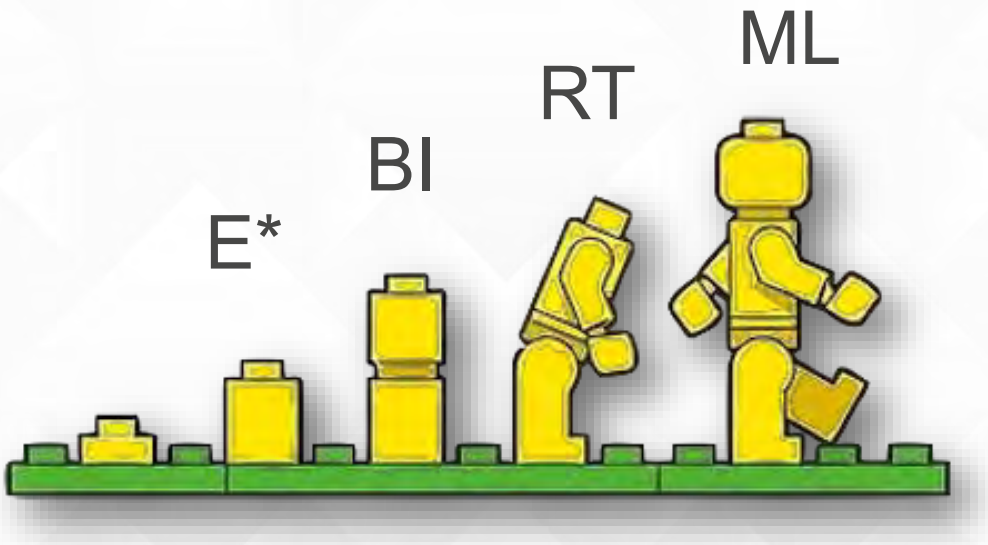
Deep Dive – Amazon Kinesis

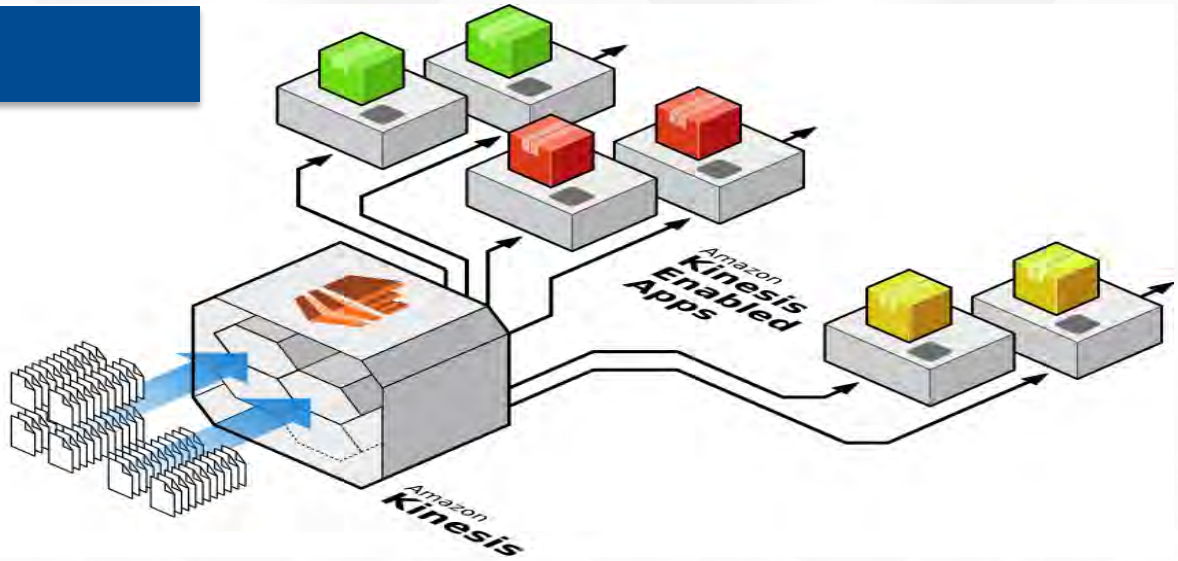
Guy Ernest, Solution Architect - Amazon Web Services
@guyernest

amazon web services

Motivation for Real Time Analytics



Analytics



- Deployment & Administration
- App Services
- **Analytics**
- Compute
- Storage
- Database
- Networking
- AWS Global Infrastructure

Amazon Kinesis

Managed Service for Real Time Big Data Processing

Create Streams to Produce & Consume Data

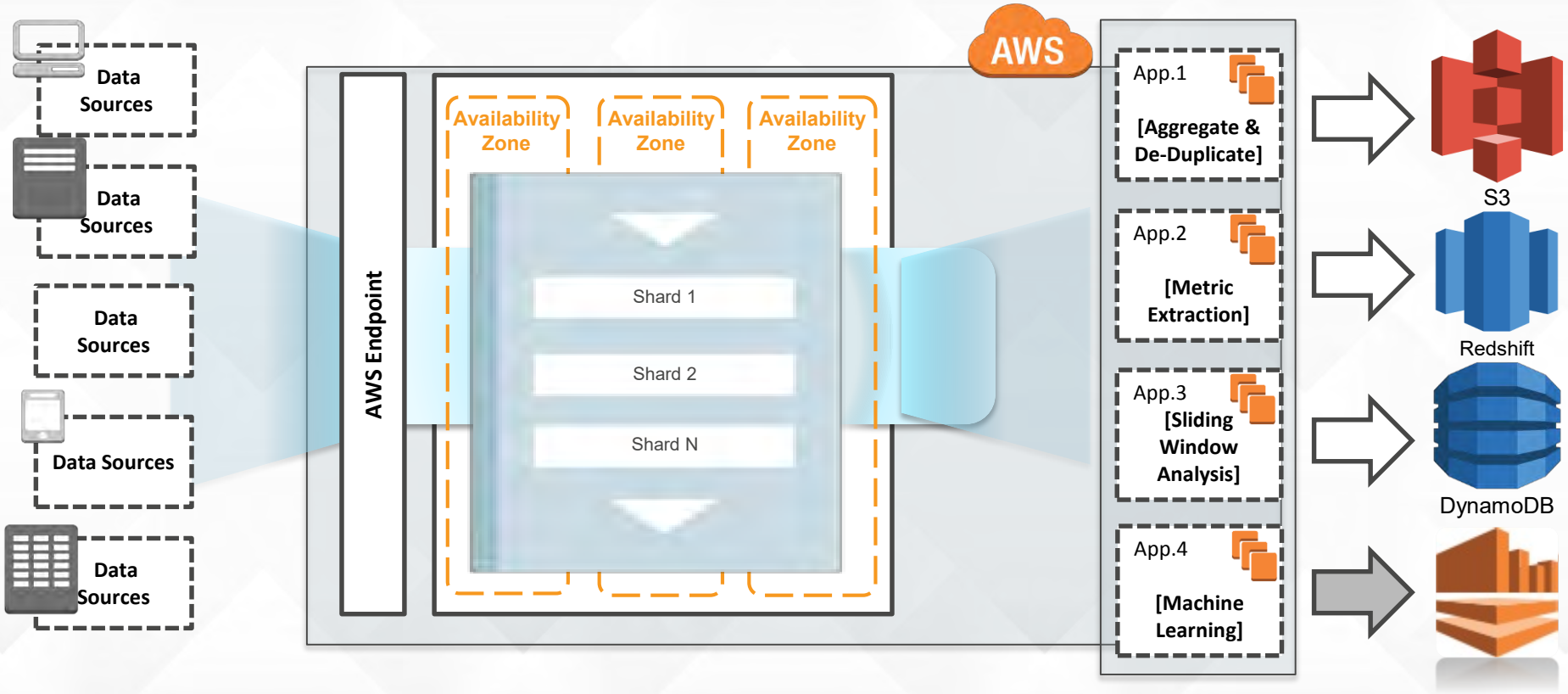
Elastically Add and Remove Shards for Performance

Use Kinesis Worker Library, AWS Lambda, Apache Spark and Apache Storm to Process Data

