

# Disclaimer: These slides are copyrighted and strictly for personal use only

- This document is reserved for people enrolled into the [AWS Certified Big Data Specialty course by Stephane Maarek and Frank Kane.](#)
- **Please do not share this document**, it is intended for personal use and exam preparation only, thank you.
- If you've obtained these slides for free on a website that is not the course's website, please reach out to [piracy@datacumulus.com](mailto:piracy@datacumulus.com). Thanks!
- **Best of luck for the exam and happy learning!**

# AWS Certified Data Analytics Specialty Course

## DAS-C01

# Welcome! We're starting in 5 minutes



- We're going to prepare for the Data Analytics Specialty exam – DAS-C01
- It's a challenging certification, so this course will be long and interesting
- Recommended to have previous AWS knowledge (EC2, networking...)
- Preferred to have some data / analytics background
- We will cover all the AWS Data Analytics services related to the exam
- Take your time, it's not a race!

# My certification: 94%

Overall Score: 94%

Topic Level Scoring:

- 1.0 Collection: 100%
- 2.0 Storage: 100%
- 3.0 Processing: 100%
- 4.0 Analysis: 87%
- 5.0 Visualization: 100%
- 6.0 Data Security: 80%



BDS-C00



DAS-C01

# About me

- I'm Stephane!
- Worked as in IT consultant and AWS Big Data Architect, Developer & SysOps
- Worked with AWS many years: built websites, apps, streaming platforms
- Veteran Instructor on AWS (Certifications, CloudFormation, Lambda, EC2...)
- You can find me on
  - LinkedIn: <https://www.linkedin.com/in/stephanemaarek>
  - Instagram: <https://www.instagram.com/stephanemaarek/>
  - Twitter: <https://twitter.com/stephanemaarek>
  - Medium: <https://medium.com/@stephane.maarek>
  - GitHub: <https://github.com/simplesteph>



★ 4.6 Instructor Rating  
💬 83,745 Reviews  
👤 282,446 Students  
▶ 29 Courses

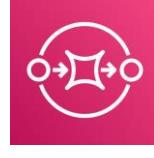
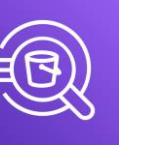
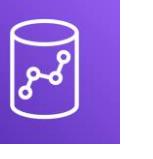
# About me

- I'm Frank!
- 9 years at Amazon as a Sr. Software Engineer and Sr. Manager
- Focused on machine learning / recommender systems in big data environment
- Owner of Sundog Education – Big Data & ML
- You can find me on
  - LinkedIn: <https://www.linkedin.com/in/fkane/>
  - Twitter: <http://www.twitter.com/SundogEducation>
  - Facebook: <https://www.facebook.com/SundogEdu>

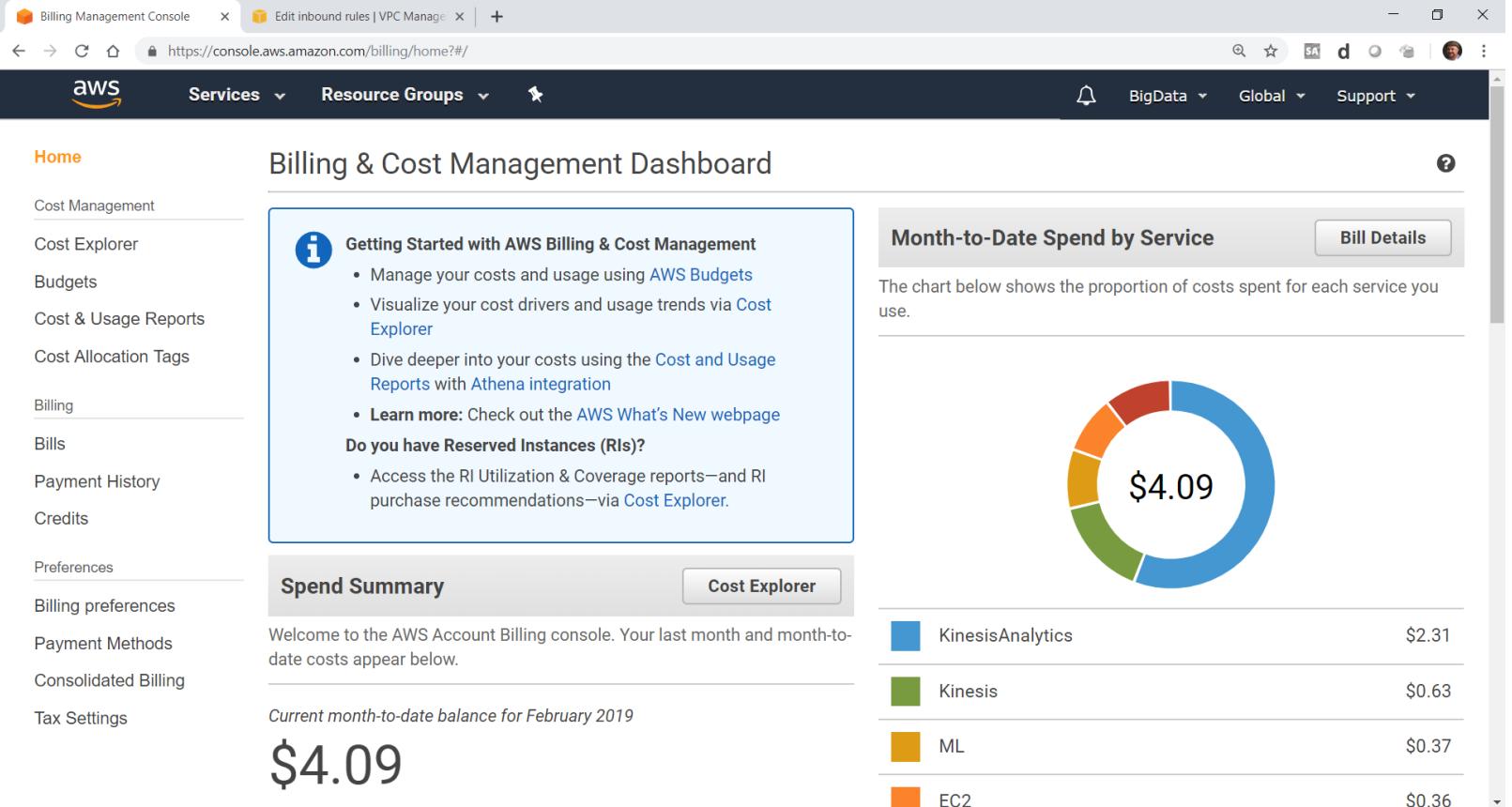


★ 4.5 Instructor Rating  
💬 48,404 Reviews  
👤 238,457 Students  
🌐 15 Courses

# Services we'll learn

COLLECTION	STORAGE	PROCESSING	ANALYSIS	VISUALIZATION	SECURITY
					
Amazon Kinesis	AWS IoT Core	S3 + Glacier	AWS Lambda	Amazon ML	Elasticsearch
					
AWS Snowball	Amazon SQS	DynamoDB	AWS Glue	Amazon SageMaker	Amazon Athena
					
Amazon DMS	AWS Direct Connect	ElastiCache	Amazon EMR	AWS Data Pipeline	Amazon Redshift
					
					AWS CloudHSM

# Course Cost



The screenshot shows the AWS Billing & Cost Management Dashboard. On the left, there's a sidebar with links for Cost Management, Cost Explorer, Budgets, Cost & Usage Reports, Cost Allocation Tags, Billing, Bills, Payment History, Credits, Preferences, Billing preferences, Payment Methods, Consolidated Billing, and Tax Settings. The main area has a title "Billing & Cost Management Dashboard". It features a "Getting Started with AWS Billing & Cost Management" box with a bulleted list about managing costs, visualizing trends, and learning more. Below this is a "Spend Summary" section with a "Cost Explorer" button. A large "\$4.09" is displayed prominently. To the right is a "Month-to-Date Spend by Service" chart and a corresponding table:

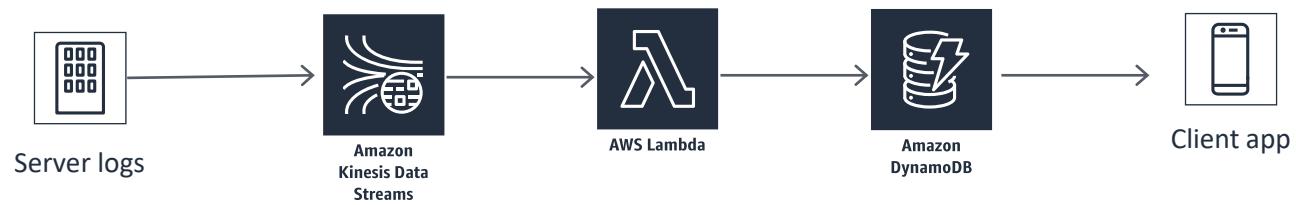
Service	Amount
KinesisAnalytics	\$2.31
Kinesis	\$0.63
ML	\$0.37
EC2	\$0.36

