words) {
            if (word.startsWith("#")) {
                if (!hashCount.containsKey(word)) hashCount.put(word, new AtomicInteger(0));
                hashCount.get(word).incrementAndGet();
            }
        }
    }

private void emitToDynamoDB(IRecordProcessorCheckpointer checkpointer) {
    persistMapToDynamoDB(hashCount);
    try {
        checkpointer.checkpoint();
    } catch (IOException | KinesisClientLibDependencyException | InvalidStateException
        | ThrottlingException | ShutdownException e) {
            // Error handling
    }
    hashCount = new HashMap<>();
    tweetsProcessed = 0;
}
```

Kinesis as a gateway into the Big Data eco-system

Big Data has a Life Cycle



Connector framework

```
public interface IKinesisConnectorPipeline<T> {

    IEmitter<T> getEmitter(KinesisConnectorConfiguration configuration);

    IBuffer<T> getBuffer(KinesisConnectorConfiguration configuration);

    ITransformer<T> getTransformer(
                          KinesisConnectorConfiguration configuration);

    IFilter<T> getFilter(KinesisConnectorConfiguration configuration);
}
```

Transformer & Filter interfaces

```
public interface ITransformer<T> {

    public T toClass(Record record) throws IOException;

    public byte[] fromClass(T record) throws IOException;
}

public interface IFilter<T> {

    public boolean keepRecord(T record);
}
```

Tweet transformer for S3 archival

```
public class TweetTransformer implements ITransformer<TwitterRecord> {

    private final JsonFactory fact =
                JsonActivityFeedProcessor.getObjectMapper().getJsonFactory();

    @Override
    public TwitterRecord toClass(Record record) {
        try {
            return fact.createJsonParser(
                    record.getData().array()).readValueAs(TwitterRecord.class);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

Tweet transformer (cont.)

```
    @Override
    public byte[] fromClass(TwitterRecord r) {
        SimpleDateFormat df = new SimpleDateFormat("YYYY-MM-dd HH:MM:SS");
        StringBuilder b = new StringBuilder();

        b.append(r.getActor().getId()).append("|")
        .append(sanitize(r.getActor().getDisplayName())).append("|")
        .append(r.getActor().getFollowersCount()).append("|")
        .append(r.getActor().getFriendsCount()).append("|")
        ...
        .append('\n');

        return b.toString().replaceAll("\ufffd\ufffd\ufffd", "").getBytes();
    }
}
```

Buffer interface

```
public interface IBuffer<T> {

    public long getBytesToBuffer();

    public long getNumRecordsToBuffer();

    public boolean shouldFlush();

    public void consumeRecord(T record, int recordBytes, String sequenceNumber);

    public void clear();

    public String getFirstSequenceNumber();

    public String getLastSequenceNumber();

    public List<T> getRecords();
}
```

Tweet aggregation buffer for Redshift

```
...

@Override
consumeRecord(TwitterAggRecord record, int recordSize, String sequenceNumber) {
     if (buffer.isEmpty()) {
          firstSequenceNumber = sequenceNumber;
     }
     lastSequenceNumber = sequenceNumber;
     TwitterAggRecord agg = null;
     if (!buffer.containsKey(record.getHashTag())) {
          agg = new TwitterAggRecord();
          buffer.put(agg);
          byteCount.addAndGet(agg.getRecordSize());
     }
     agg = buffer.get(record.getHashTag());
     agg.aggregate(record);
}
```

Tweet Redshift aggregation transformer

```
public class TweetTransformer implements ITransformer<TwitterAggRecord> {

    private final JsonFactory fact =
                JsonActivityFeedProcessor.getObjectMapper().getJsonFactory();

    @Override
    public TwitterAggRecord toClass(Record record) {
        try {
            return new TwitterAggRecord(
                    fact.createJsonParser(
                    record.getData().array()).readValueAs(TwitterRecord.class));
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
```