Integration with S3, Redshift and Dynamo DB

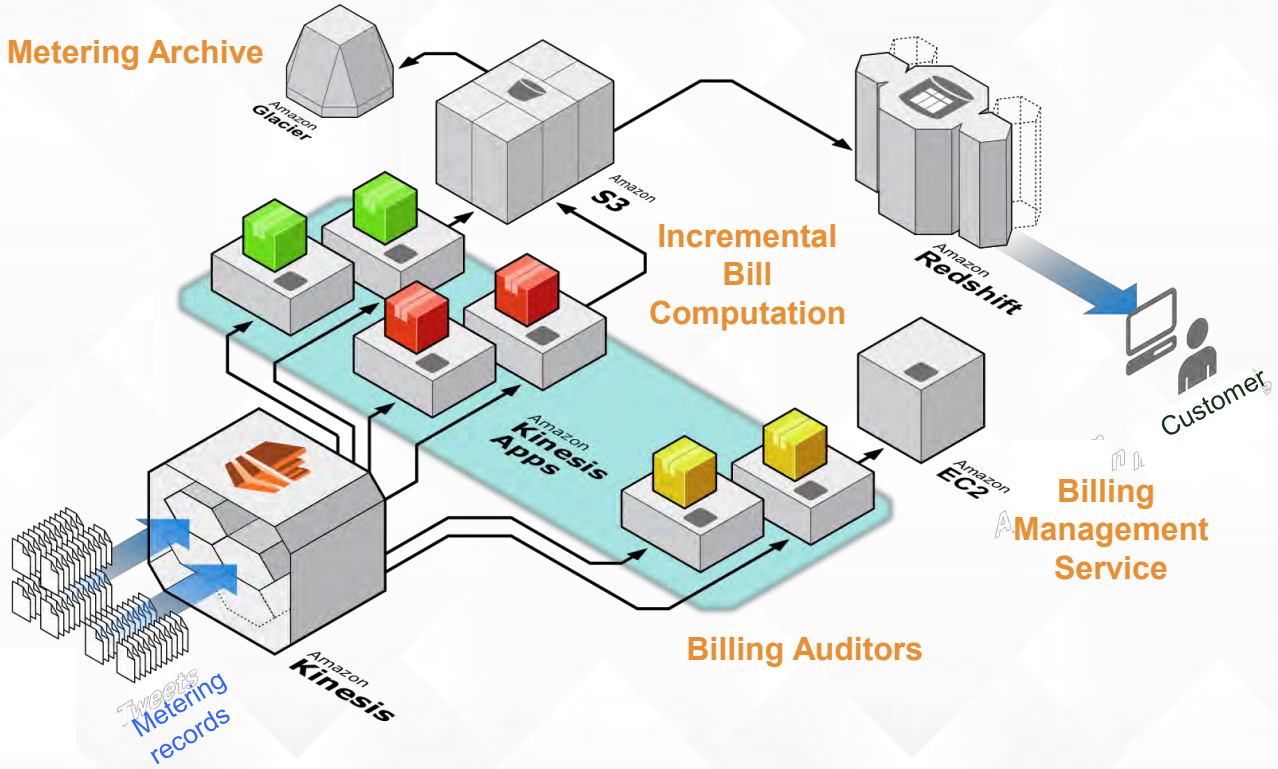
amazon web services

Amazon Kinesis Dataflow



amazon web services

Example Architecture - Metering



amazon web services

Amazon Kinesis Components

Streams

- Named Event Streams of Data
- All data is stored for 24 hours

Shards

- You scale Kinesis streams by adding or removing Shards
- Each Shard ingests up to 1MB/sec of data and up to 1000 TPS

Partition Key

- Identifier used for Ordered Delivery & Partitioning of Data across Shards

Sequence

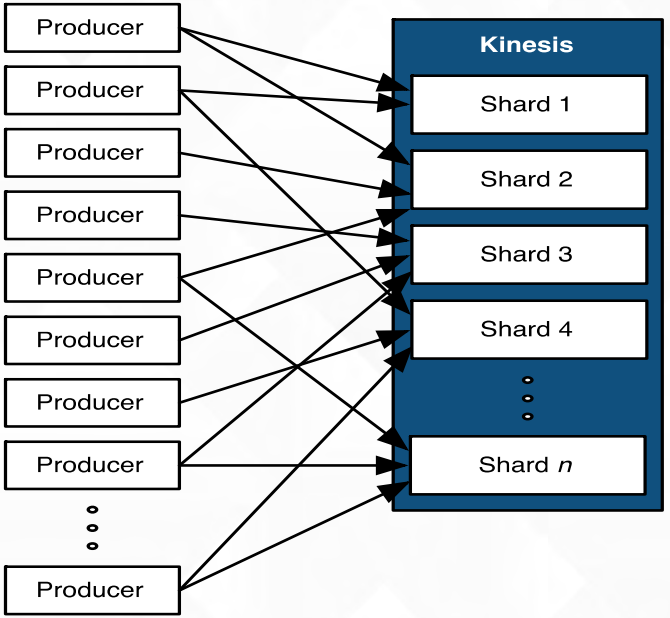
- Number of an event as assigned by Kinesis

Getting Data In

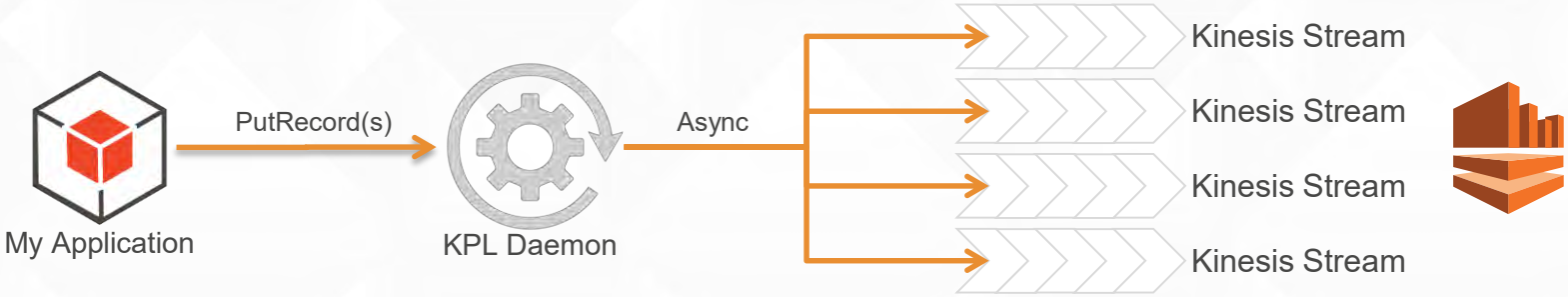
amazon web services

Kinesis - Ingesting Fast Moving Data

- Producers use a PUT call to store data in a Stream
- A Partition Key is used to distribute the PUTs across Shards
- A unique Sequence # is returned to the Producer for each Event
- Data can be ingested at 1MB/second or 1000 Transactions/second per Shard
- 1MB / Event