# Introducing our case study

# Our case study: cadabra.com



# Requirement 1: Order history app

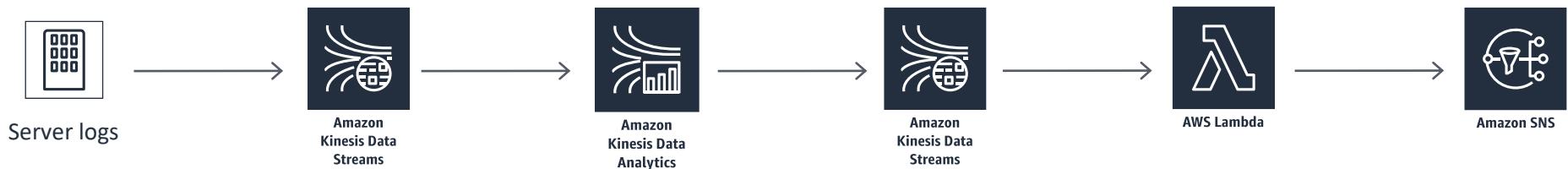


# Requirement 2: Product recommendations



# Requirement 3:

## Transaction rate alarm

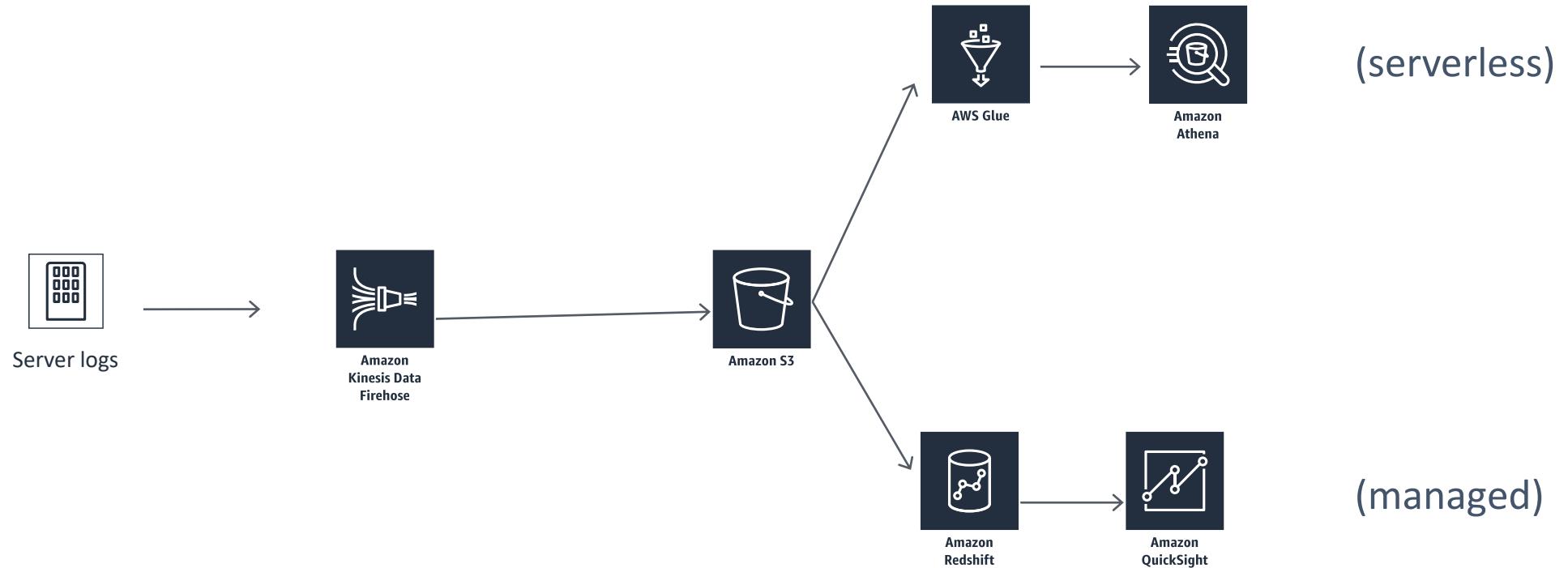


# Requirement 4:

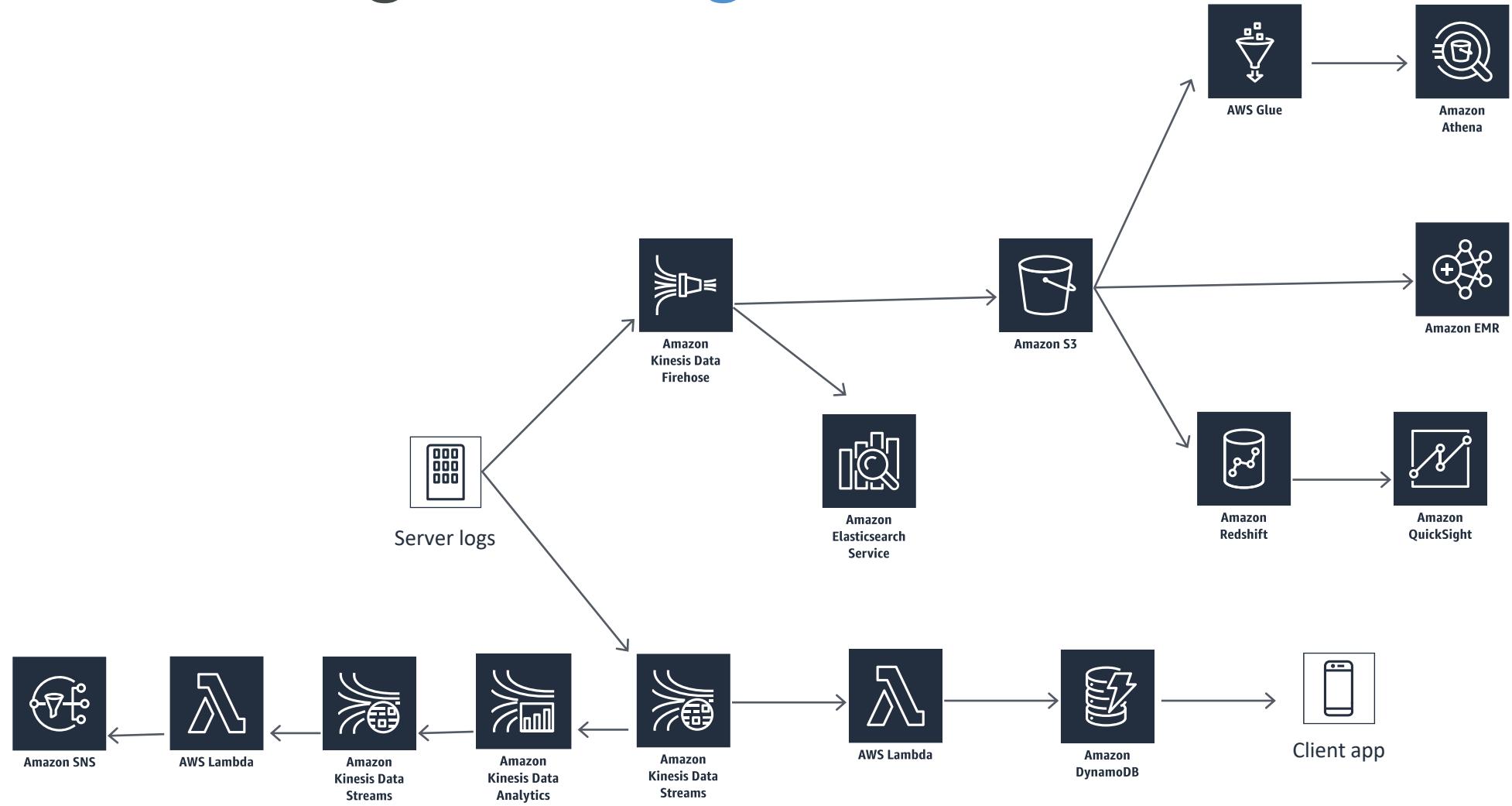
## Near-real-time log analysis



# Requirement 5: Data warehousing & visualization



# Putting it all together



# Collection

Moving data into AWS

# Collection Introduction

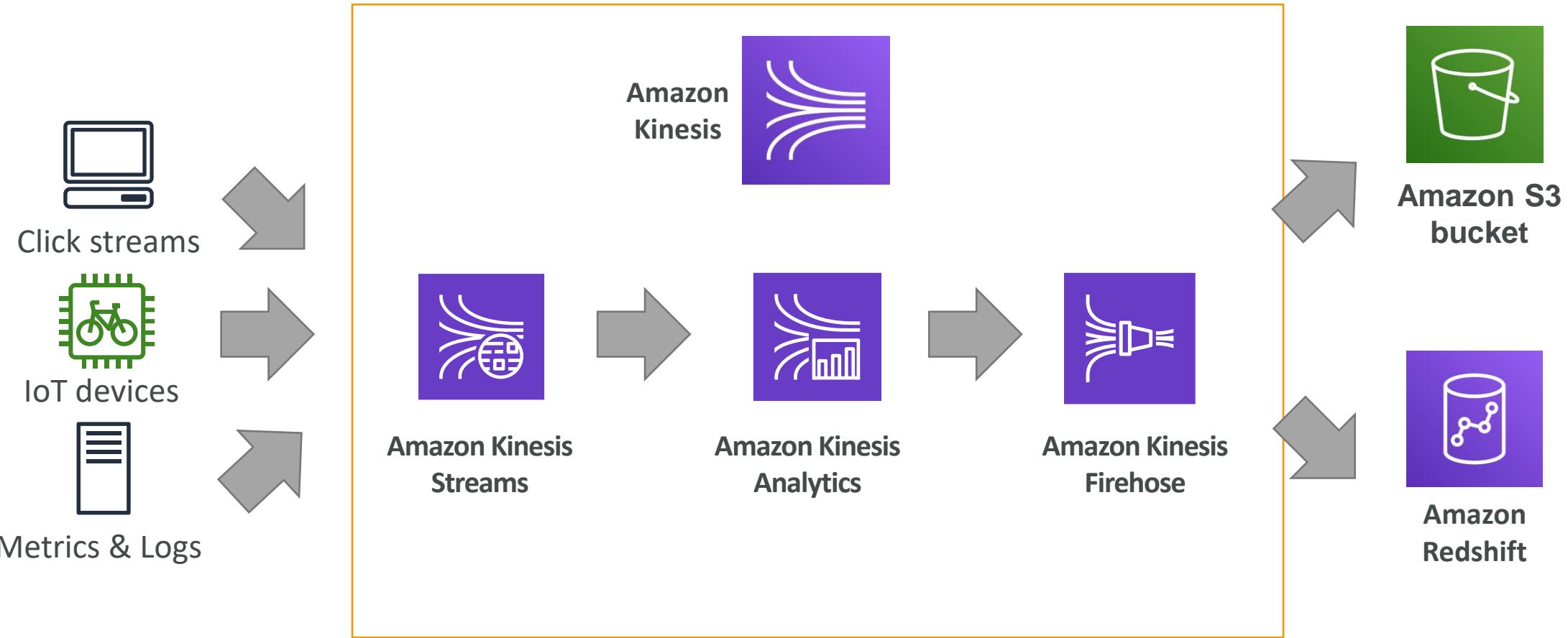
- Real Time - Immediate actions
  - Kinesis Data Streams (KDS)
  - Simple Queue Service (SQS)
  - Internet of Things (IoT)
- Near-real time - Reactive actions
  - Kinesis Data Firehose (KDF)
  - Database Migration Service (DMS)
- Batch - Historical Analysis
  - Snowball
  - Data Pipeline

# AWS Kinesis Overview



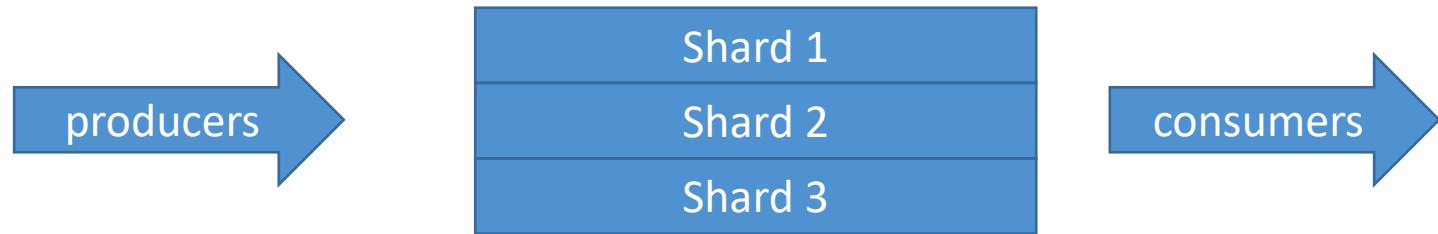
- **Kinesis** is a managed alternative to Apache Kafka
  - Great for application logs, metrics, IoT, clickstreams
  - Great for “real-time” big data
  - Great for streaming processing frameworks (Spark, NiFi, etc...)
  - Data is automatically replicated synchronously to 3 AZ
- 
- **Kinesis Streams:** low latency streaming ingest at scale
  - **Kinesis Analytics:** perform real-time analytics on streams using SQL
  - **Kinesis Firehose:** load streams into S3, Redshift, ElasticSearch & Splunk

# Kinesis



# Kinesis Streams Overview

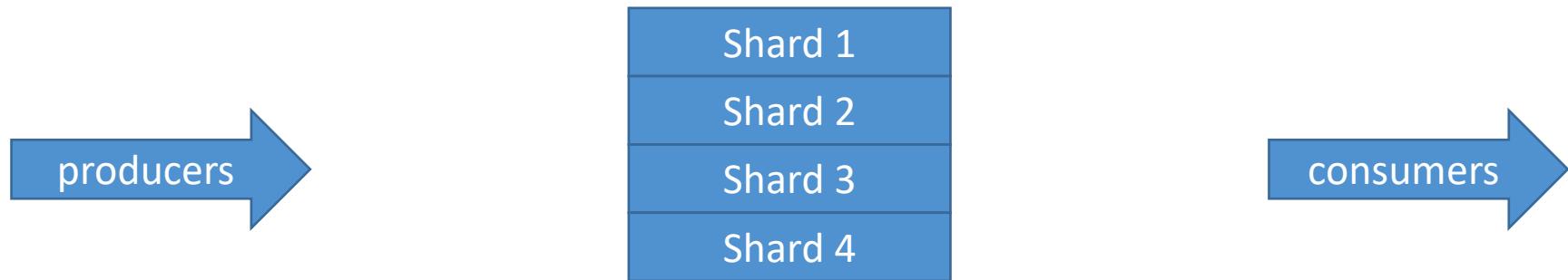
- Streams are divided in ordered Shards / Partitions



- Data retention is 24 hours by default, can go up to 7 days
- Ability to reprocess / replay data
- Multiple applications can consume the same stream
- Real-time processing with scale of throughput
- Once data is inserted in Kinesis, it can't be deleted (immutability)

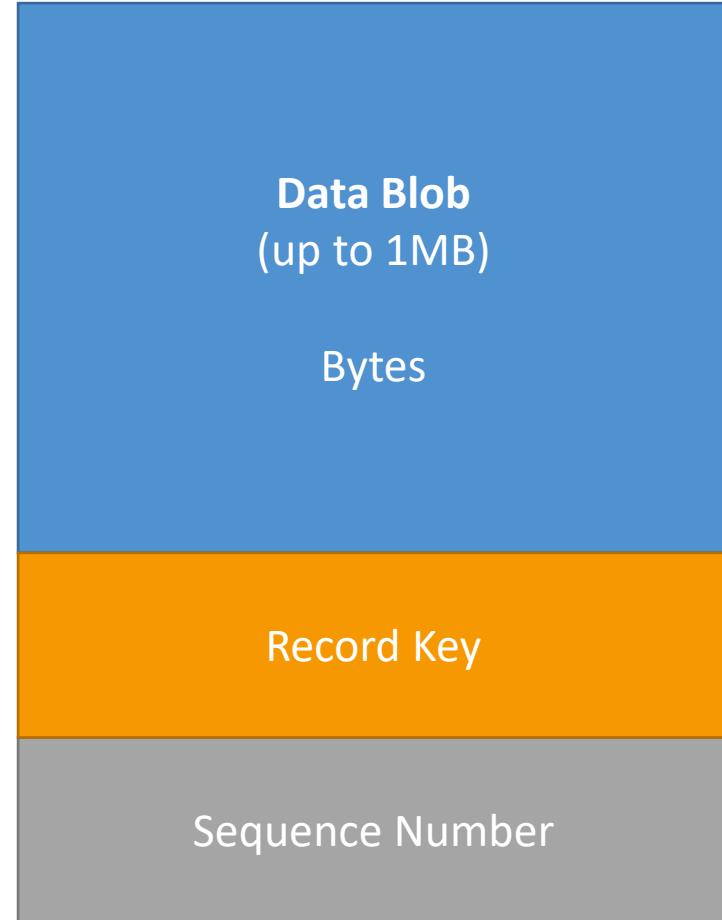
# Kinesis Streams Shards

- One stream is made of many different shards
- Billing is per shard provisioned, can have as many shards as you want
- Batching available or per message calls.
- The number of shards can evolve over time (reshard / merge)
- **Records are ordered per shard**



# Kinesis Streams Records

- Data Blob: data being sent, serialized as **bytes**. Up to 1 MB. Can represent anything
- Record Key:
  - sent alongside a record, helps to group records in Shards. Same key = Same shard.
  - Use a highly distributed key to avoid the “hot partition” problem
- Sequence number: Unique identifier for each records put in shards. Added by Kinesis after ingestion

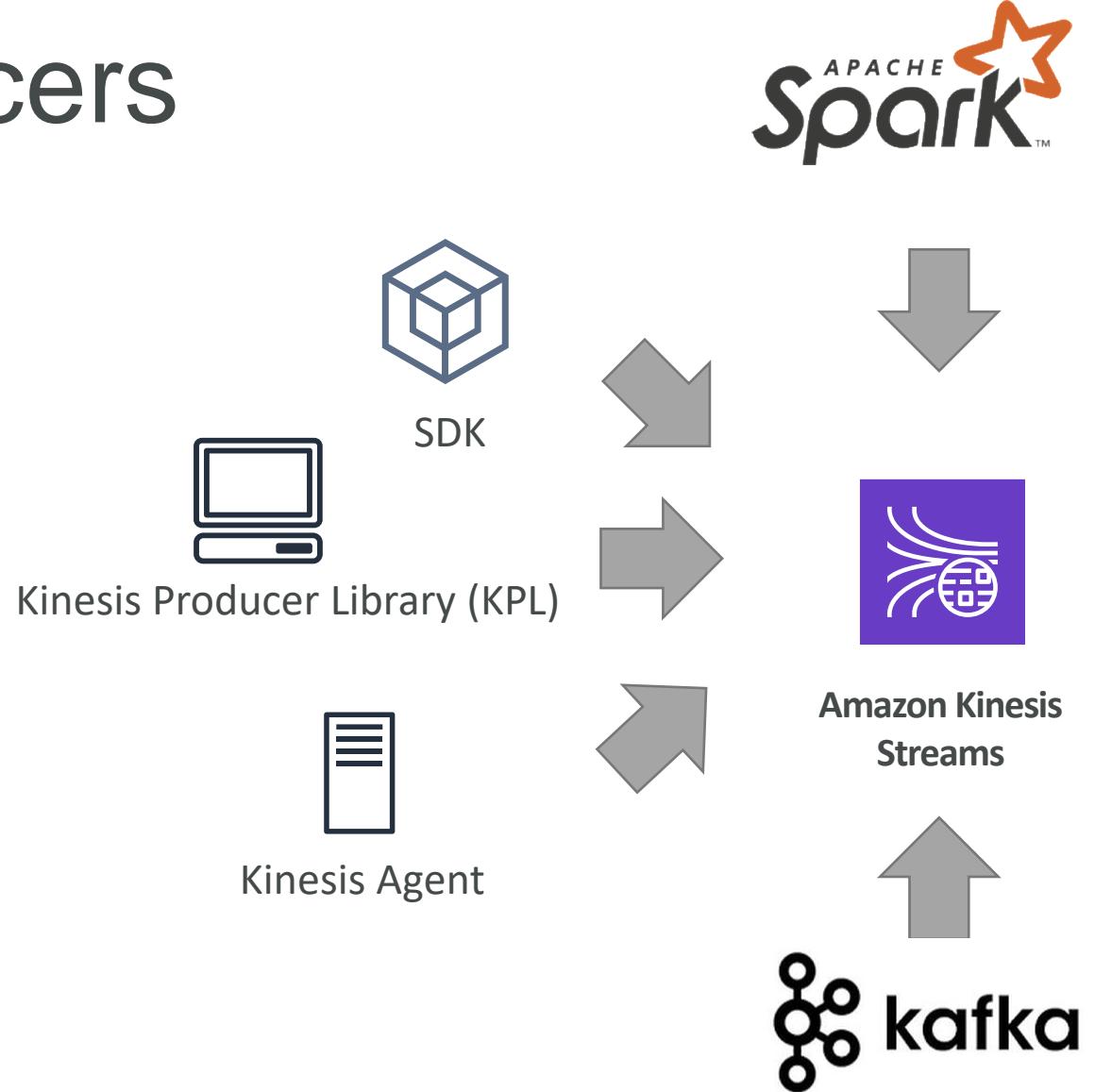


# Kinesis Data Streams Limits to know

- Producer:
  - 1MB/s or 1000 messages/s at write PER SHARD
  - “ProvisionedThroughputException” otherwise
- Consumer Classic:
  - 2MB/s at read PER SHARD across all consumers
  - 5 API calls per second PER SHARD across all consumers
- Consumer Enhanced Fan-Out:
  - 2MB/s at read PER SHARD, PER ENHANCED CONSUMER
  - No API calls needed (push model)
- Data Retention:
  - 24 hours data retention by default
  - Can be extended to 7 days

# Kinesis Producers

- Kinesis SDK
- Kinesis Producer Library (KPL)
- Kinesis Agent
- 3<sup>rd</sup> party libraries:  
Spark, Log4J  
Appenders, Flume,  
Kafka Connect,  
NiFi...



# Kinesis Producer SDK - PutRecord(s)

- APIs that are used are PutRecord (one) and PutRecords (many records)
- **PutRecords** uses batching and increases throughput => less HTTP requests
- **ProvisionedThroughputExceeded** if we go over the limits
- + AWS Mobile SDK: Android, iOS, etc...
- Use case: low throughput, higher latency, simple API, AWS Lambda
- Managed AWS sources for Kinesis Data Streams:
  - CloudWatch Logs
  - AWS IoT
  - Kinesis Data Analytics

# AWS Kinesis API – Exceptions

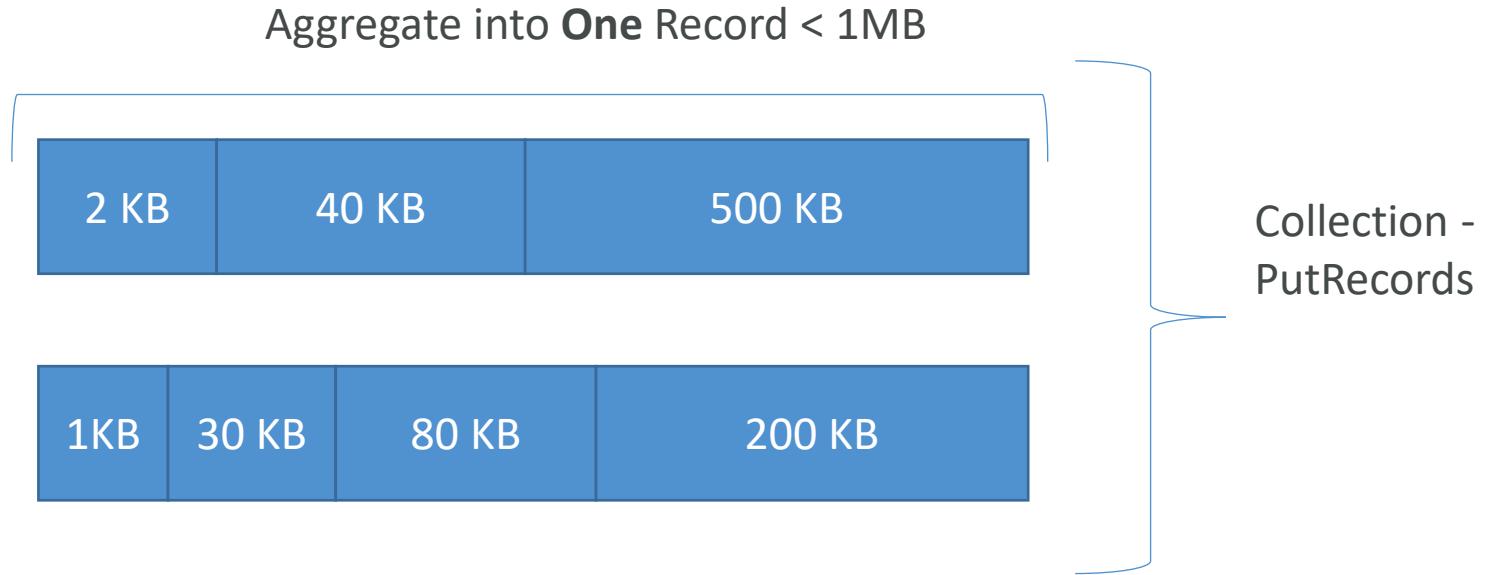
- ProvisionedThroughputExceeded Exceptions
  - Happens when sending more data (exceeding MB/s or TPS for any shard)
  - Make sure you don't have a hot shard (such as your partition key is bad and too much data goes to that partition)
- Solution:
  - Retries with backoff
  - Increase shards (scaling)
  - Ensure your partition key is a good one