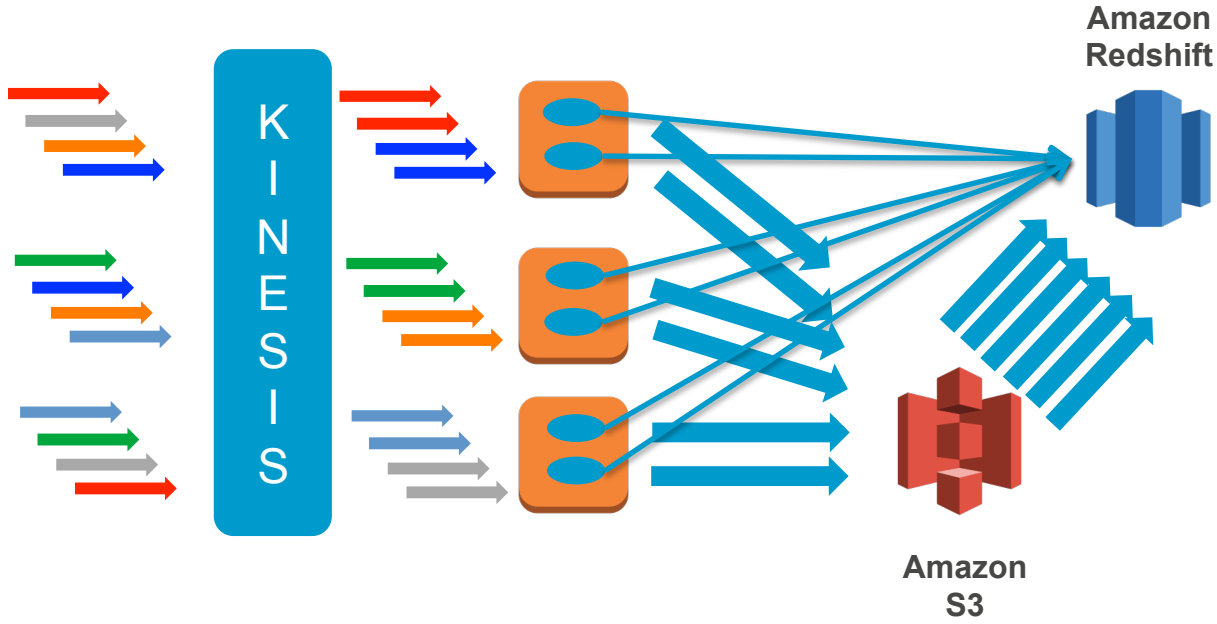
Tweet aggregation transformer (cont.)

```
    @Override
    public byte[] fromClass(TwitterAggRecord r) {
        SimpleDateFormat df = new SimpleDateFormat("YYYY-MM-dd HH:MM:SS");
        StringBuilder b = new StringBuilder();

        b.append(r.getActor().getId()).append("|")
        .append(sanitize(r.getActor().getDisplayName())).append("|")
        .append(r.getActor().getFollowersCount()).append("|")
        .append(r.getActor().getFriendsCount()).append("|")
        ...
        .append('\n');

        return b.toString().replaceAll("\ufffd\ufffd\ufffd", "").getBytes();
    }
}
```