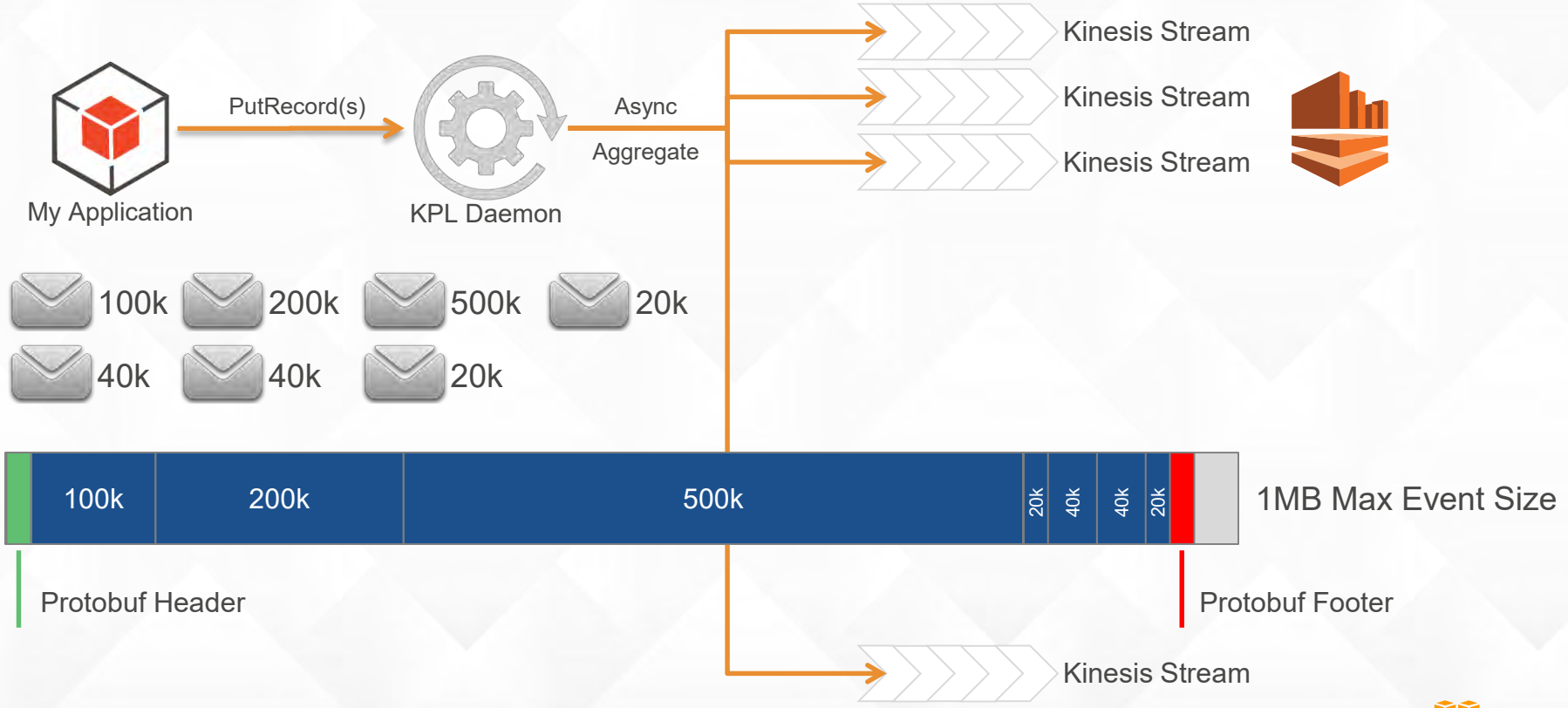


Introducing the Kinesis Producer Library

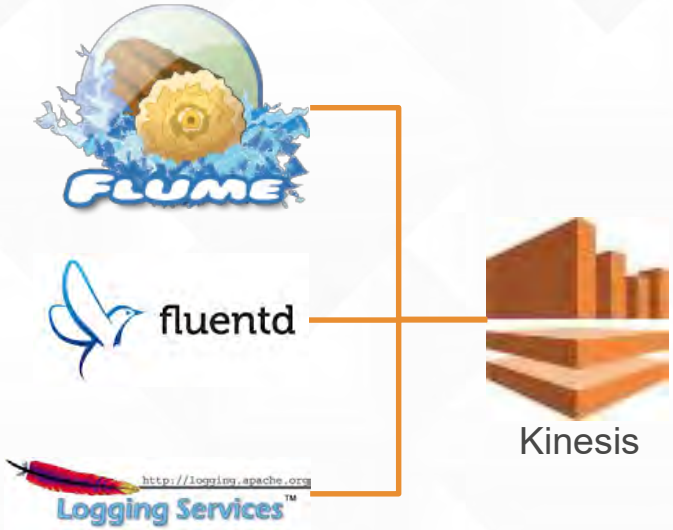


- Native Code Module to perform efficient writes to Multiple Kinesis Streams
- C++/Boost
- Asynchronous Execution
- Configurable Aggregation of Events

KPL Aggregation



Kinesis Ecosystem - Ingest



Apache Flume

Source & Sink

https://github.com/pdeyhim/flume-kinesis

FluentD

Dynamic Partitioning Support

https://github.com/awslabs/aws-fluent-plugin-kinesis

Log4J & Log4Net

Included in Kinesis Samples

amazon web services

Best Practices for Partition Key

- Random will give even distribution
- If events should be processed together, choose a relevant high cardinality partition key and monitor shard distribution
- If partial order is important use sequence number

Getting Data Out

amazon web services

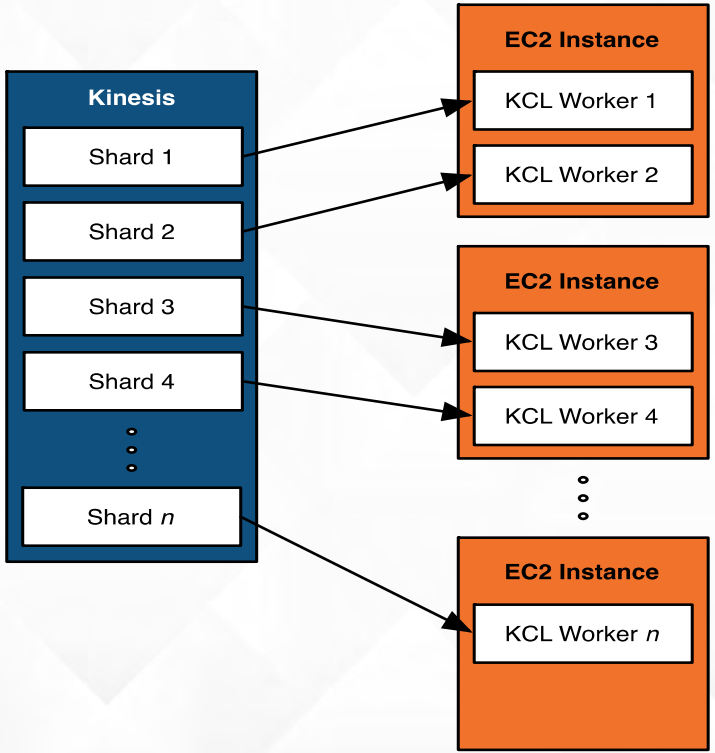
Kinesis Client Library

- KCL Libraries available for Java, Ruby, Node, Go, and a Multi-Lang Implementation with Native Python support
- All State Management in Dynamo DB

Consuming Data - Kinesis Enabled Applications

Client library for fault-tolerant, at least-once, real-time processing

- Kinesis Client Library (KCL) **s**implifies reading from the stream by abstracting your code from individual shards
- Automatically starts a Worker Thread for each Shard
- Increases and decreases Thread count as number of Shards changes
- Uses checkpoints to keep track of a Thread's location in the stream
- Restarts Threads & Workers if they fail



Kinesis Connectors

Analytics Tooling Integration (github.com/awslabs/amazon-kinesis-connectors)

S3

- Batch Write Files for Archive into S3
- Sequence Based File Naming

Redshift

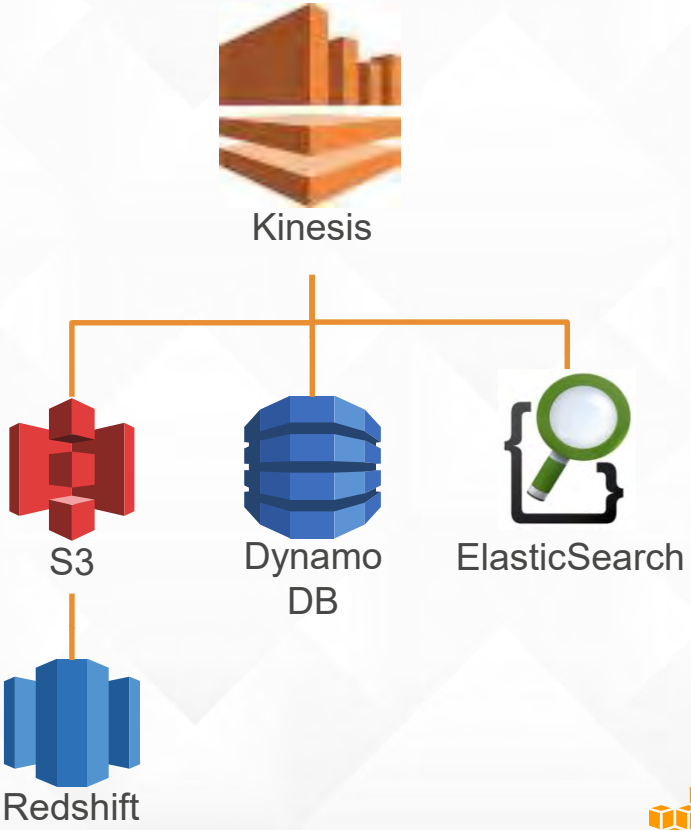
- Once Written to S3, Load to Redshift
- Manifest Support
- User Defined Transformers

DynamoDB

- BatchPut Append to Table
- User Defined Transformers

ElasticSearch

- Automatically index Stream



Connectors Architecture

Kinesis

ITransformer

- Defines the transformation of records from the Amazon Kinesis stream in order to suit the user-defined data model

IFilter

- Excludes irrelevant records from the processing.

IBuffer

- Buffers the set of records to be processed by specifying size limit (# of records)& total byte count

IEmitter

- Makes client calls to other AWS services and persists the records stored in the buffer.

S3

DynamoDB

Redshift

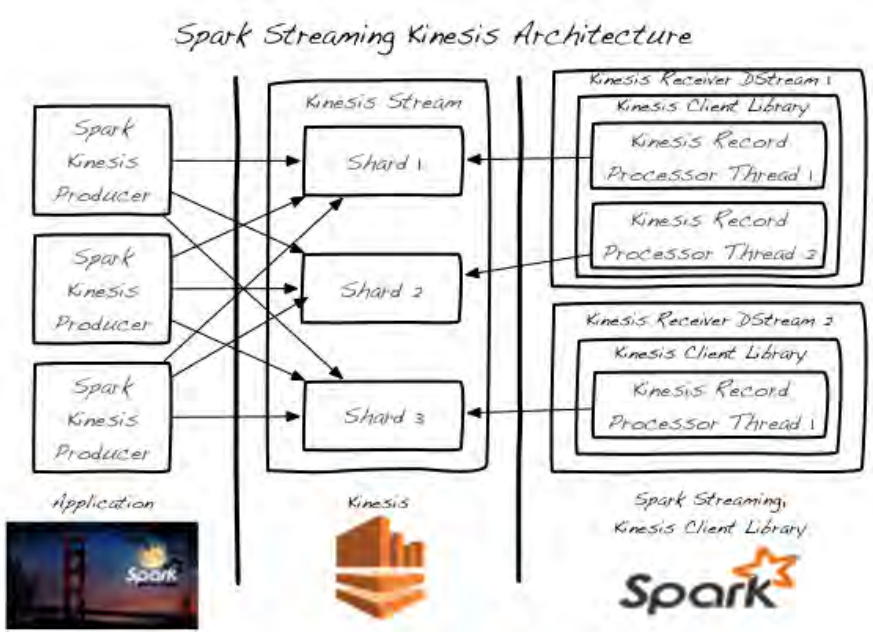
amazon web services

Kinesis Ecosystem - Storm

Apache Storm

- Kinesis Spout
- Automatic Checkpointing with Zookeeper
- https://github.com/awslabs/kinesis-storm-spout