# Kinesis Producer Library (KPL)

- Easy to use and highly configurable C++ / Java library
- Used for building high performance, long-running producers
- Automated and configurable **retry** mechanism
- **Synchronous or Asynchronous API** (better performance for async)
- Submits metrics to CloudWatch for monitoring
- **Batching** (both turned on by default) – increase throughput, decrease cost:
  - **Collect** Records and Write to multiple shards in the same PutRecords API call
  - **Aggregate** – increased latency
    - Capability to store multiple records in one record (go over 1000 records per second limit)
    - Increase payload size and improve throughput (maximize 1MB/s limit)
- Compression must be implemented by the user
- KPL Records must be de-coded with KCL or special helper library

# Kinesis Producer Library (KPL)

## Batching



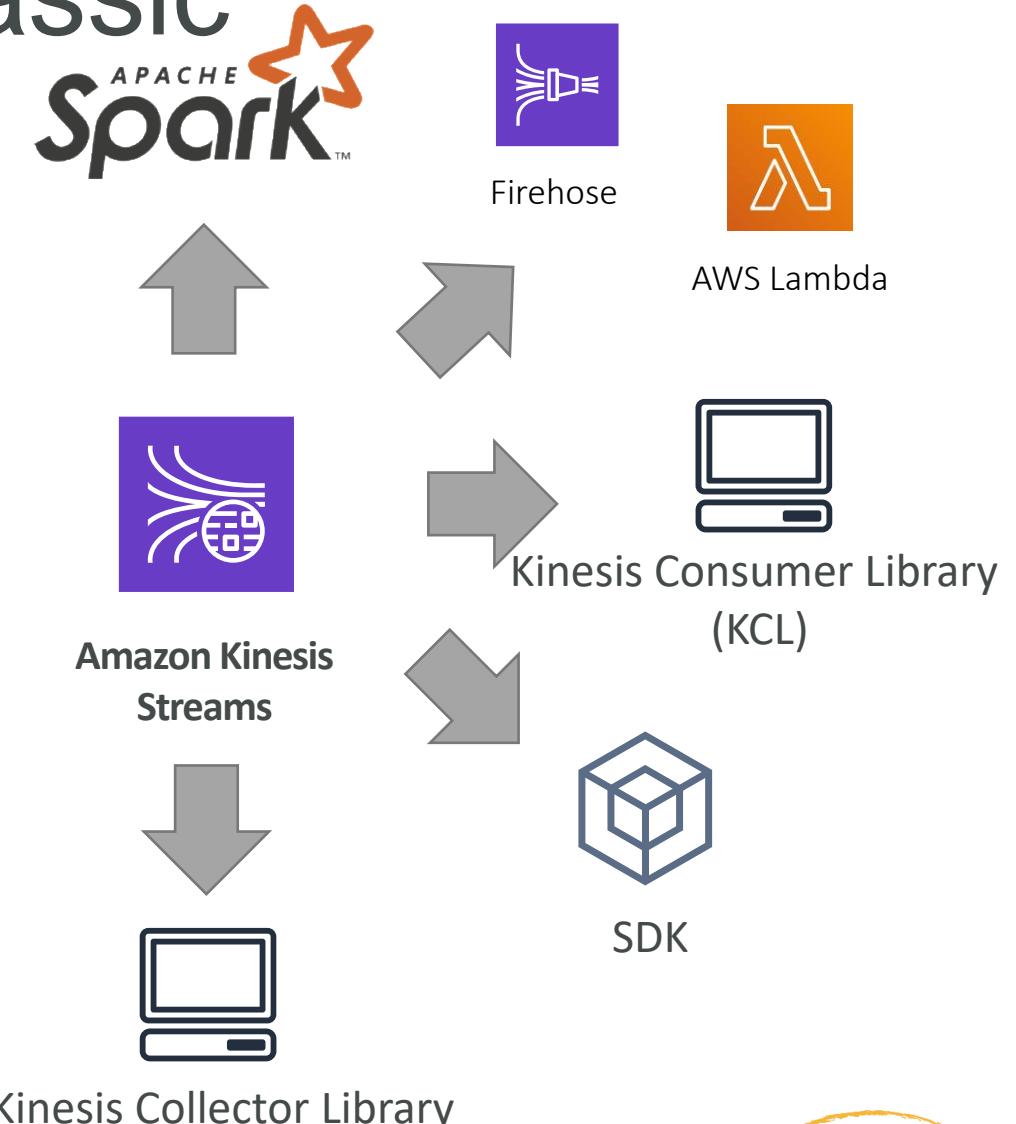
- We can influence the batching efficiency by introducing some delay with `RecordMaxBufferedTime` (default 100ms)

# Kinesis Agent

- Monitor Log files and sends them to Kinesis Data Streams
- Java-based agent, built on top of KPL
- Install in Linux-based server environments
- Features:
  - Write from multiple directories and write to multiple streams
  - Routing feature based on directory / log file
  - Pre-process data before sending to streams (single line, csv to json, log to json...)
  - The agent handles file rotation, checkpointing, and retry upon failures
  - Emits metrics to CloudWatch for monitoring

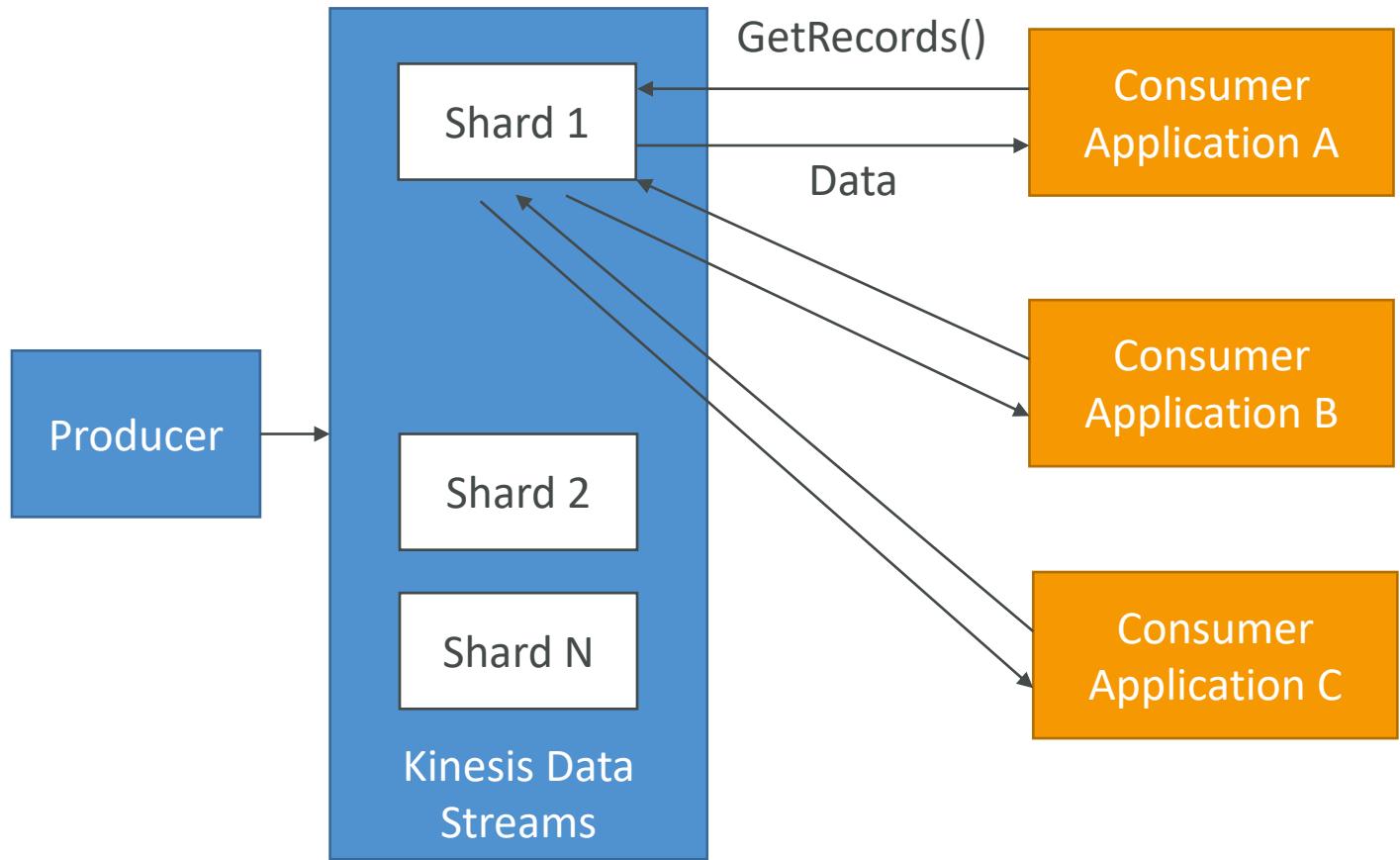
# Kinesis Consumers - Classic

- Kinesis SDK
- Kinesis Client Library (KCL)
- Kinesis Connector Library
- 3<sup>rd</sup> party libraries: Spark, Log4J Appenders, Flume, Kafka Connect...
- Kinesis Firehose
- AWS Lambda
- (Kinesis Consumer Enhanced Fan-Out discussed in the next lecture)



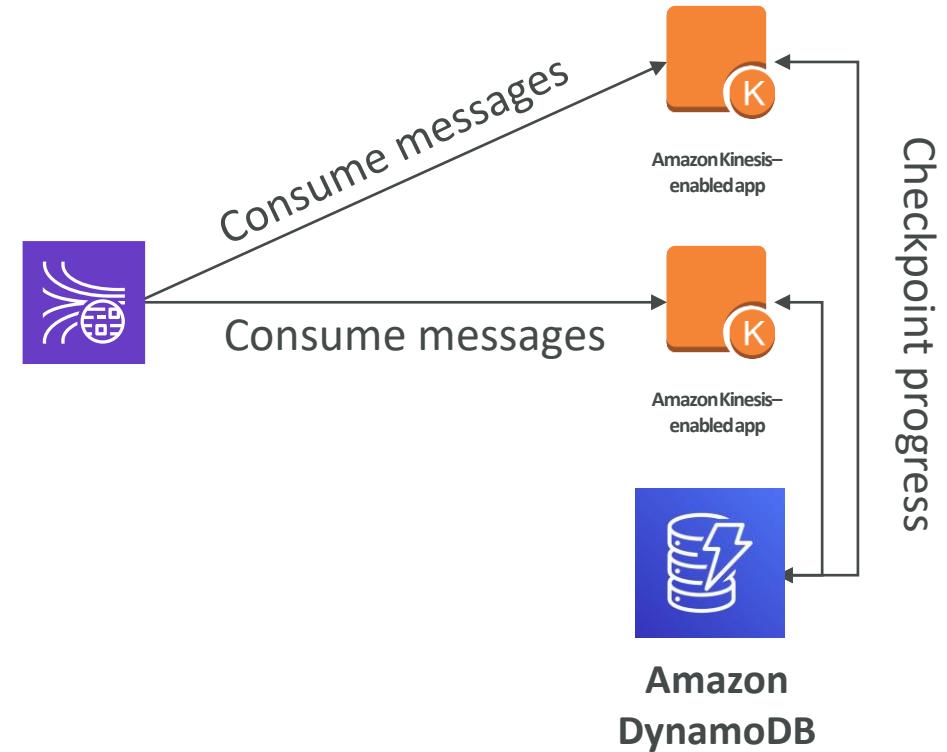
# Kinesis Consumer SDK - GetRecords

- **Classic Kinesis** - Records are polled by consumers from a shard
- **Each shard has 2 MB total aggregate throughput**
- GetRecords returns up to 10MB of data (then throttle for 5 seconds) or up to 10000 records
- Maximum of 5 GetRecords API calls per shard per second = 200ms latency
- If 5 consumers application consume from the same shard, means every consumer can poll once a second and receive less than 400 KB/s



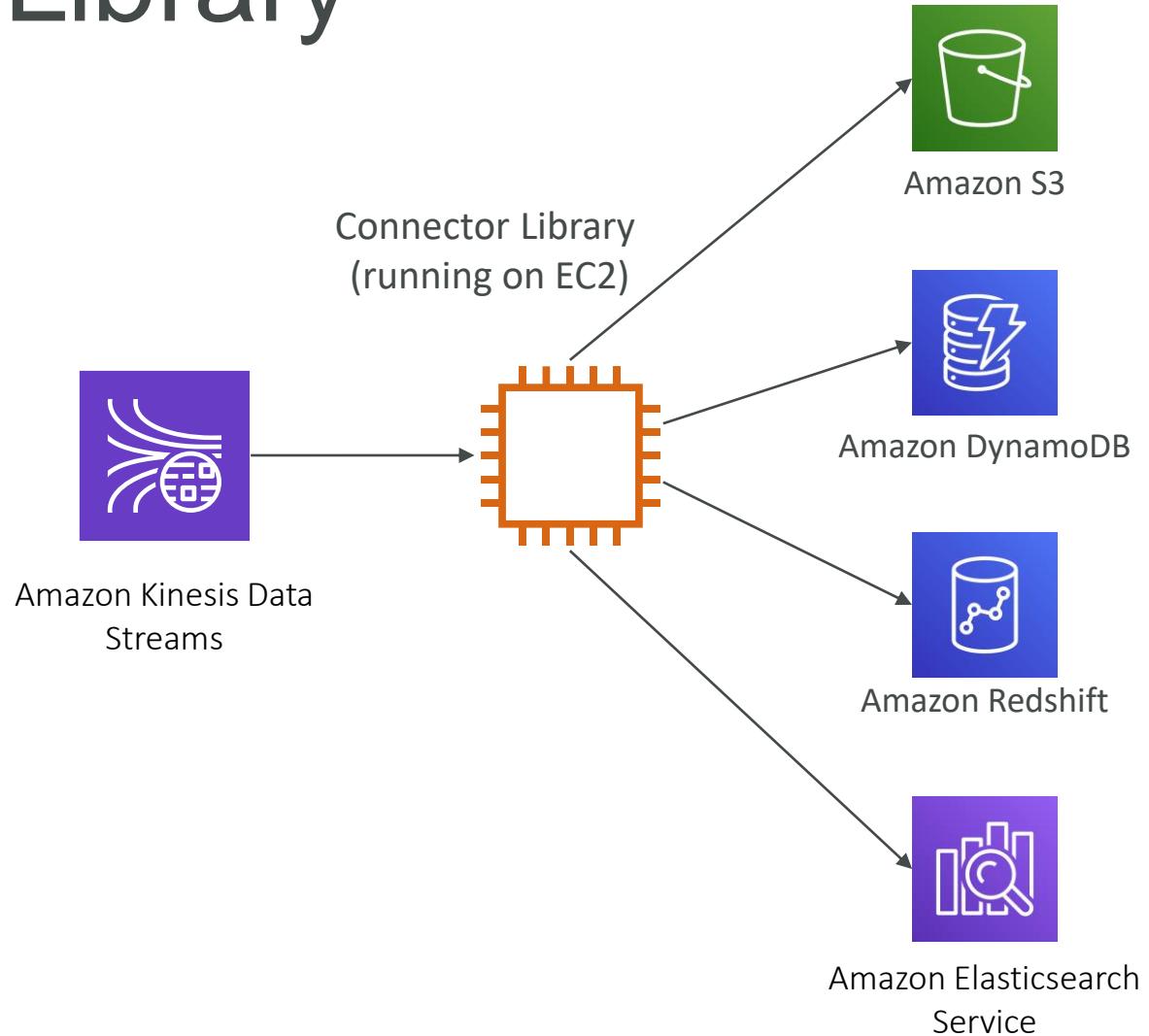
# Kinesis Client Library (KCL)

- Java-first library but exists for other languages too (Golang, Python, Ruby, Node, .NET ...)
- Read records from Kinesis produced with the KPL (de-aggregation)
- Share multiple shards with multiple consumers in one “group”, **shard discovery**
- **Checkpointing** feature to resume progress
- Leverages DynamoDB for coordination and checkpointing (one row per shard)
  - Make sure you provision enough WCU / RCU
  - Or use On-Demand for DynamoDB
  - Otherwise DynamoDB may slow down KCL
- Record processors will process the data



# Kinesis Connector Library

- Older Java library (2016), leverages the KCL library
- Write data to:
  - Amazon S3
  - DynamoDB
  - Redshift
  - ElasticSearch
- Kinesis Firehose replaces the Connector Library for a few of these targets, Lambda for the others