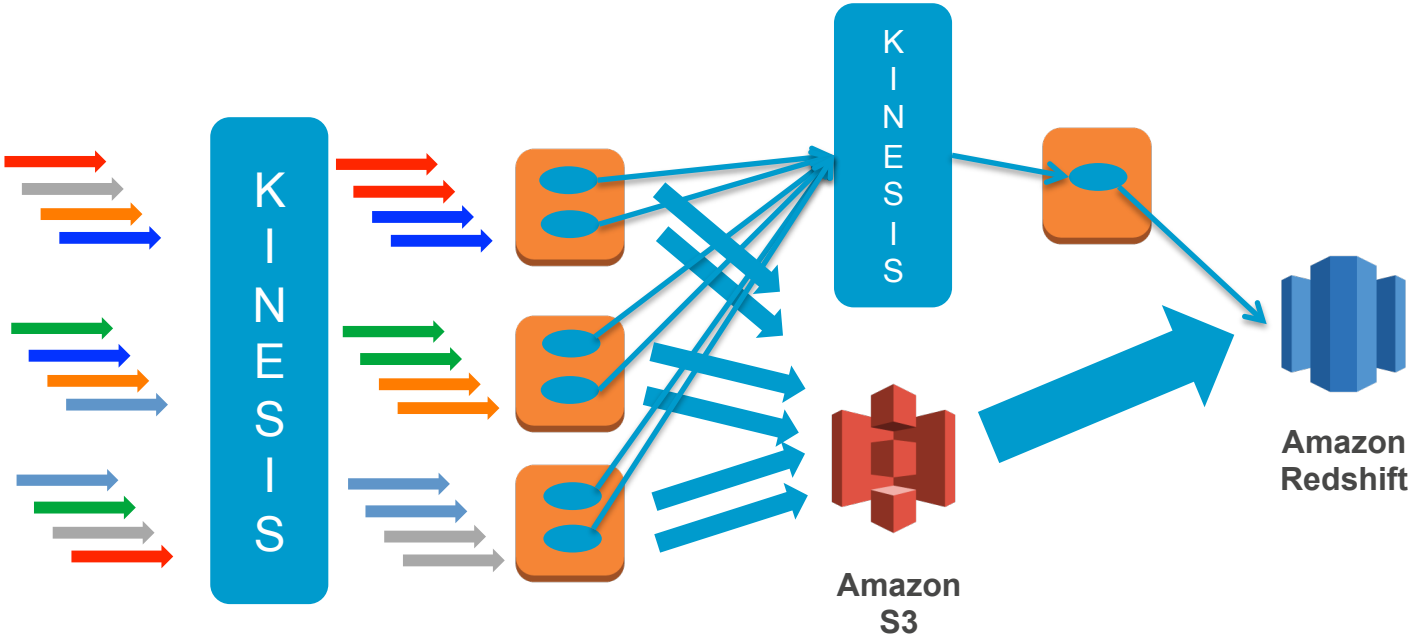
Emitter interface

```
public interface IEmitter<T> {

    List<T> emit(UnmodifiableBuffer<T> buffer) throws IOException;

    void fail(List<T> records);

    void shutdown();
}
```

Example Redshift connector



Example Redshift connector – v2



Amazon Kinesis: Key Developer Benefits

Easy Administration

Managed service for real-time streaming data collection, processing and analysis. Simply create a new stream, set the desired level of capacity, and let the service handle the rest.

Real-time Performance

Perform continual processing on streaming big data. Processing latencies fall to a few seconds, compared with the minutes or hours associated with batch processing.

Elastic

Seamlessly scale to match your data throughput rate and volume. You can easily scale up to gigabytes per second. The service will scale up or down based on your operational or business needs.

S3, Redshift, & DynamoDB Integration

Reliably collect, process, and transform all of your data in real-time & deliver to AWS data stores of choice, with Connectors for S3, Redshift, and DynamoDB.

Build Real-time Applications

Client libraries that enable developers to design and operate real-time streaming data processing applications.

Low Cost

Cost-efficient for workloads of any scale. You can get started by provisioning a small stream, and pay low hourly rates only for what you use.