Kinesis Ecosystem - Spark



Apache Spark

- DStream Receiver runs KCL
- One DStream per Shard
- Checkpointed via KCL

Spark Natively Available on EMR

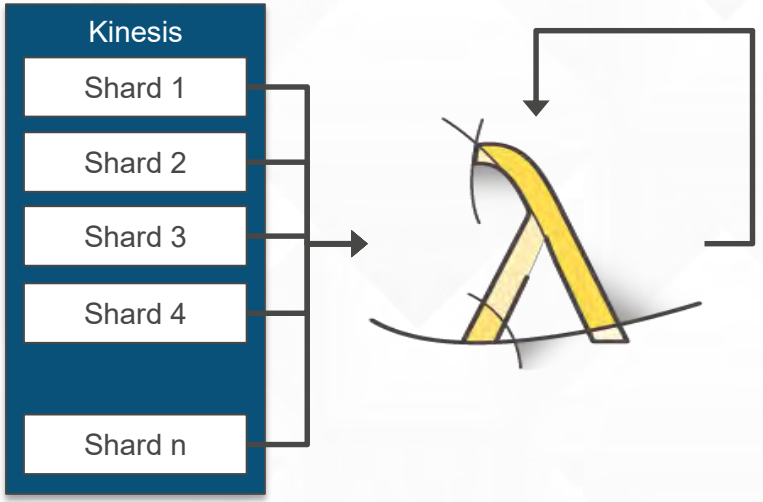
- EMRFS overlay on HDFS
- AMI 3.8.0
- https://aws.amazon.com/elasticmapreduce/details/spark

amazon web services

Consuming Data - AWS Lambda

Distributed Event Processing Platform

- **Stateless JavaScript & Java functions run against an Event Stream**
- **AWS SDK Built In**
- **Configure RAM and Execution Timeout**
- **Functions automatically invoked against a Shard**
- **Community libraries for Python & Go**
- **Access to underlying filesystem for read/write**
- **Call other Lambda Functions**



Why Kinesis? Durability

Regional Service
Synchronous Writes to Multiple AZ's
Extremely High Durability

May be in-memory for Performance
Requirement to understand Disk Sync Semantics
User Managed Replication
Replication Lag -> RPO

Why Kinesis? Performance

Perform continual processing on streaming big data. Processing latencies fall to a <1 second, compared with the minutes or hours associated with batch processing

Processing latencies < 1 second
Based on CPU & Disk Performance
Cluster Interruption -> Processing Outage

Why Kinesis? Availability

Regional Service
Synchronous Writes to Multiple AZ's
Extremely High Durability
AZ, Networking, & Chain Server Issues Transparent to Producers & Consumers

Many Depend on a CP Database
Lost Quorum can result in failure/inconsistency of the cluster
Highest Availability is determined by Availability of Cross-AZ Links or Availability of an AZ

Why Kinesis? Operations

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

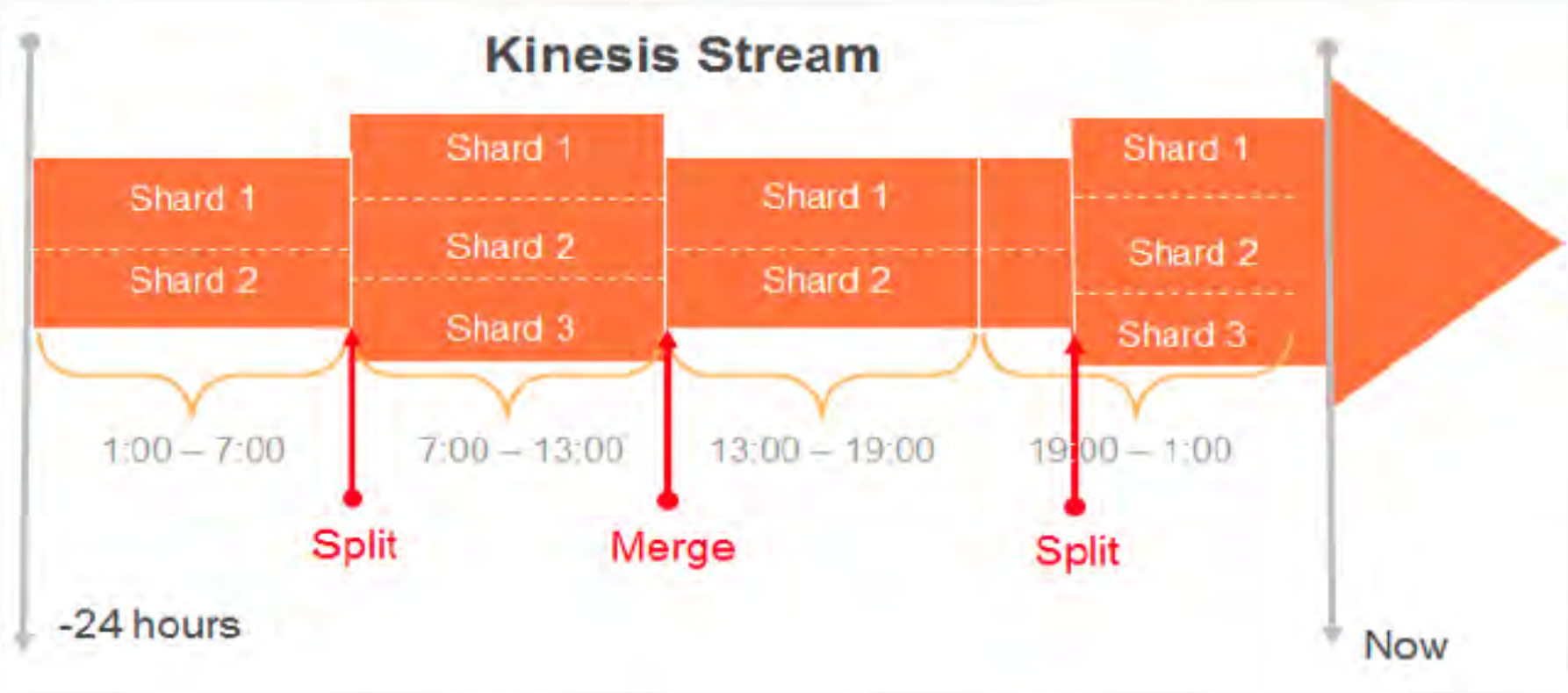
Build Instances
Install Software
Operate Cluster
Manage Disk Space
Manage Replication
Migrate to new Stream on Scale Up

Why Kinesis? Elasticity

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

Fixed Partition Count up Front
Maximum Performance ~ 1 Partition/Core | Machine
Convert from 1 Stream to Another to Scale
Application Reconfiguration

Scaling Streams



https://github.com/awslabs/amazon-kinesis-scaling-utils

amazon web services

Why Kinesis? 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.
Scale Up/Down Dynamically

$.015/Hour/1MB

Run your Own EC2 Instances
Multi-AZ Configuration for increased Durability
Utilise Instance AutoScaling on Worker Lag from HEAD with Custom Metrics

amazon web services

Why Kinesis? Cost

- *Price Dropped on 2nd June 2015, Restructured to support KPL*
- *Old Pricing: $.028 / 1M Records PUT*
- *New Pricing: $.014/1M 25KB "Payload Units"*