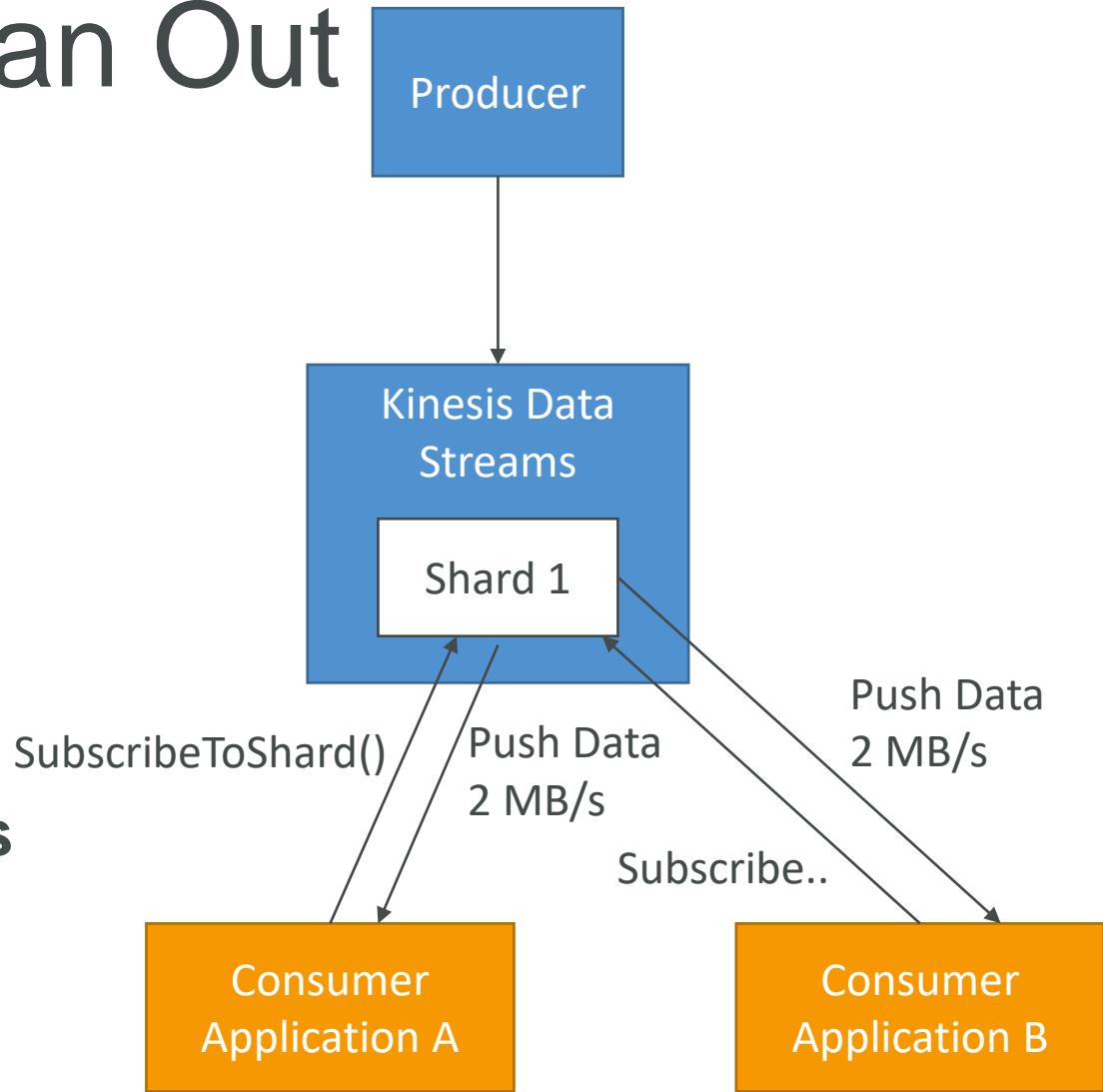


# AWS Lambda sourcing from Kinesis

- AWS Lambda can source records from Kinesis Data Streams
- Lambda consumer has a library to de-aggregate record from the KPL
- Lambda can be used to run lightweight ETL to:
  - Amazon S3
  - DynamoDB
  - Redshift
  - ElasticSearch
  - Anywhere you want
- Lambda can be used to trigger notifications / send emails in real time
- Lambda has a configurable batch size (more in Lambda section)

# Kinesis Enhanced Fan Out

- New **game-changing** feature from August 2018.
- Works with KCL 2.0 and AWS Lambda (Nov 2018)
- Each Consumer get 2 MB/s of provisioned throughput per shard
- That means 20 consumers will get 40MB/s per shard aggregated
- No more 2 MB/s limit!
- Enhanced Fan Out: Kinesis **pushes** data to consumers over HTTP/2
- Reduce latency (~70 ms)

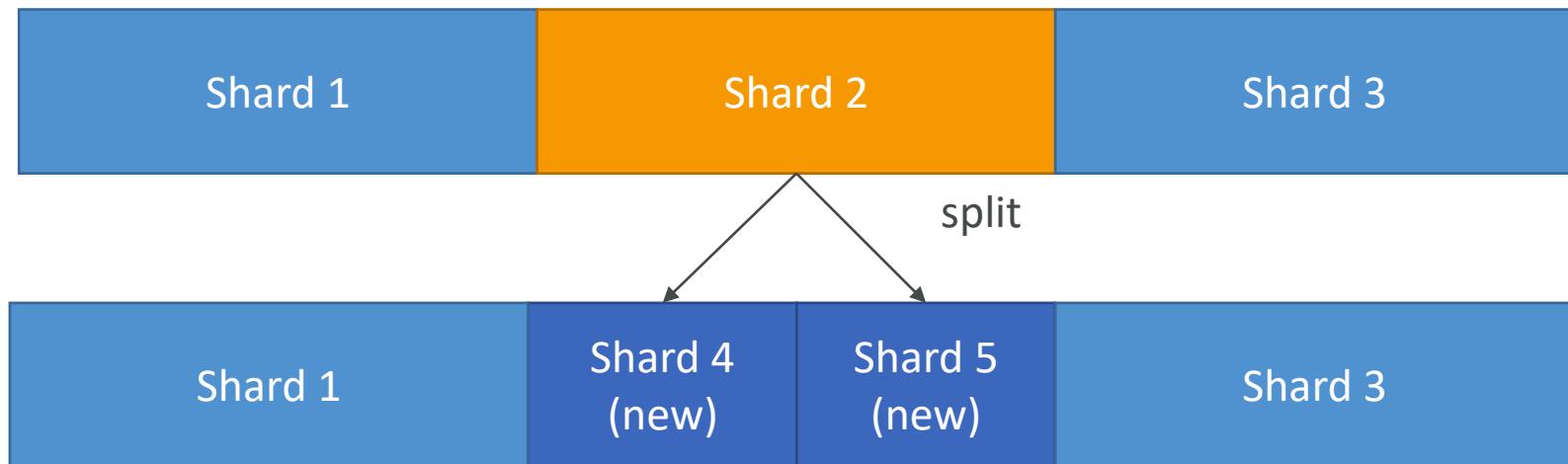


# Enhanced Fan-Out vs Standard Consumers

- Standard consumers:
  - Low number of consuming applications (1,2,3...)
  - Can tolerate ~200 ms latency
  - Minimize cost
- Enhanced Fan Out Consumers:
  - Multiple Consumer applications for the same Stream
  - Low Latency requirements ~70ms
  - Higher costs (see Kinesis pricing page)
  - Default limit of 5 consumers using enhanced fan-out per data stream

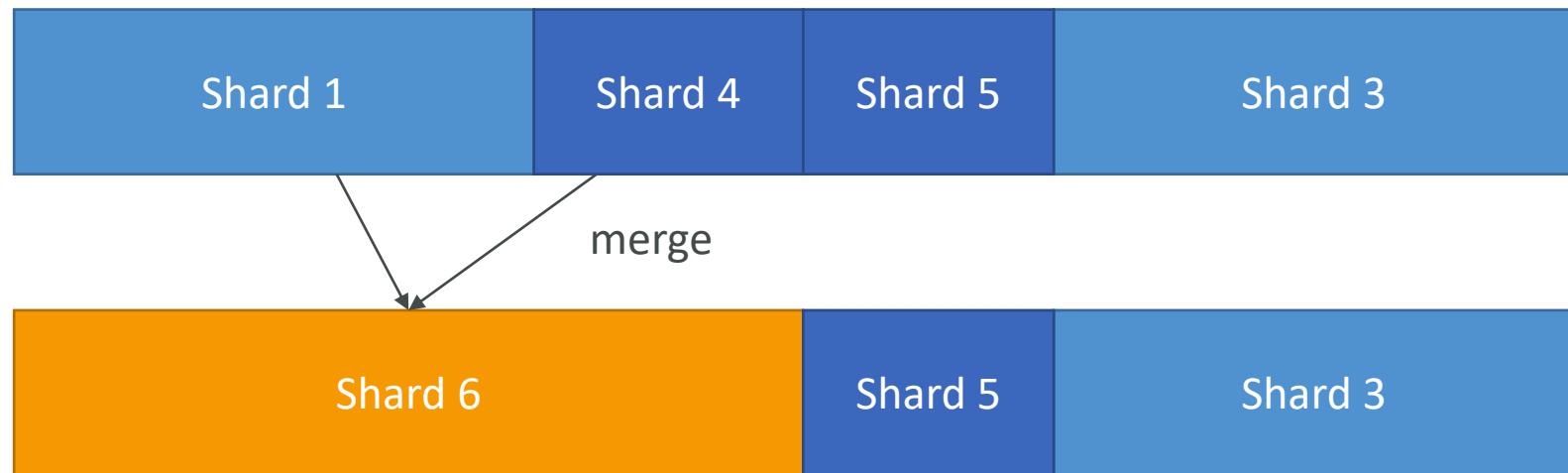
# Kinesis Operations – Adding Shards

- Also called “Shard Splitting”
- Can be used to increase the Stream capacity (1 MB/s data in per shard)
- Can be used to divide a “hot shard”
- The old shard is closed and will be deleted once the data is expired



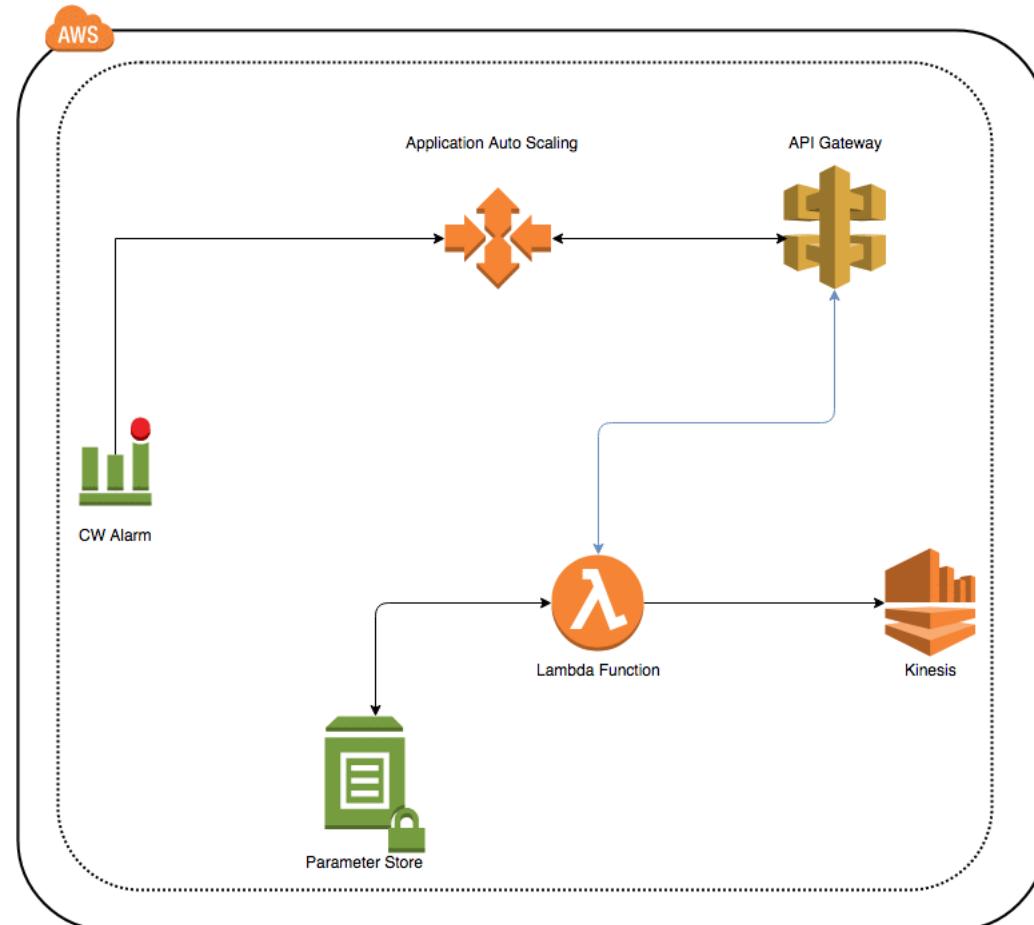
# Kinesis Operations – Merging Shards

- Decrease the Stream capacity and save costs
- Can be used to group two shards with low traffic
- Old shards are closed and deleted based on data expiration



# Kinesis Operations – Auto Scaling

- Auto Scaling is not a native feature of Kinesis
- The API call to change the number of shards is `UpdateShardCount`
- We can implement Auto Scaling with AWS Lambda
- See:  
<https://aws.amazon.com/blogs/big-data/scaling-amazon-kinesis-data-streams-with-aws-application-auto-scaling/>



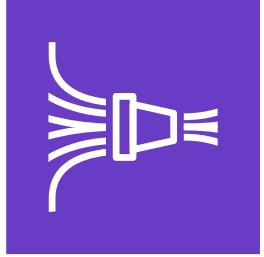
# Kinesis Scaling Limitations

- Resharding cannot be done in parallel. Plan capacity in advance
- You can only perform one resharding operation at a time and it takes a few seconds
- For 1000 shards, it takes 30K seconds (8.3 hours) to double the shards to 2000
- **You can't do the following:**
  - Scale more than twice for each rolling 24-hour period for each stream
  - Scale up to more than double your current shard count for a stream
  - Scale down below half your current shard count for a stream
  - Scale up to more than 500 shards in a stream
  - Scale a stream with more than 500 shards down unless the result is fewer than 500 shards
  - Scale up to more than the shard limit for your account

# Kinesis Security

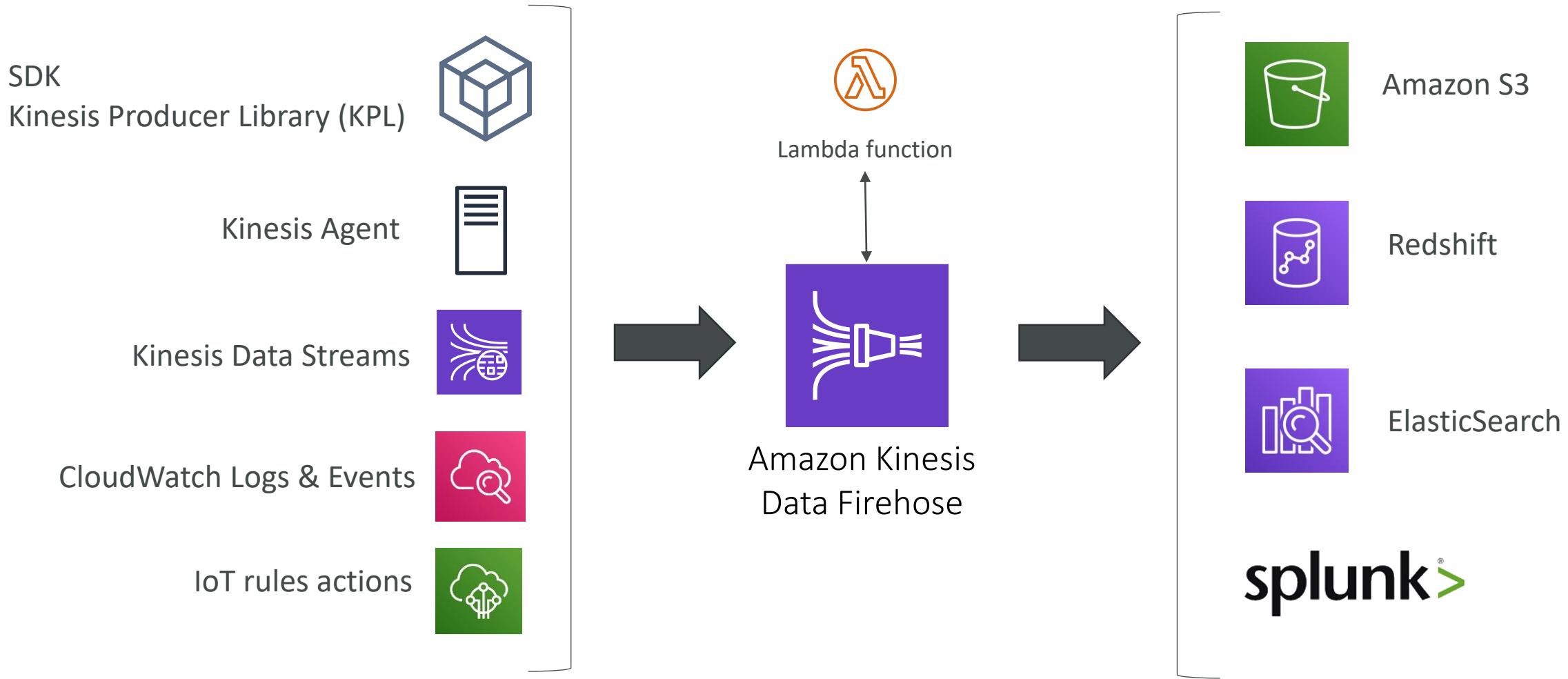
- Control access / authorization using IAM policies
- Encryption in flight using HTTPS endpoints
- Encryption at rest using KMS
- Client side encryption must be manually implemented (harder)
- VPC Endpoints available for Kinesis to access within VPC