Thank You

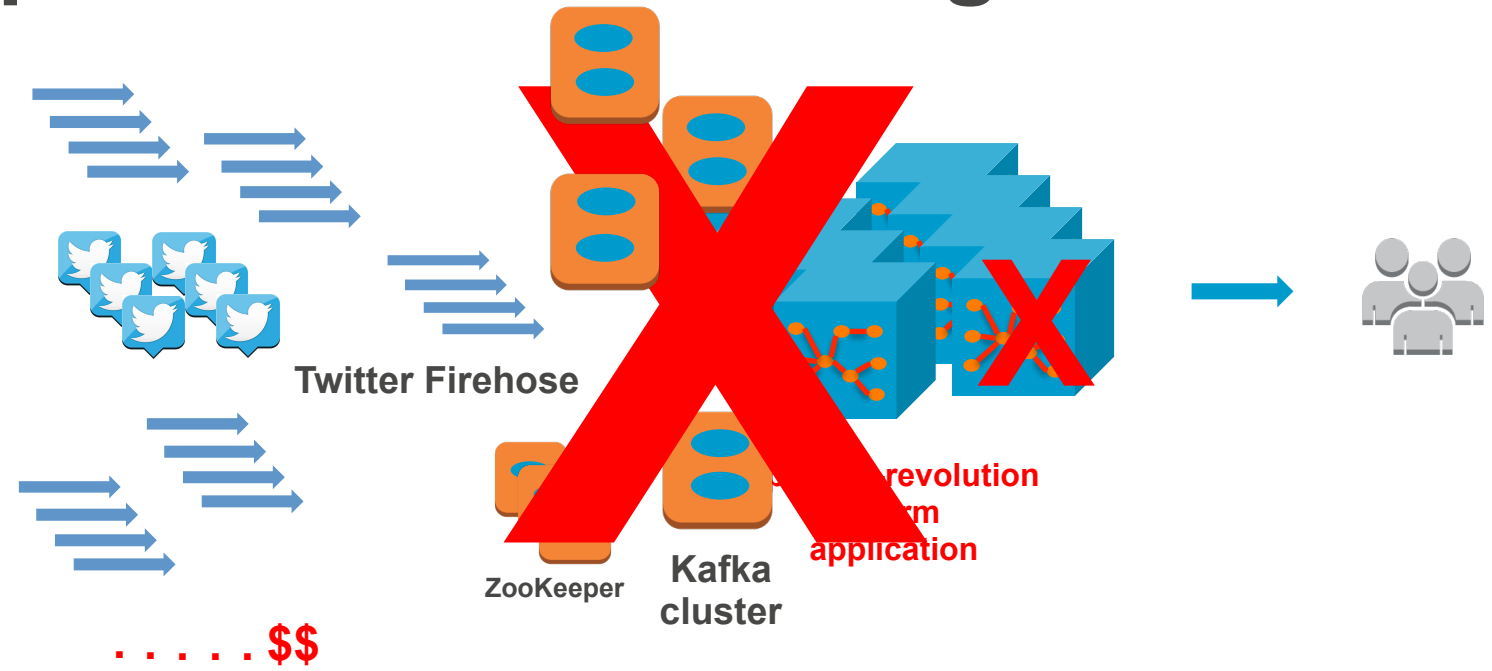
You can follow us at

http://aws.amazon.com/kinesis

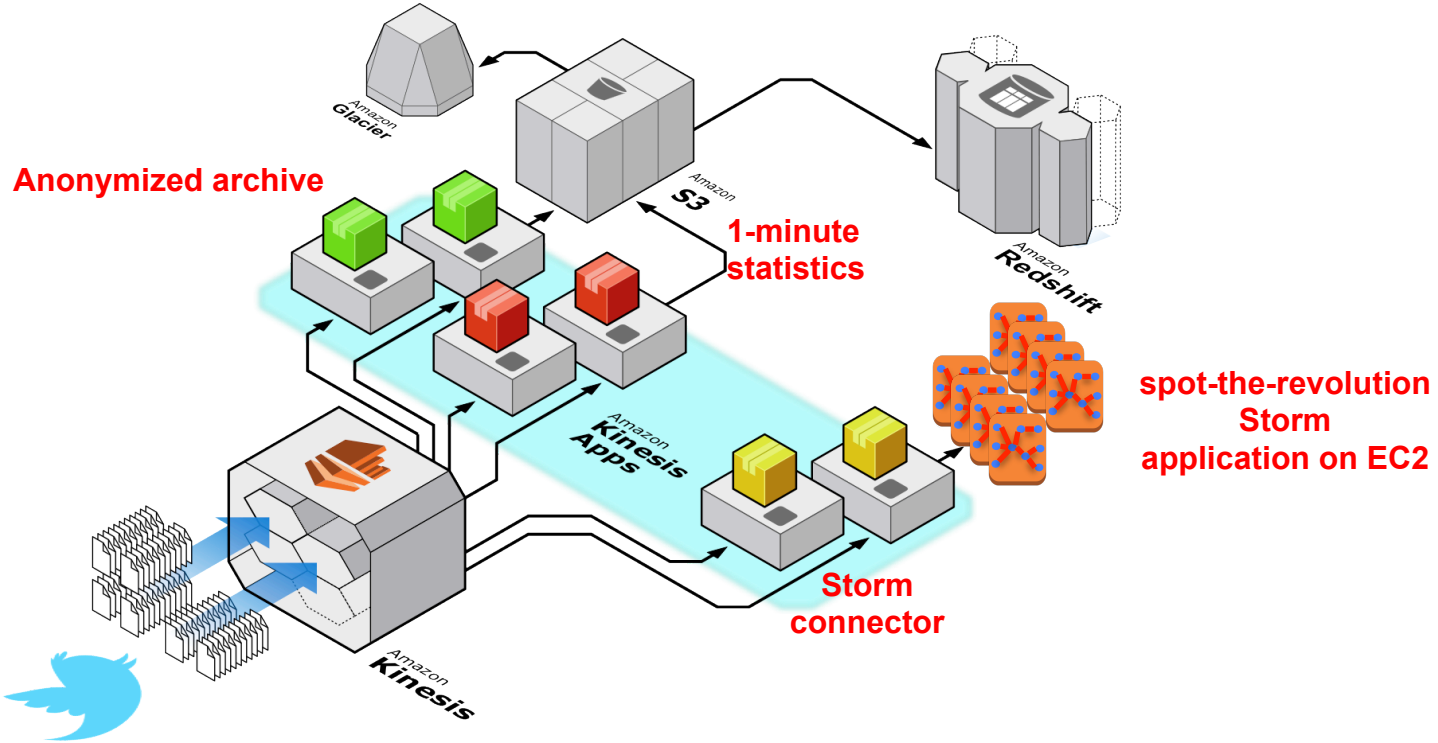
@shawnagram #kinesis

Implementing spot-the-revolution.org