Scenario: 50,000 Events / Second, 512B / Event = 24.4 MB/Second

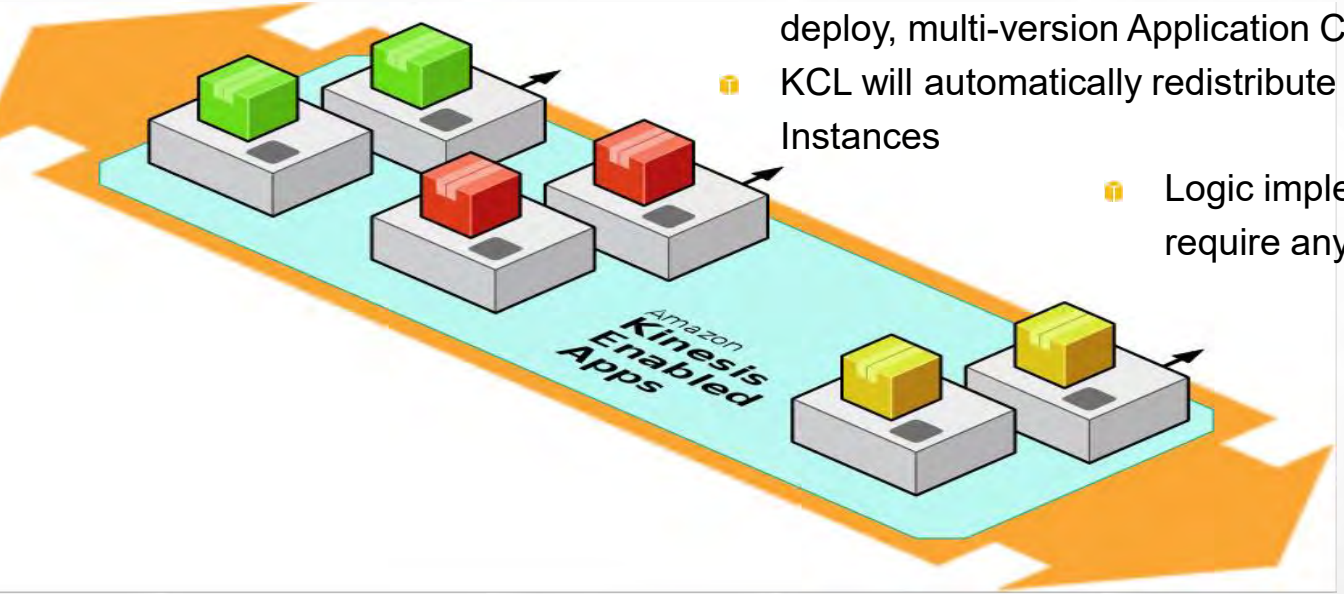
	Old Pricing		New Pricing + KPL	
	Units	**Cost**	**Units**	**Cost**
Shards	50	$558	25	$279
PutRecords	4,320M Records	$120.96	2,648M Payload Units	$37.50
		$678.96		$316.50

amazon web services


Everything
Fails,
All the Time

Kinesis – Consumer Application Best Practices

- Tolerate Failure of: Threads – Consider Data Serialisation issues and Lease Stealing; Hardware – AutoScaling may add nodes as needed
- Scale Consumers up and down as the number of Shards increase or decrease
- Don't store data in memory in the workers. Use an elastic data store such as Dynamo DB for State

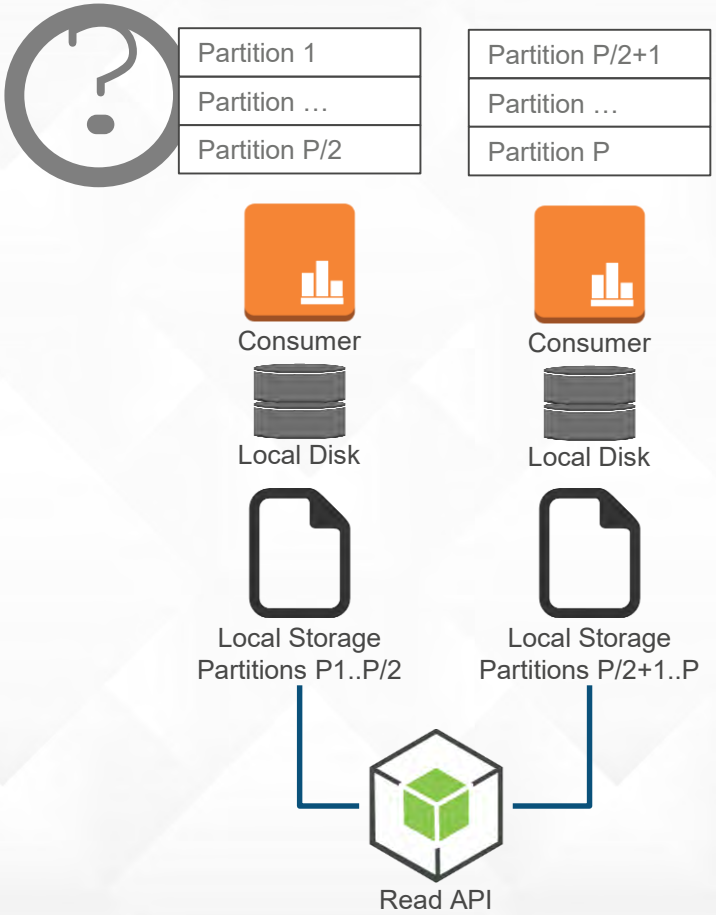


- Elastic Beanstalk provides all Best Practices in a simple to deploy, multi-version Application Container
- KCL will automatically redistribute Workers to use new Instances
- Logic implemented in Lambda doesn't require any Servers at all!

Managing Application State

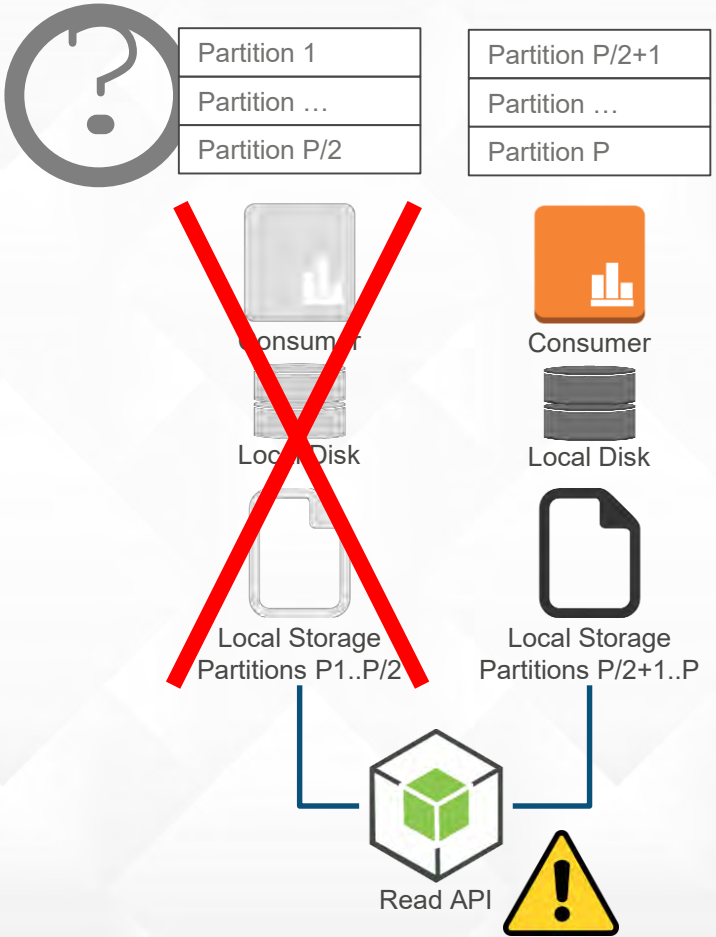
amazon web services

Consumer Local State Anti-Pattern



- Consumer binds to a configured number of Partitions
- Consumer stores the ‘state’ of a data structure, as defined by the event stream, on local storage
- Read API can access that local storage as a ‘shard’ of the overall database

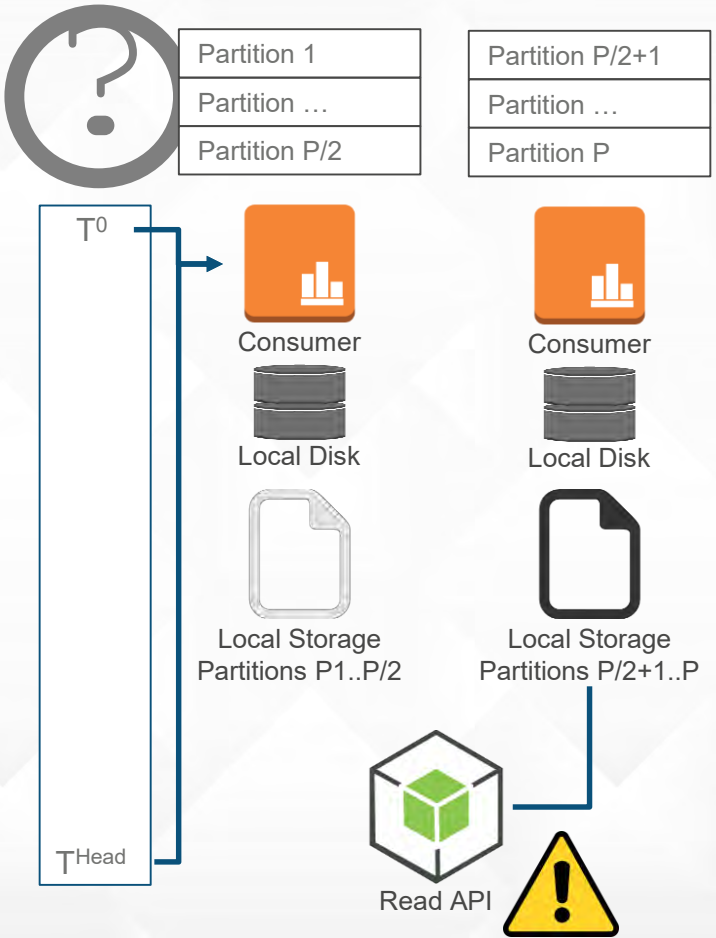
Consumer Local State Anti-Pattern



But what happens when an instance fails?