# AWS Kinesis Data Firehose

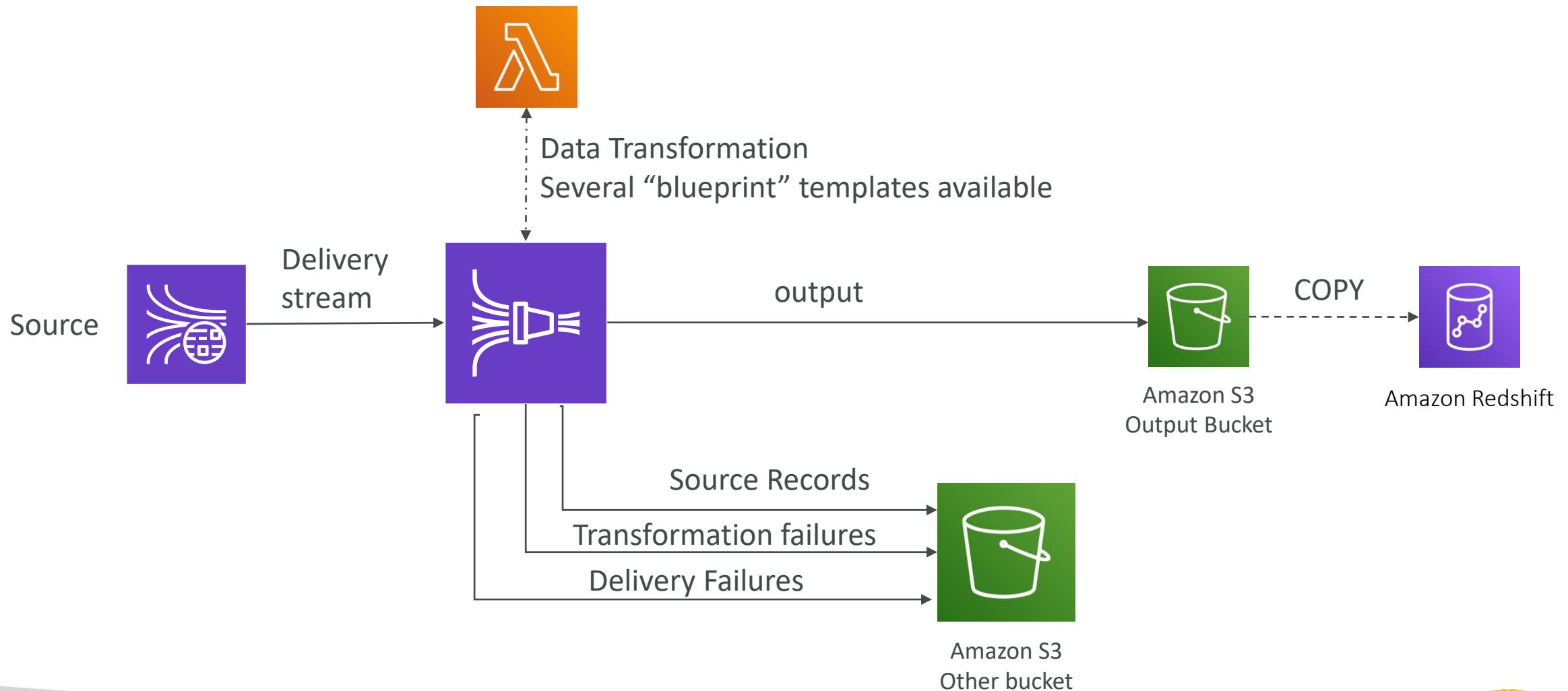


- Fully Managed Service, no administration
- **Near Real Time** (60 seconds latency minimum for non full batches)
- Load data into Redshift / Amazon S3 / ElasticSearch / Splunk
- Automatic scaling
- Supports many data formats
- Data Conversions from JSON to Parquet / ORC (only for S3)
- Data Transformation through AWS Lambda (ex: CSV => JSON)
- Supports **compression** when target is Amazon S3 (GZIP, ZIP, and SNAPPY)
- Only GZIP is the data is further loaded into Redshift
- Pay for the amount of data going through Firehose
- Spark / KCL do not read from KDF

# Kinesis Data Firehose Diagram



# Kinesis Data Firehose Delivery Diagram



# Firehose Buffer Sizing

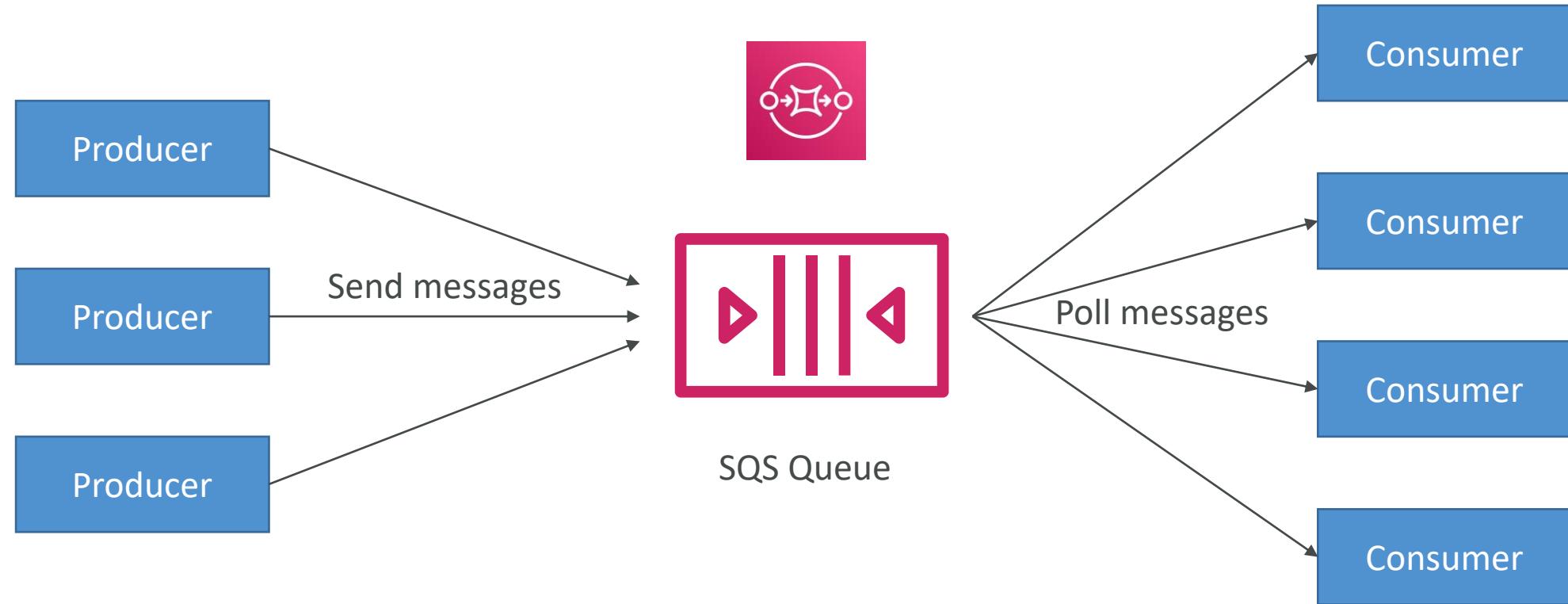
- Firehose accumulates records in a buffer
- The buffer is flushed based on time and size rules
- Buffer Size (ex: 32MB): if that buffer size is reached, it's flushed
- Buffer Time (ex: 2 minutes): if that time is reached, it's flushed
- Firehose can automatically increase the buffer size to increase throughput
- High throughput => Buffer Size will be hit
- Low throughput => Buffer Time will be hit

# Kinesis Data Streams vs Firehose

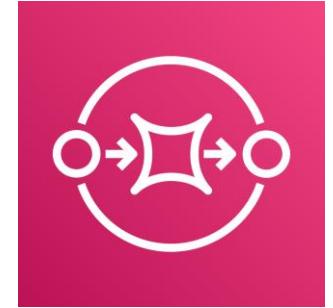
- Streams
  - Going to write custom code (producer / consumer)
  - Real time (~200 ms latency for classic, ~70 ms latency for enhanced fan-out)
  - Must manage scaling (shard splitting / merging)
  - Data Storage for 1 to 7 days, replay capability, multi consumers
  - Use with Lambda to insert data in real-time to ElasticSearch (for example)
- Firehose
  - Fully managed, send to S3, Splunk, Redshift, ElasticSearch
  - Serverless data transformations with Lambda
  - **Near** real time (lowest buffer time is 1 minute)
  - Automated Scaling
  - No data storage

# AWS SQS

## What's a queue?



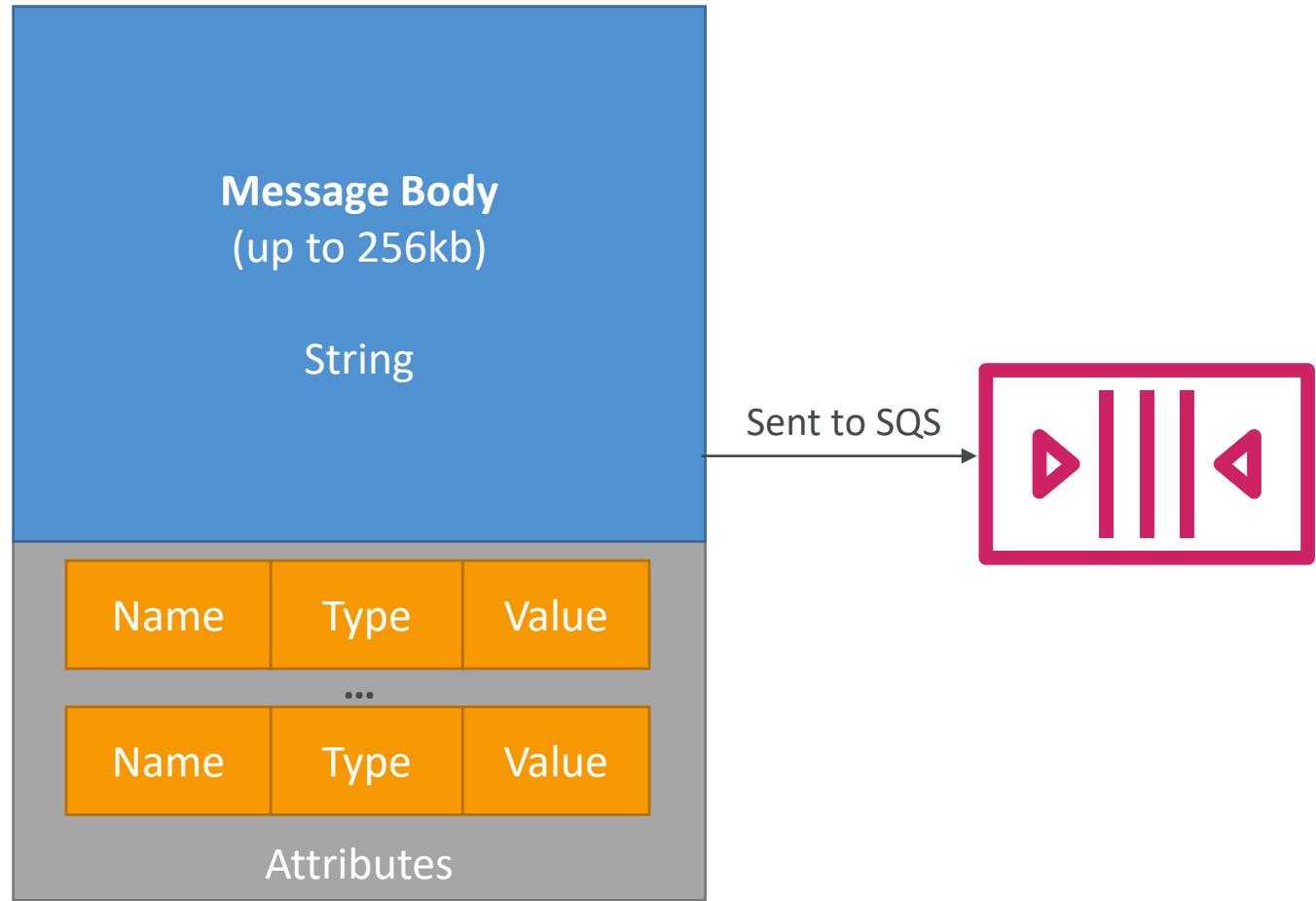
# AWS SQS – Standard Queue



- Oldest offering (over 10 years old)
- Fully managed
- Scales from 1 message per second to 10,000s per second
- Default retention of messages: 4 days, maximum of 14 days
- No limit to how many messages can be in the queue
- Low latency (<10 ms on publish and receive)
- Horizontal scaling in terms of number of consumers
- Can have duplicate messages (at least once delivery, occasionally)
- Can have out of order messages (best effort ordering)
- Limitation of 256KB per message sent

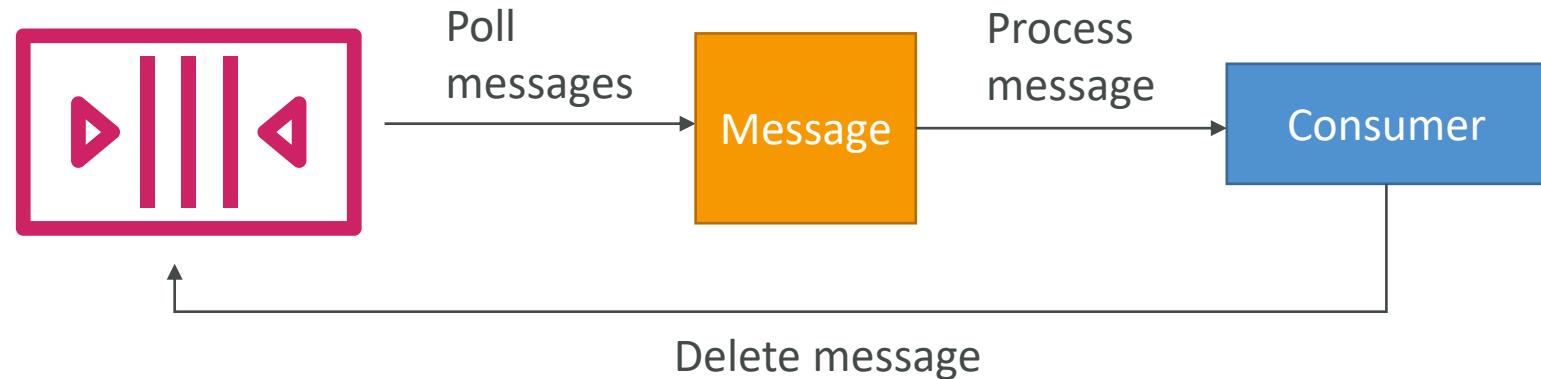
# SQS – Producing Messages

- Define Body
- Add message attributes (metadata – optional)
- Provide Delay Delivery (optional)
- Get back
  - Message identifier
  - MD5 hash of the body