spot-the-revolution.org – version 1

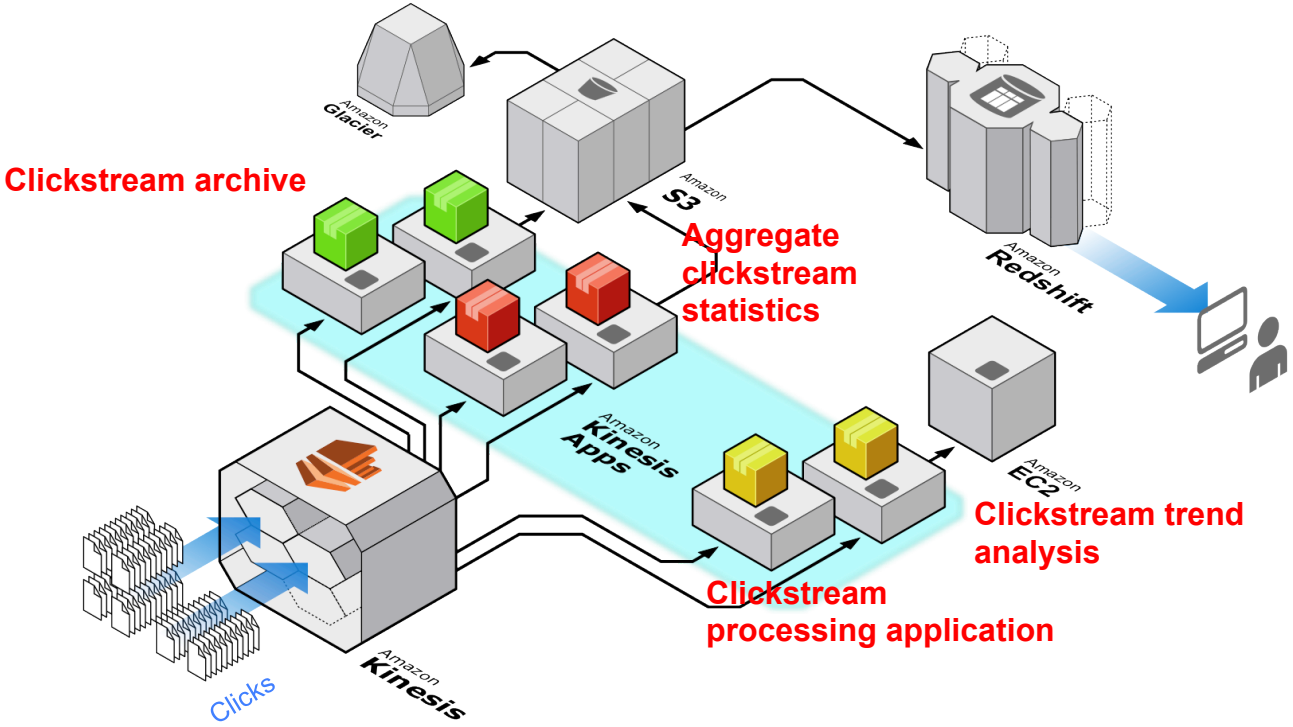


spot-the-revolution.org – Kinesis version



Generalizing the design pattern

Clickstream analytics example



Simple metering & billing example