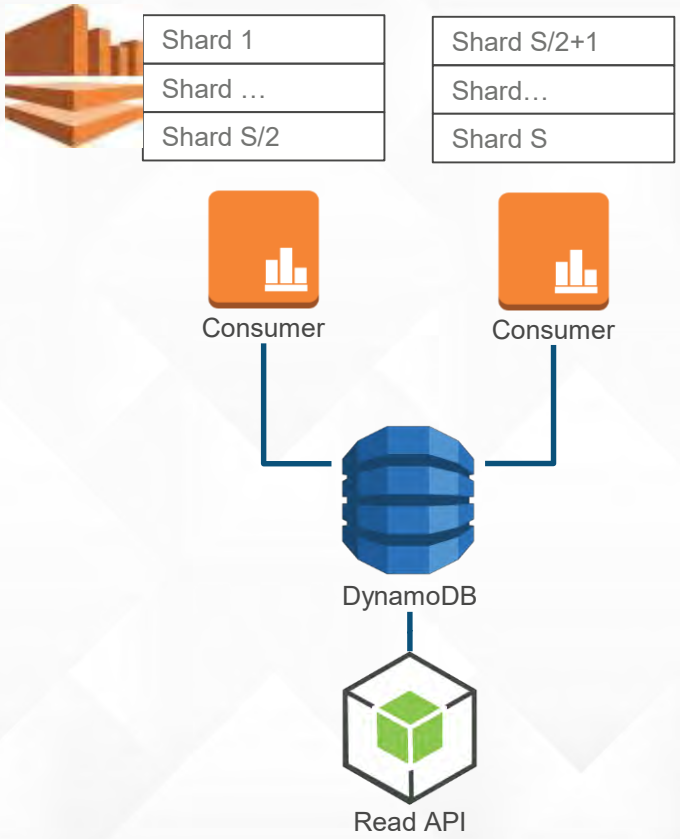
amazon web services

Consumer Local State Anti-Pattern



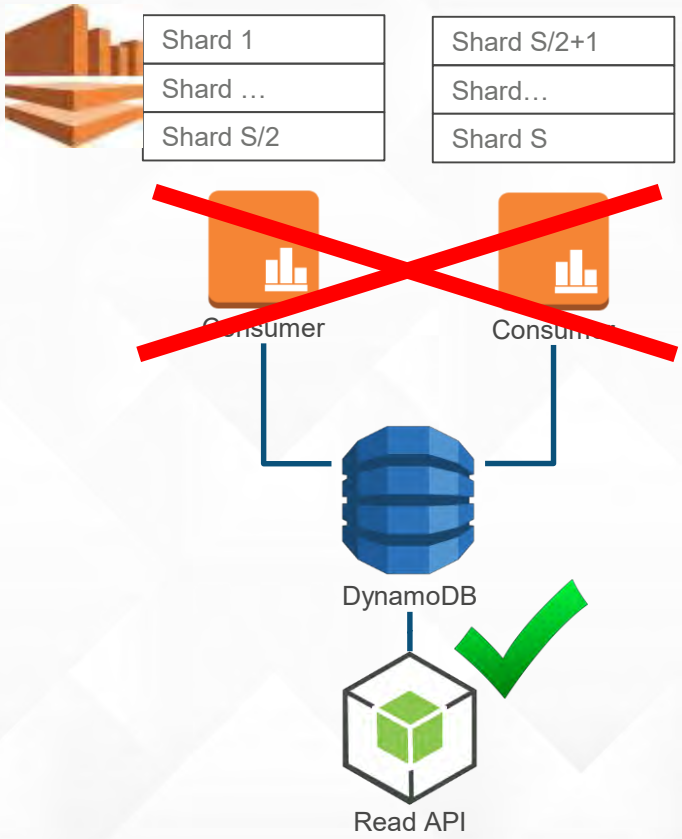
- A new consumer process starts up for the required Partitions
- Consumer must read from the beginning of the Stream to rebuild local storage
- Complex, error prone, user constructed software
- Long Startup Time

External Highly Available State – Best Practice



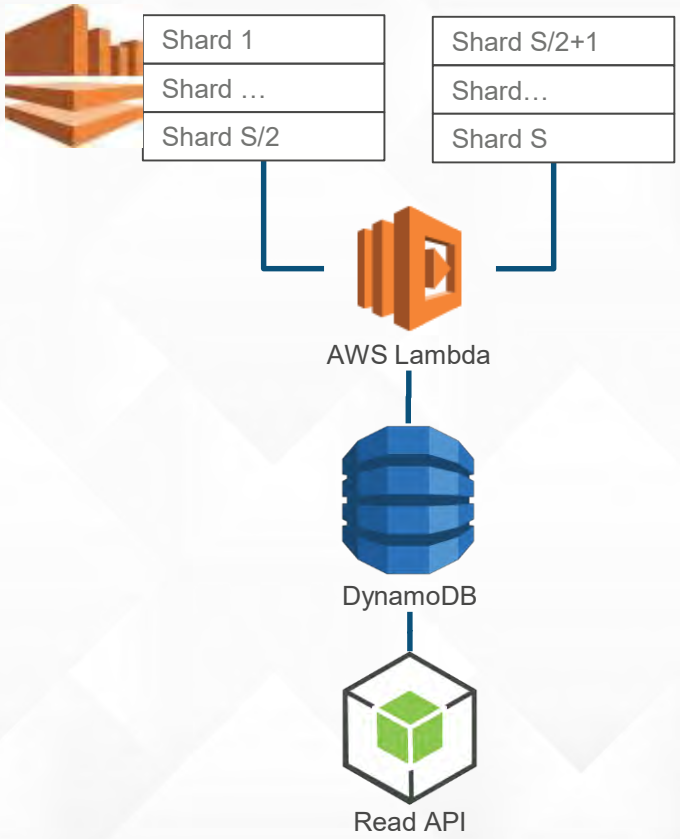
- Consumer binds to a even number of Shards based on number of Consumers
- Consumer stores the ‘state’ in Dynamo DB
- Dynamo DB is Highly Available, Elastic & Durable
- Read API can access Dynamo DB

External Highly Available State – Best Practice



- Consumer binds to a even number of Shards based on number of Consumers
- Consumer stores the ‘state’ in Dynamo DB
- Dynamo DB is Highly Available, Elastic & Durable
- Read API can access Dynamo DB

External Highly Available State – Best Practice



- Consumer binds to a even number of Shards based on number of Consumers
- Consumer stores the ‘state’ in Dynamo DB
- Dynamo DB is Highly Available, Elastic & Durable
- Read API can access Dynamo DB

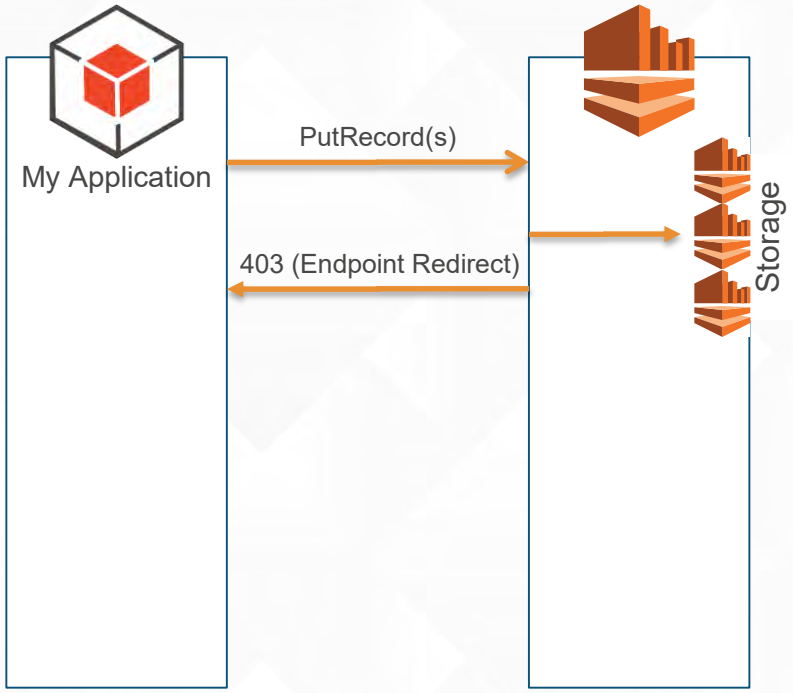
Idempotency

Property of a system whereby the repeated application of a function on a single input results in the same end state of the system