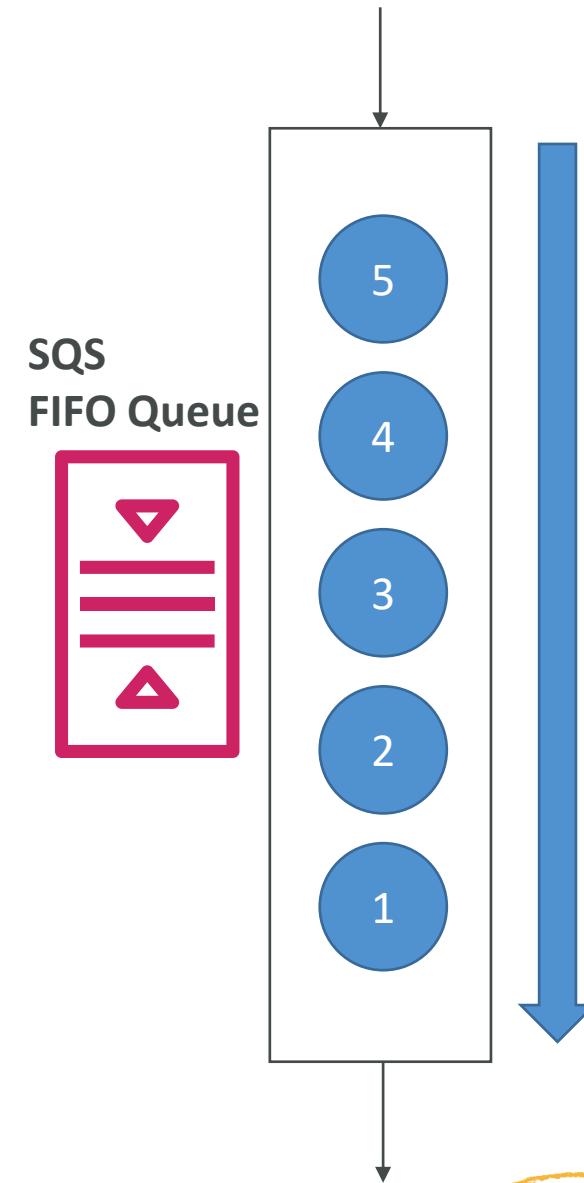
# SQS – Consuming Messages

- Consumers...
- Poll SQS for messages (receive up to 10 messages at a time)
- Process the message within the visibility timeout
- Delete the message using the message ID & receipt handle



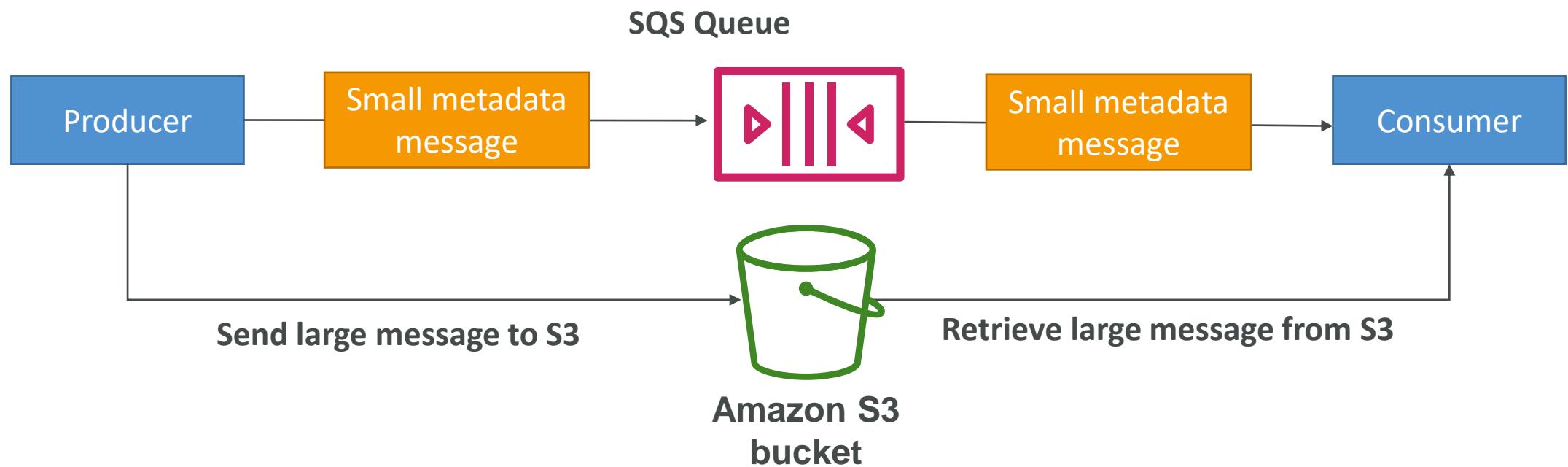
# AWS SQS – FIFO Queue

- Newer offering (First In - First out) – not available in all regions!
- Name of the queue must end in .fifo
- Lower throughput (up to 3,000 per second with batching, 300/s without)
- Messages are processed in order by the consumer
- Messages are sent exactly once
- 5-minute interval de-duplication using “Duplication ID”



# SQS Extended Client

- Message size limit is 256KB, how to send large messages?
- Using the SQS Extended Client (Java Library)



# AWS SQS Use Cases

- Decouple applications  
(for example to handle payments asynchronously)
- Buffer writes to a database  
(for example a voting application)
- Handle large loads of messages coming in  
(for example an email sender)
- SQS can be integrated with Auto Scaling through CloudWatch!

# SQS Limits

- Maximum of 120,000 in-flight messages being processed by consumers
- Batch Request has a maximum of 10 messages – max 256KB
- Message content is XML, JSON, Unformatted text
- Standard queues have an unlimited TPS
- FIFO queues support up to 3,000 messages per second (using batching)
- Max message size is 256KB (or use Extended Client)
- Data retention from 1 minute to 14 days
- Pricing:
  - Pay per API Request
  - Pay per network usage

# AWS SQS Security

- Encryption in flight using the HTTPS endpoint
- Can enable SSE (Server Side Encryption) using KMS
  - Can set the CMK (Customer Master Key) we want to use
  - SSE only encrypts the body, not the metadata (message ID, timestamp, attributes)
- IAM policy must allow usage of SQS
- SQS queue access policy
  - Finer grained control over IP
  - Control over the time the requests come in

# Kinesis Data Stream vs SQS

- **Kinesis Data Stream:**

- Data can be consumed many times
- Data is deleted after the retention period
- Ordering of records is preserved (at the shard level) – even during replays
- Build multiple applications reading from the same stream independently (Pub/Sub)
- “Streaming MapReduce” querying capability
- Checkpointing needed to track progress of consumption
- Shards (capacity) must be provided ahead of time

- **SQS:**

- Queue, decouple applications
- One application per queue
- Records are deleted after consumption (ack / fail)
- Messages are processed independently for standard queue
- Ordering for FIFO queues
- Capability to “delay” messages
- Dynamic scaling of load (no-ops)

# Kinesis Data Streams vs SQS

	Kinesis Data Streams	Kinesis Data Firehose	Amazon SQS Standard	Amazon SQS FIFO
Managed by AWS	yes	yes	yes	yes
Ordering	Shard / Key	No	No	Specify Group ID
Delivery	At least once	At least once	At least once	Exactly Once
Replay	Yes	No	No	No
Max Data Retention	7 days	No	14 days	14 days
Scaling	Provision Shards: 1MB/s producer 2MB/s consumer	No limit	No limit	~3000 messages per second with batching (soft limit)
Max Object Size	1MB	128 MB at destination	256KB (more if using extended lib)	256KB (more if using extended lib)

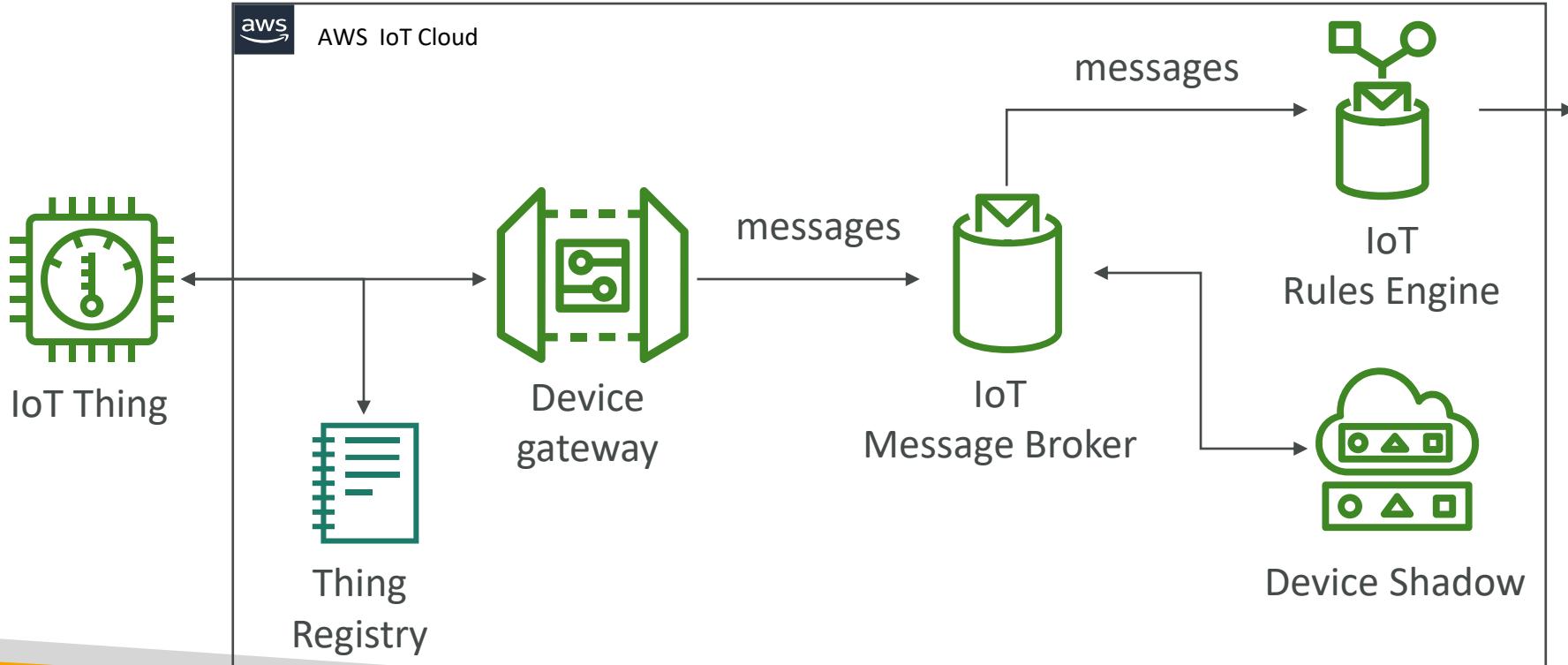
# SQS vs Kinesis – Use cases

- SQS Use cases :
  - Order processing
  - Image Processing
  - Auto scaling queues according to messages.
  - Buffer and Batch messages for future processing.
  - Request Offloading
- Amazon Kinesis Data Streams Use cases :
  - Fast log and event data collection and processing
  - Real Time metrics and reports
  - Mobile data capture
  - Real Time data analytics
  - Gaming data feed
  - Complex Stream Processing
  - Data Feed from “Internet of Things”

# IoT Overview

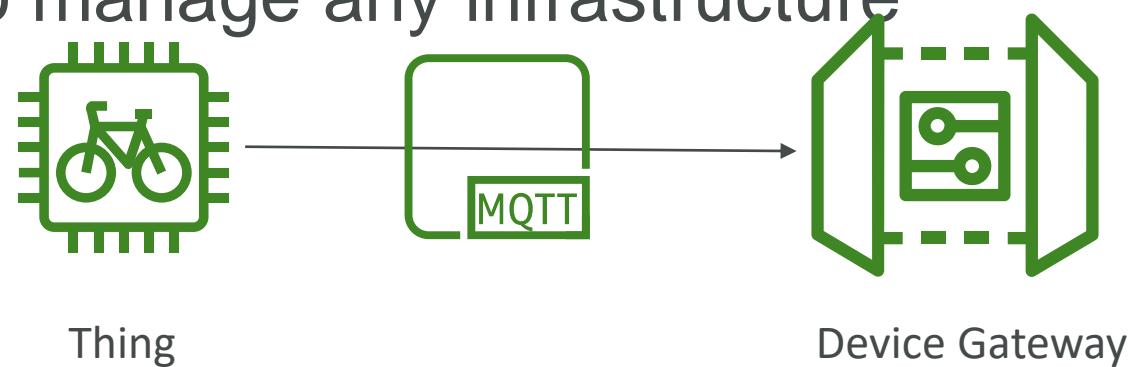


- We deploy IoT devices (“Things”)
- We configure them and retrieve data from them



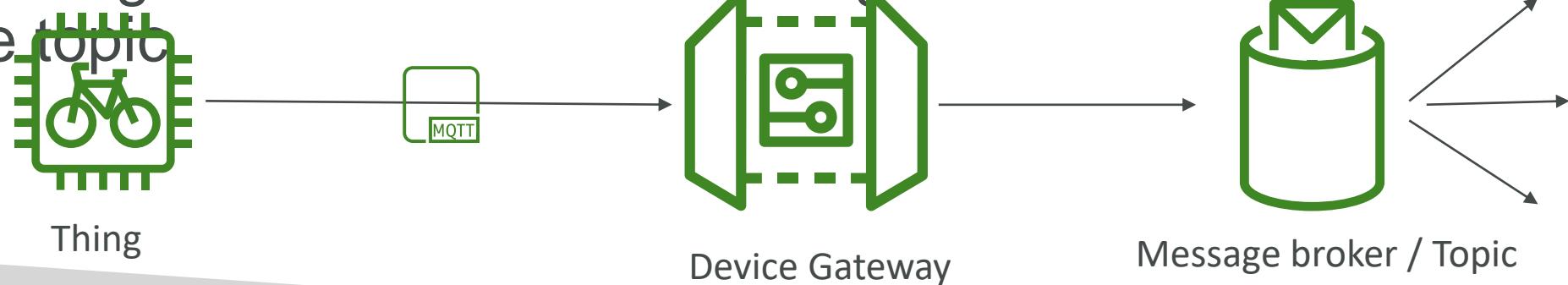
# IoT Device Gateway

- Serves as the entry point for IoT devices connecting to AWS
- Allows devices to securely and efficiently communicate with AWS IoT
- Supports the MQTT, WebSockets, and HTTP 1.1 protocols
- Fully managed and scales automatically to support over a billion devices
- No need to manage any infrastructure



# IoT Message Broker

- Pub/sub (publishers/subscribers) messaging pattern - low latency
- Devices can communicate with one another this way
- Messages sent using the MQTT, WebSockets, or HTTP 1.1 protocols
- Messages are published into topics (just like SNS)
- Message Broker forwards messages to all clients connected to the topic



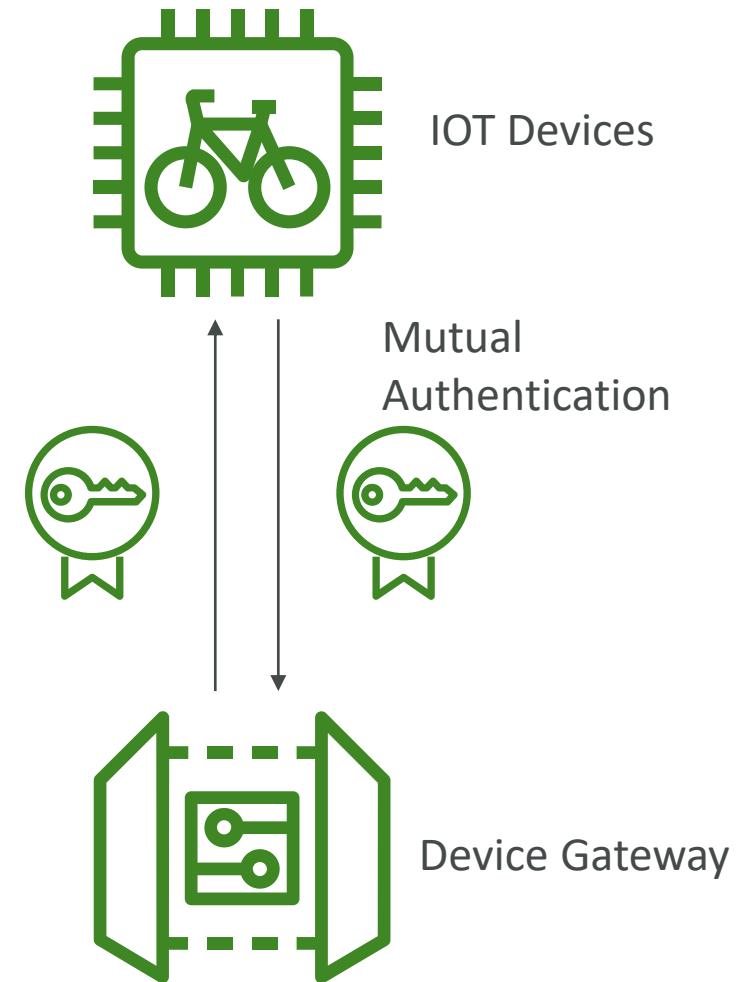
# IoT Thing Registry = IAM of IoT



- All connected IoT devices are represented in the AWS IoT registry
- Organizes the resources associated with each device in the AWS Cloud
- Each device gets a unique ID
- Supports metadata for each device (ex: Celsius vs Fahrenheit, etc...)
- Can create X.509 certificate to help IoT devices connect to AWS
- IoT Groups: group devices together and apply permissions to the group

# Authentication

- 3 possible authentication methods for Things:
  - Create X.509 certificates and load them securely onto the Things
  - AWS SigV4
  - Custom tokens with Custom authorizers
- For mobile apps:
  - Cognito identities (extension to Google, Facebook login, etc...)
- Web / Desktop / CLI:
  - IAM
  - Federated Identities

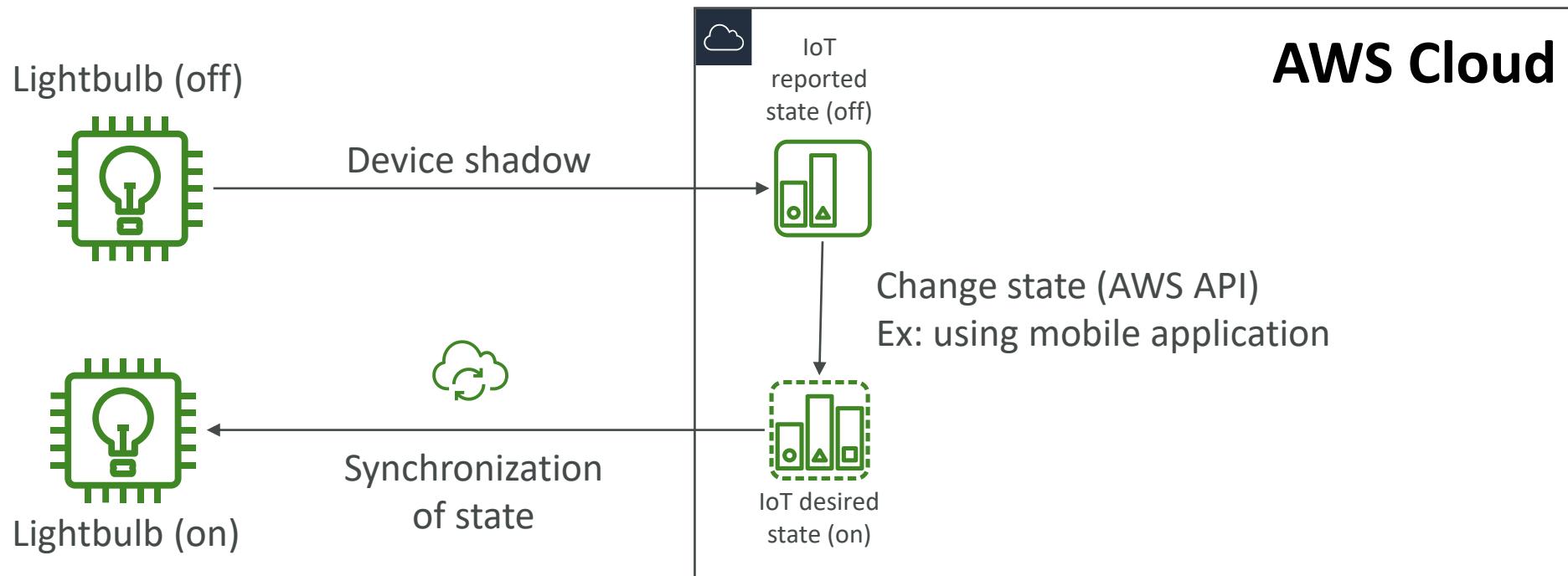


# Authorization

- AWS IoT policies:
  - Attached to X.509 certificates or Cognito Identities
  - Able to revoke any device at any time
  - IoT Policies are JSON documents
  - Can be attached to groups instead of individual Things.
- IAM Policies:
  - Attached to users, group or roles
  - Used for controlling IoT AWS APIs

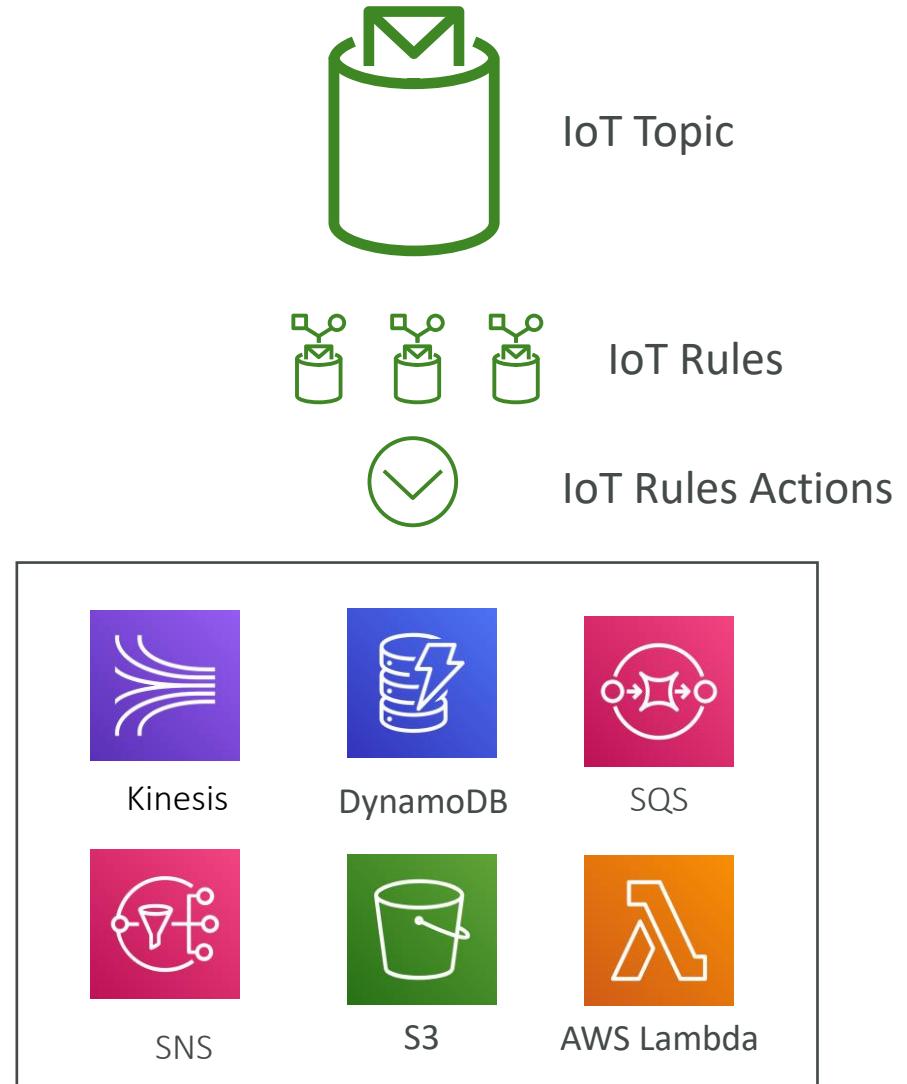
# Device Shadow

- JSON document representing the state of a connected Thing
- We can set the state to a different desired state (ex: light on)
- The IoT thing will retrieve the state when online and adapt



# Rules Engine

- Rules are defined on the MQTT topics
- Rules = when it's triggered | Action = what is does
- Rules use cases:
  - Augment or filter data received from a device
  - Write data received from a device to a **DynamoDB** database
  - Save a file to **S3**
  - Send a push notification to all users using **SNS**
  - Publish data to a **SQS** queue
  - Invoke a **Lambda** function to extract data
  - Process messages from a large number of devices using **Amazon Kinesis**
  - Send data to the Amazon **Elasticsearch Service**
  - Capture a **CloudWatch metric** and Change a **CloudWatch alarm**
  - Send the data from an MQTT message to **Amazon Machine Learning** to make predictions based on an Amazon ML model
  - & more
- Rules need IAM Roles to perform their actions



# IoT Greengrass

- IoT Greengrass brings the compute layer to the device directly
- You can execute AWS Lambda functions on the devices:
  - Pre-process the data
  - Execute predictions based on ML models
  - Keep device data in sync
  - Communicate between local devices
- Operate offline
- Deploy functions from the cloud directly to the devices

## Local Device



AWS IoT Greengrass



Lambda  
function

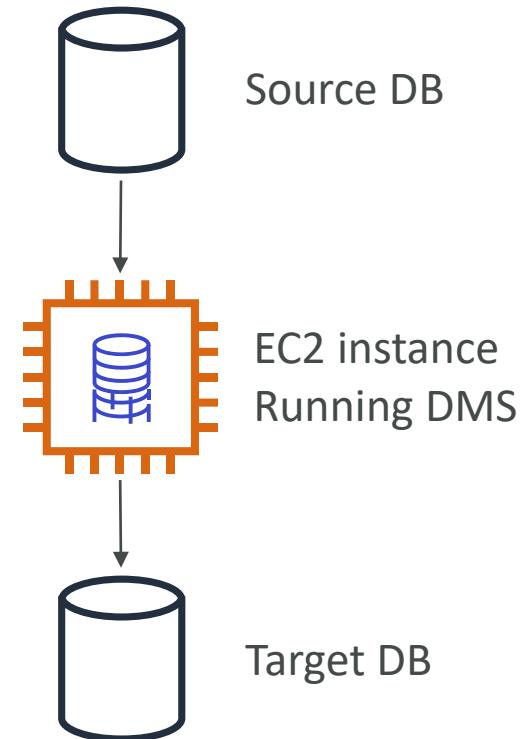


IoT thing  
coffee pot

# DMS – Database Migration Service



- Quickly and securely migrate databases to AWS, resilient, self healing
- The source database remains available during the migration
- Supports:
  - Homogeneous migrations: ex Oracle to Oracle
  - Heterogeneous migrations: ex Microsoft SQL Server to Aurora
- Continuous Data Replication using CDC
- You must create an EC2 instance to perform the replication tasks



# DMS Sources and Targets

## SOURCES:

- On-Premise and EC2 instances databases: *Oracle, MS SQL Server, MySQL, MariaDB, PostgreSQL, MongoDB, SAP, DB2*
- Azure: *Azure SQL Database*
- Amazon RDS: all including Aurora
- Amazon S3

## TARGETS:

- On-Premise and EC2 instances databases: Oracle, MS SQL Server, MySQL, MariaDB, PostgreSQL, SAP
- Amazon RDS
- Amazon Redshift
- Amazon DynamoDB
- Amazon S3
- ElasticSearch Service
- Kinesis Data Streams
- DocumentDB

# AWS Schema Conversion Tool (SCT)

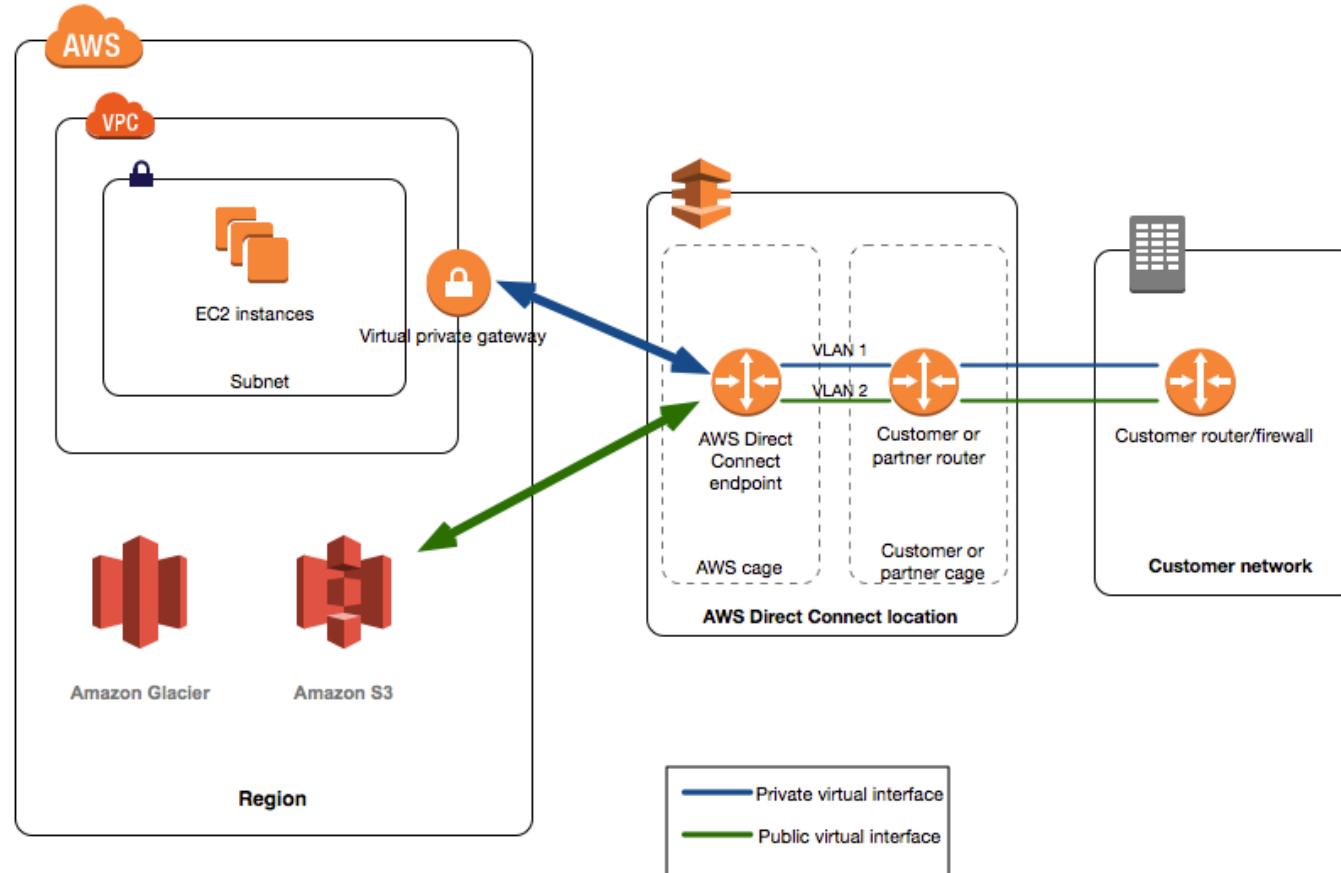
- Convert your Database's Schema from one engine to another
- Example OLTP: (SQL Server or Oracle) to MySQL, PostgreSQL, Aurora
- Example OLAP: (Teradata or Oracle) to Amazon Redshift
- **You can use AWS SCT to create AWS DMS endpoints and tasks.**

# Direct Connect



- Provides a dedicated **private** connection from a remote network to your VPC
- Can setup multiple **1 Gbps** or **10 Gbps dedicated network connections**
- Setup Dedicated connection between your DC and Direct Connect locations
- You need to setup a Virtual Private Gateway on your VPC
- Access public resources (S3) and private (EC2) on same connection
- Use Cases:
  - Increase bandwidth throughput - working with large data sets – lower cost
  - More consistent network experience - applications using real-time data feeds
  - Hybrid Environments (on prem + cloud)
  - Enhanced security (private connection)
- Supports both IPv4 and IPv6
- High-availability: Two DC as failover or use Site-to-Site VPN as a failover

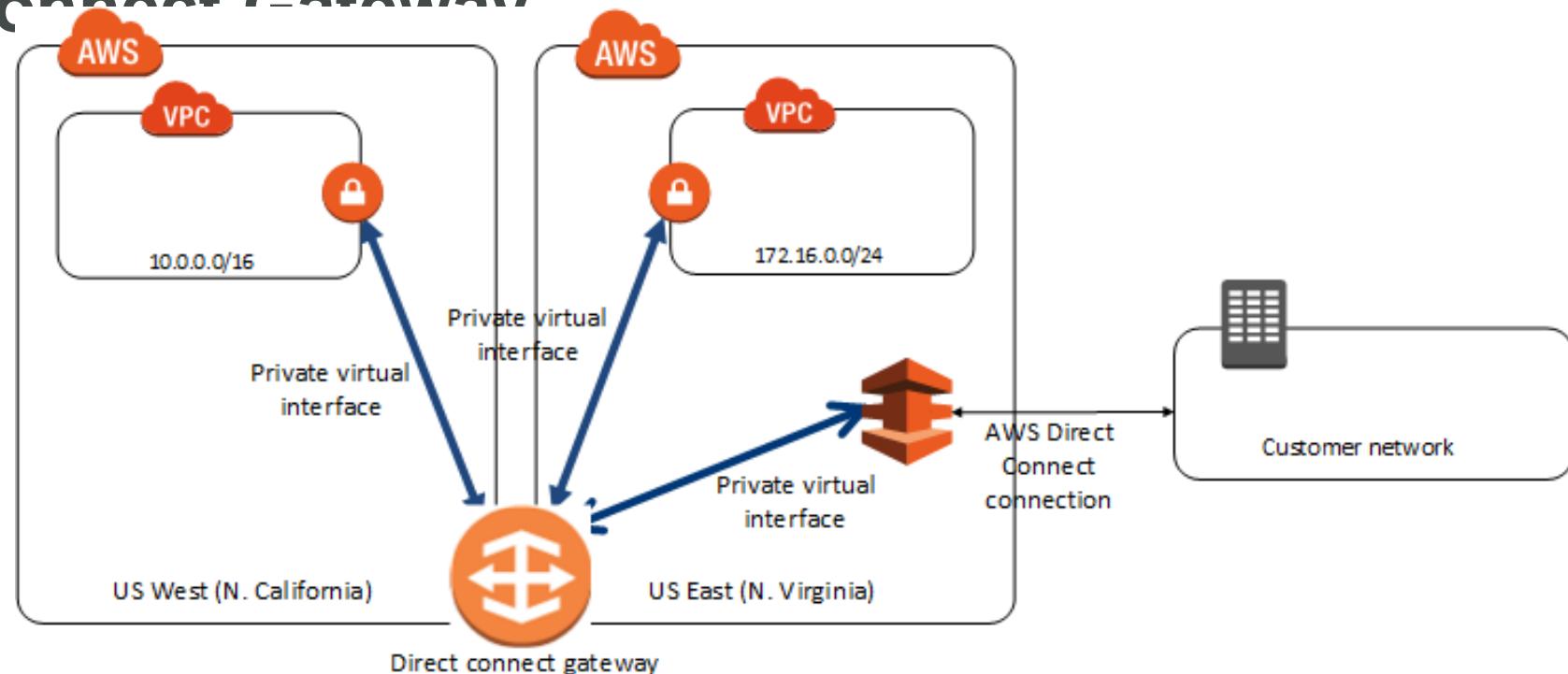
# Direct Connect Diagram



[https://docs.aws.amazon.com/directconnect/latest/UserGuide/images/direct\\_connect\\_overview.png](https://docs.aws.amazon.com/directconnect/latest/UserGuide/images/direct_connect_overview.png)

# Direct Connect Gateway

- If you want to setup a Direct Connect to one or more VPC in many different regions (same account), you must use a **Direct Connect Gateway**



<https://docs.aws.amazon.com/directconnect/latest/UserGuide/direct-connect-gateways.html>

# Snowball



- Physical data transport solution that helps moving TBs or PBs of data in or out of AWS
- Alternative to moving data over the network (and paying network fees)
- Secure, tamper resistant, uses KMS 256 bit encryption
- Tracking using SNS and text messages. E-ink shipping label
- Pay per data transfer job
- Use cases: large data cloud migrations, DC decommission, disaster recovery
- If it takes more than a **week** to transfer over the network, use Snowball devices!