…

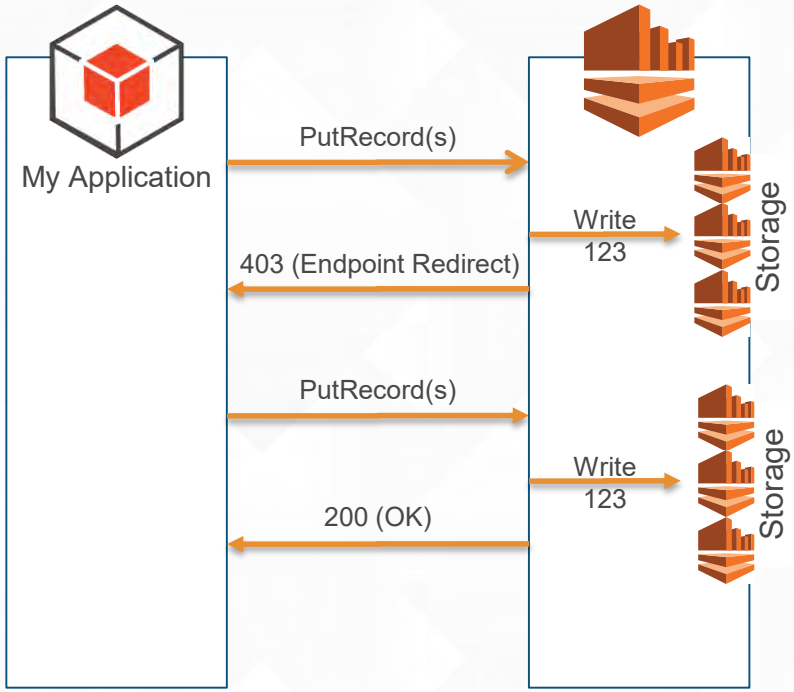
Exactly Once Processing

Idempotency – Writing Data



- The Kinesis SDK & KPL may retry PUT in certain circumstances
- Kinesis Record acknowledged with a Sequence Number **is** durable to Multiple Availability Zones…
- But there could be a duplicate entry

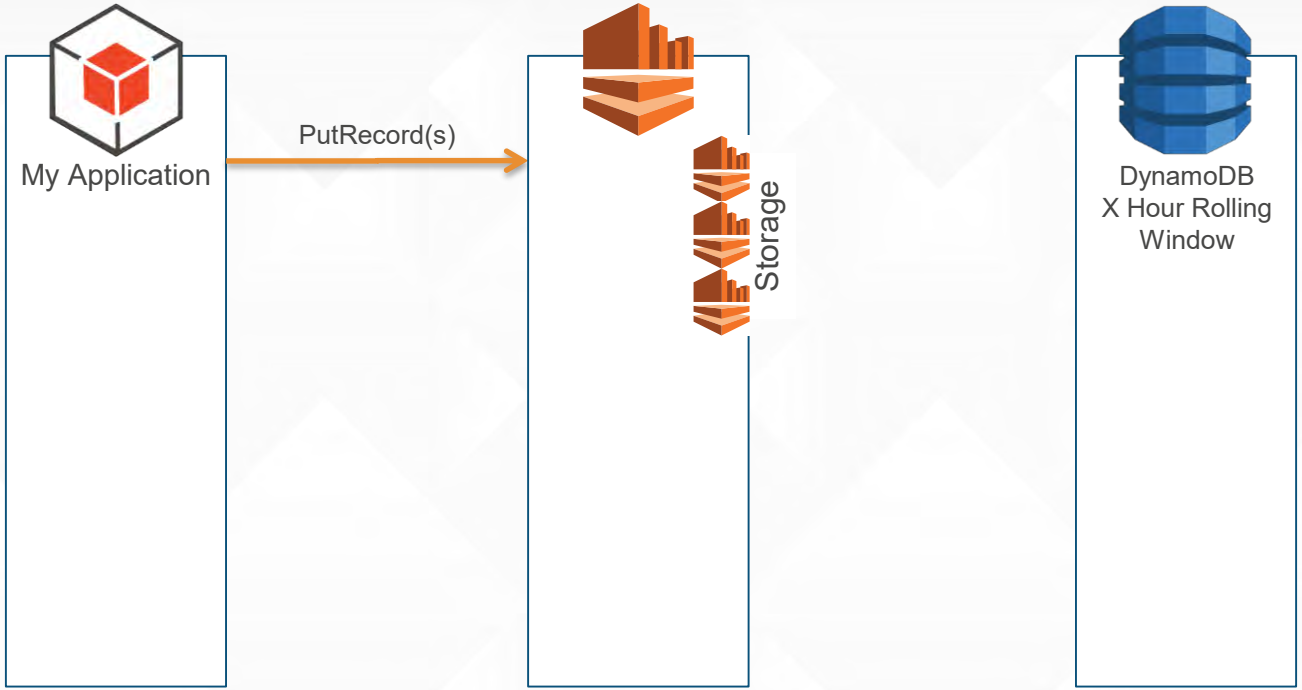
Idempotency – Writing Data



- The Kinesis SDK & KPL may retry PUT in certain circumstances
- Kinesis Record acknowledged with a Sequence Number **is** durable to Multiple Availability Zones…
- But there could be a duplicate entry

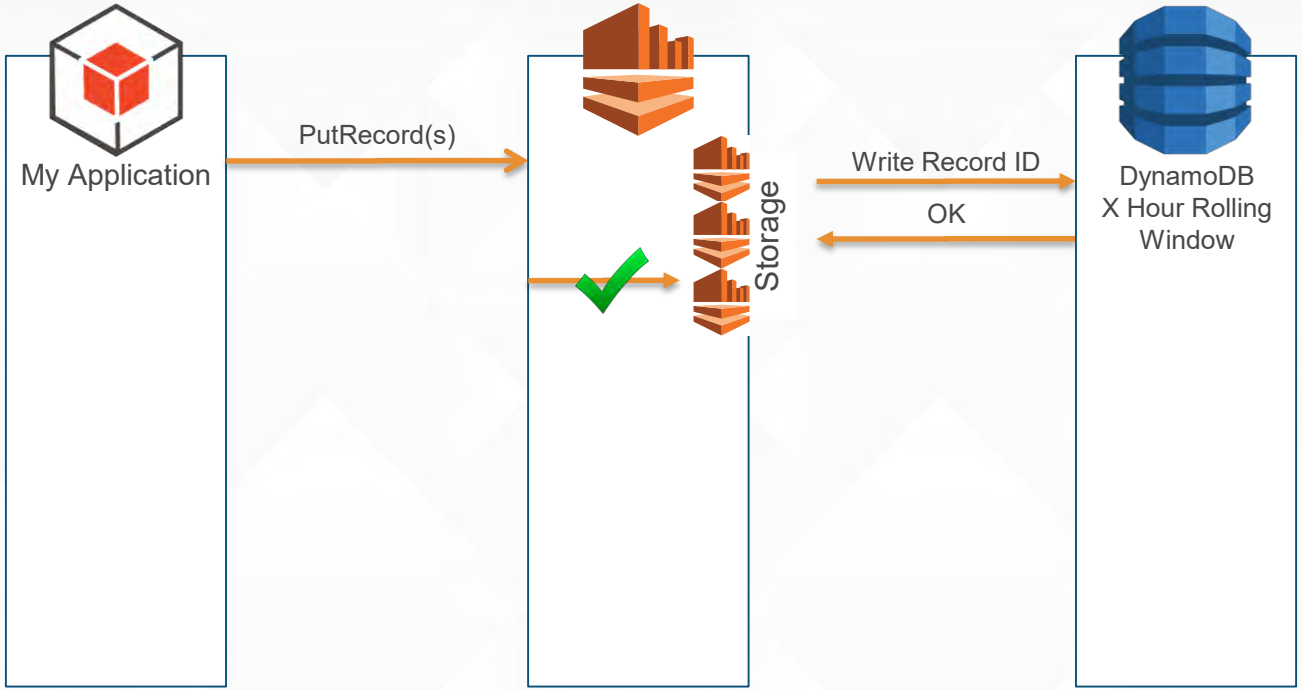
Coming Soon…

Idempotency – Rolling Idempotency Check



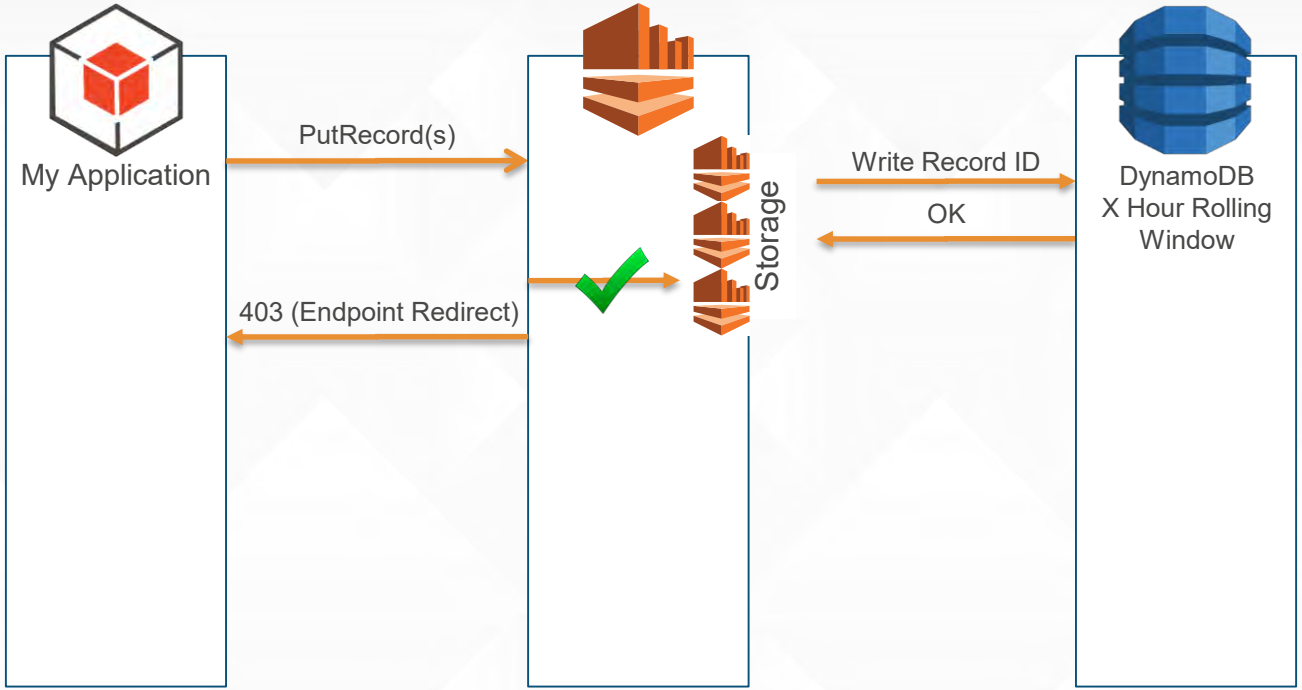
- Kinesis will manage a rolling time window of Record ID's in Dynamo DB
- Record ID's are User Based
- Duplicates in storage tier will be acknowledged as Successful

Idempotency – Rolling Idempotency Check



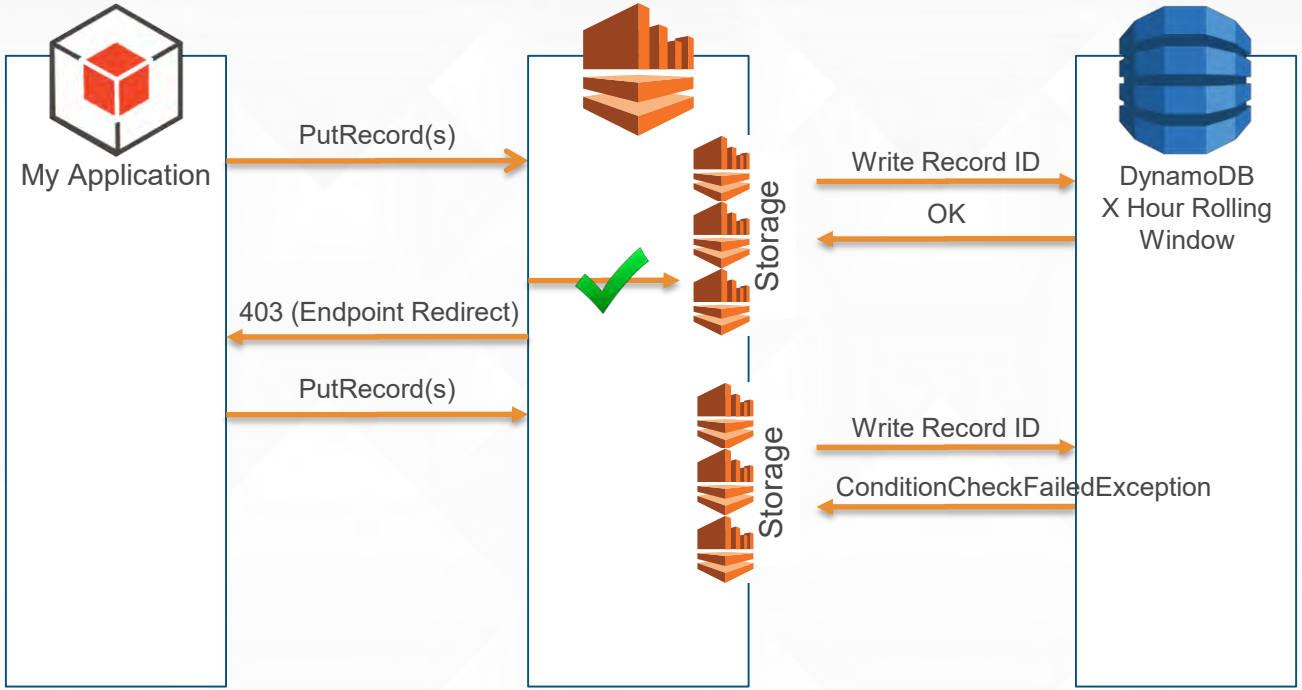
- Kinesis will manage a rolling time window of Record ID's in Dynamo DB
- Record ID's are User Based
- Duplicates in storage tier will be acknowledged as Successful

Idempotency – Rolling Idempotency Check



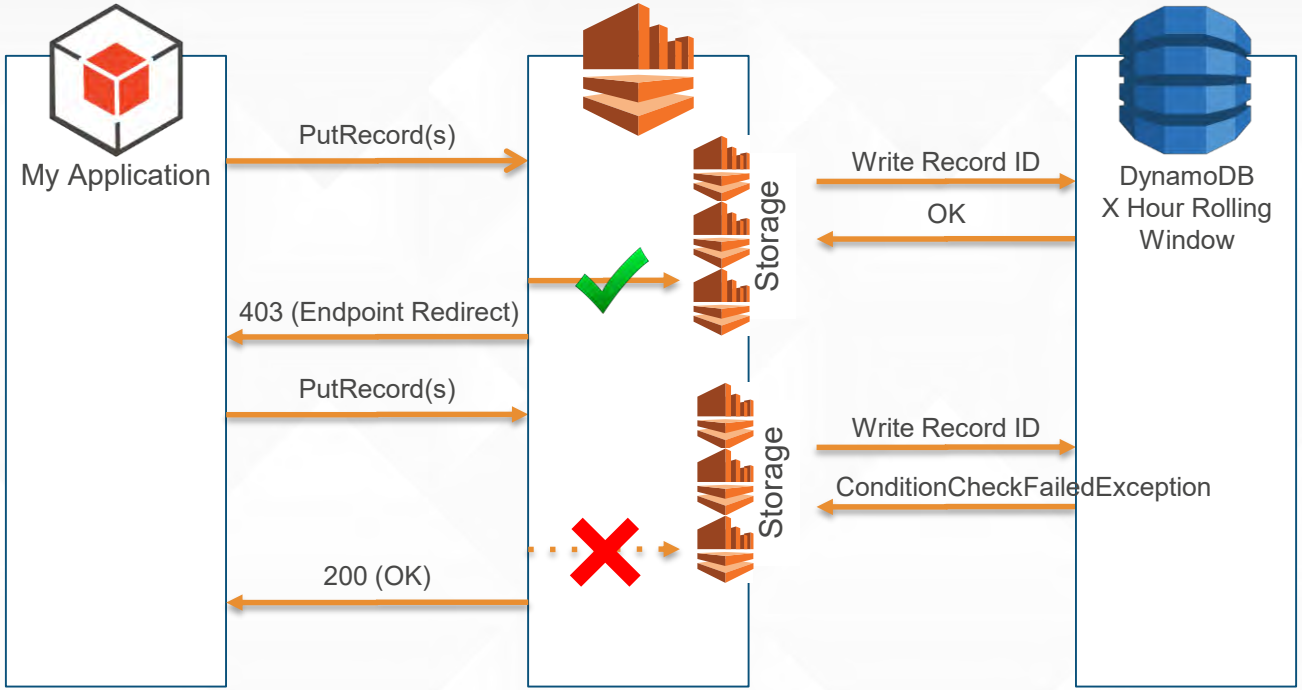
- Kinesis will manage a rolling time window of Record ID's in Dynamo DB
- Record ID's are User Based
- Duplicates in storage tier will be acknowledged as Successful

Idempotency – Rolling Idempotency Check



- Kinesis will manage a rolling time window of Record ID's in Dynamo DB
- Record ID's are User Based
- Duplicates in storage tier will be acknowledged as Successful

Idempotency – Rolling Idempotency Check



- Kinesis will manage a rolling time window of Record ID's in Dynamo DB
- Record ID's are User Based
- Duplicates in storage tier will be acknowledged as Successful

In Short…

Easy Administration

Real-time
Performance.
High Durability

High Throughput.
Elastic

S3, Redshift, &
DynamoDB
Integration

Large Ecosystem

Low Cost

amazon web services

Thank You

amazon web services