# Snowball Process

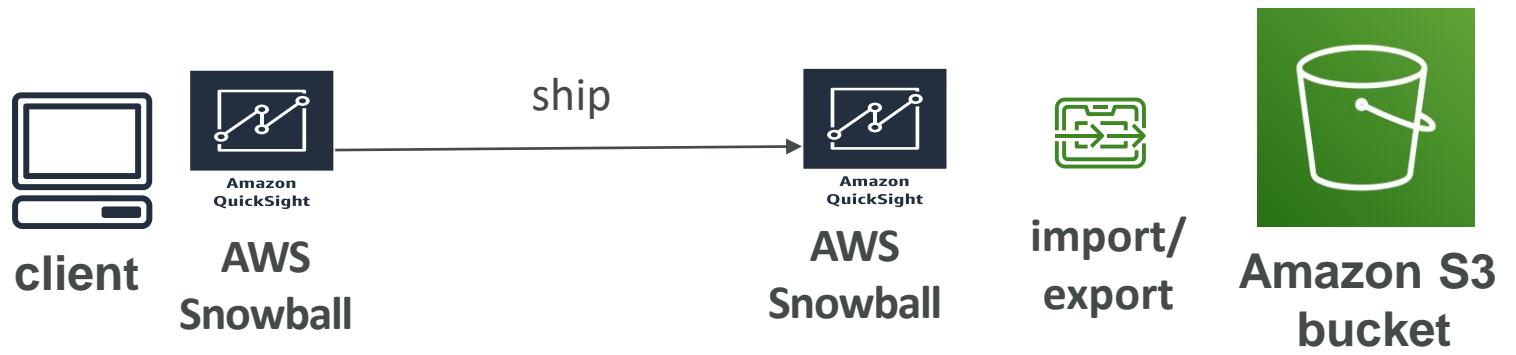
1. Request snowball devices from the AWS console for delivery
2. Install the snowball client on your servers
3. Connect the snowball to your servers and copy files using the client
4. Ship back the device when you're done (goes to the right AWS facility)
5. Data will be loaded into an S3 bucket
6. Snowball is completely wiped
7. Tracking is done using SNS, text messages and the AWS console

# Snowball Diagrams

- Direct upload to S3:



- With snowball



# Snowball Edge



- Snowball Edges add computational capability to the device
  - 100 TB capacity with either:
    - Storage optimized – 24 vCPU
    - Compute optimized – 52 vCPU & optional GPU
  - Supports a custom EC2 AMI so you can perform processing on the go
  - Supports custom Lambda functions
- 
- Very useful to pre-process the data while moving
  - Use case: data migration, image collation, IoT capture, machine learning



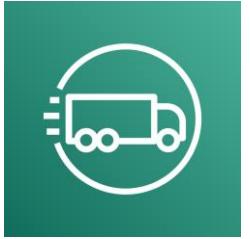
AWS Glue



Lambda  
function

AMI

# AWS Snowmobile



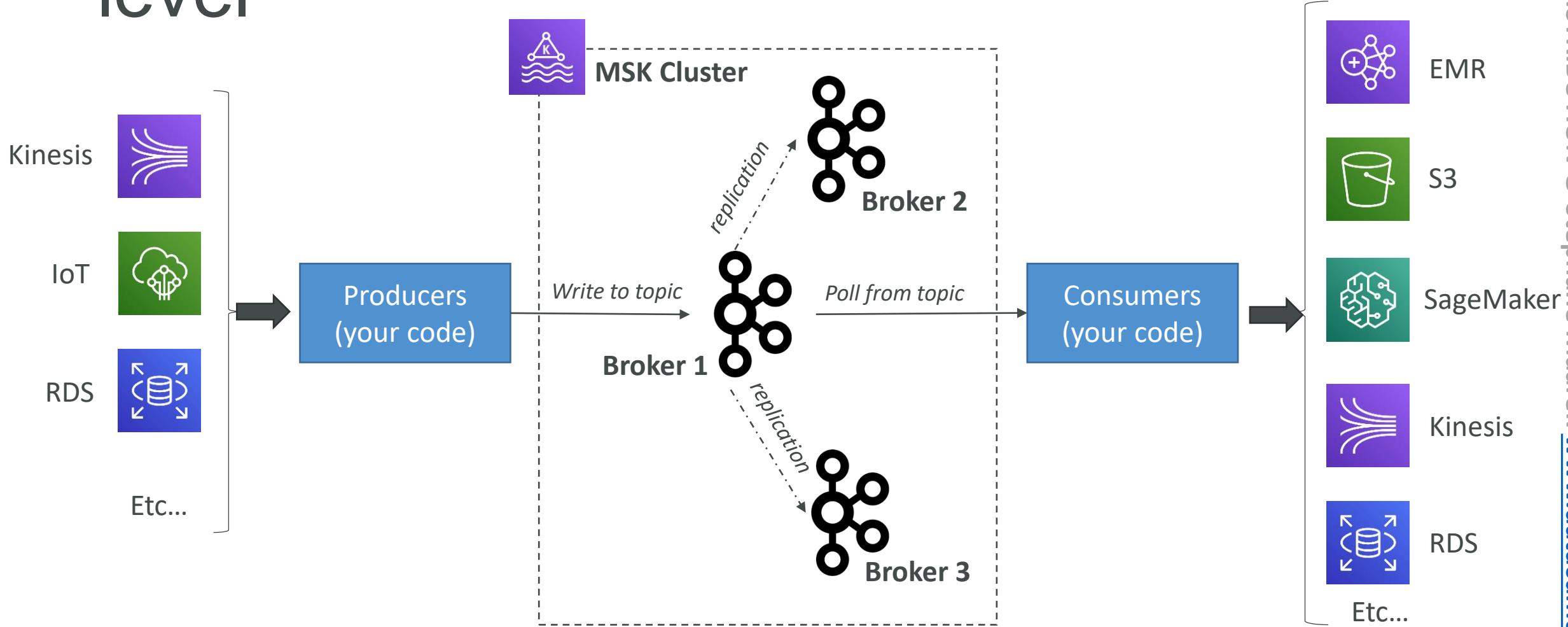
- Transfer exabytes of data (1 EB = 1,000 PB = 1,000,000 TBs)
- Each Snowmobile has 100 PB of capacity (use multiple in parallel)
- Better than Snowball if you transfer more than 10 PB

# MSK Managed Streaming for Apache Kafka



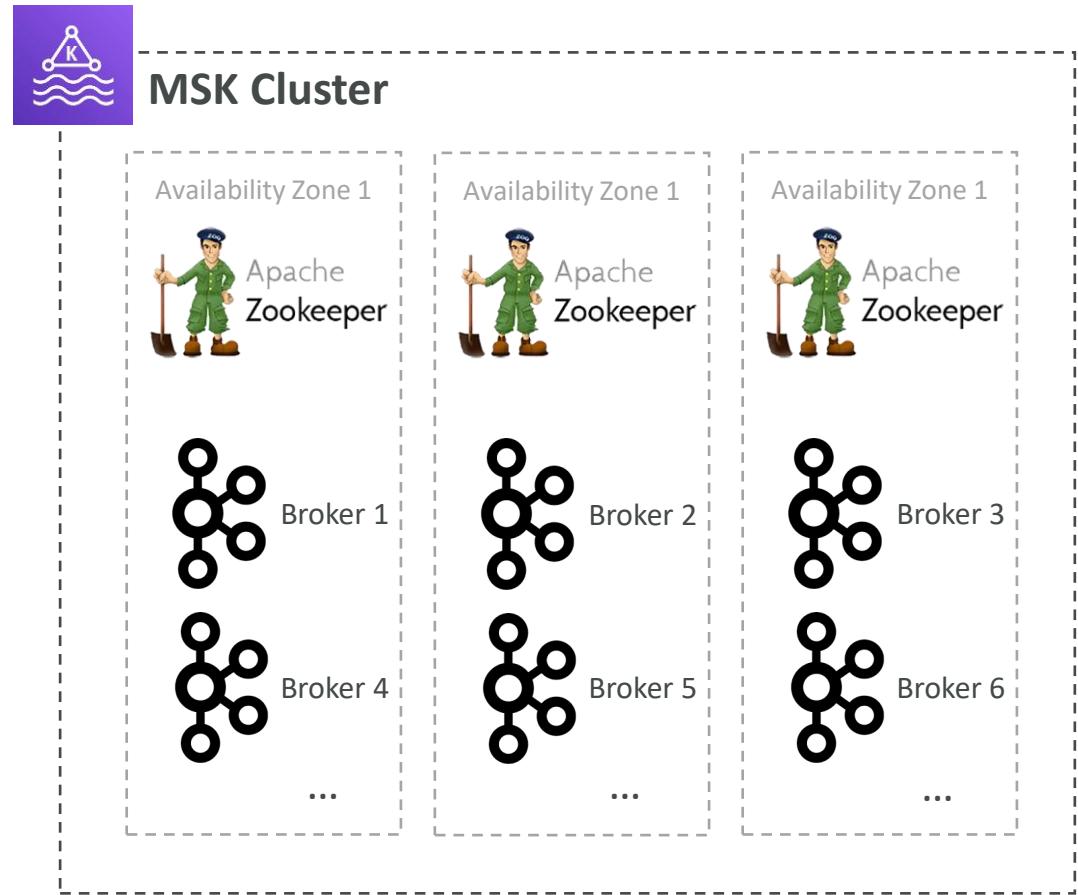
- Fully managed Apache Kafka on AWS (alternative to Kinesis)
- Allow you to create, update, delete clusters (control plane)
- MSK creates & manages brokers nodes & Zookeeper nodes for you
- Deploy the MSK cluster in your VPC, multi AZ (up to 3 for HA)
- Automatic recovery from common Apache Kafka failures
- Can create custom configurations for your clusters
- Data is stored on EBS volumes
- You can build producers and consumers of data (data plane)

# Apache Kafka at a high level



# MSK – Configurations

- Choose the number of AZ (3 – recommended, or 2)
- Choose the VPC & Subnets
- The broker instance type (ex: kafka.m5.large)
- The number of brokers per AZ (can add brokers later)
- Size of your EBS volumes (1GB - 16 TB)



# MSK – Override Kafka Configurations

- List of properties you can set:

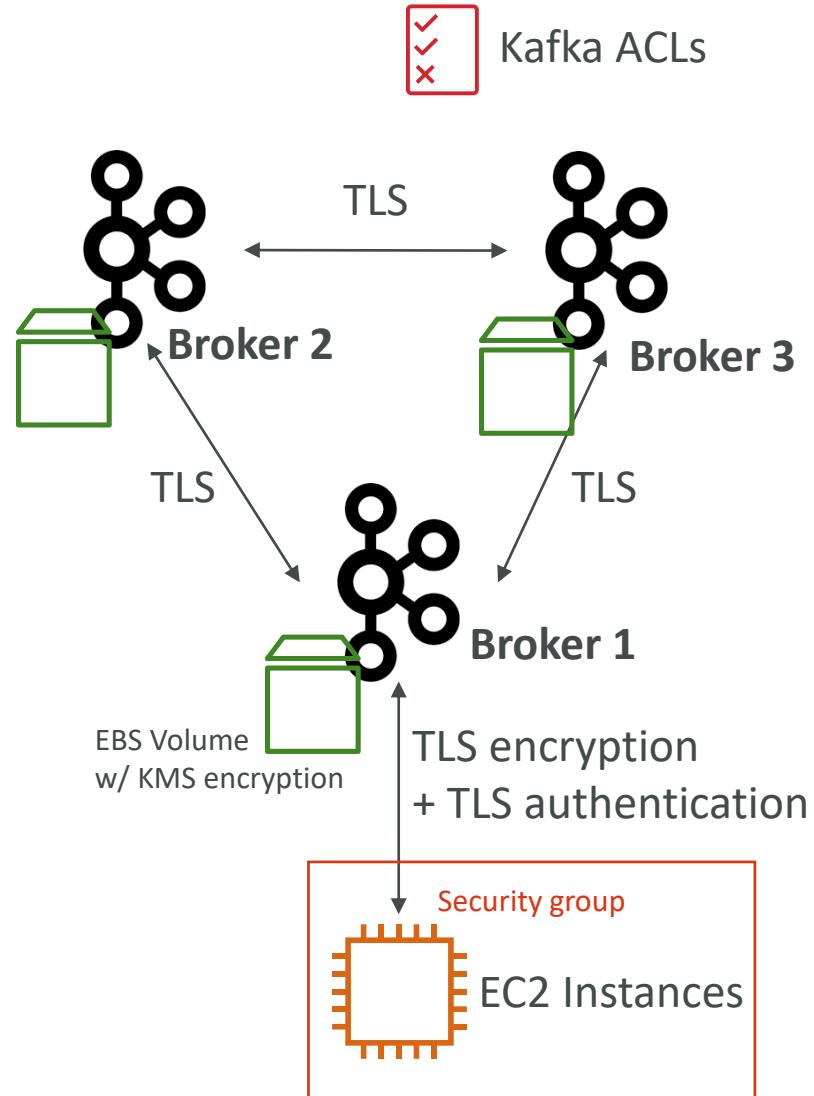
<https://docs.aws.amazon.com/msk/latest/developerguide/msk-configuration-properties.html>

## Important to note:

- Max message size in Kafka by default is 1MB
  - Can override this with the broker **message.max.bytes** setting
  - Must also change the consumer **max.fetch.bytes** setting
- Latency:
  - By default it's low in Kafka 10-40ms (way less than Kinesis)
  - The producer can increase latency to increase batching using **linger.ms**

# MSK – Security

- Can enable encryption in flight using TLS between the brokers
- Can say PLAINTEXT and/or TLS-encrypted between the clients and brokers
- Encryption at rest for your EBS volumes using KMS
- Supports TLS client authentication using a Private Certificate Authority (CA) from ACM
- Authorize specific security groups for your Apache Kafka clients
- Security and ACLs for clients is done within the Apache Kafka cluster



# MSK - Monitoring

- **CloudWatch Metrics**
  - Basic monitoring (cluster and broker metrics)
  - Enhanced monitoring (++enhanced broker metrics)
  - Topic-level monitoring (++enhanced topic-level metrics)
- **Prometheus (Open-Source Monitoring)**
  - Opens a port on the broker to export cluster, broker and topic-level metrics
  - Setup the JMX Exporter (metrics) or Node Exporter (CPU and disk metrics)
- **Broker Log Delivery**
  - Delivery to CloudWatch Logs
  - Delivery to Amazon S3
  - Delivery to Kinesis Data Firehose

# Storage

# AWS S3 Overview - Buckets



- Amazon S3 allows people to store objects (files) in “buckets” (directories)
- Buckets must have a **globally unique name**
- Buckets are defined at the region level
- Naming convention
  - No uppercase
  - No underscore
  - 3-63 characters long
  - Not an IP
  - Must start with lowercase letter or number

# AWS S3 Overview - Objects

- Objects (files) have a Key. The key is the **FULL** path:
  - <my\_bucket>/[my\\_file.txt](#)
  - <my\_bucket>/[my\\_folder1/another\\_folder/my\\_file.txt](#)
- There's no concept of "directories" within buckets (although the UI will trick you to think otherwise)
- Just keys with very long names that contain slashes ("/")
- Object Values are the content of the body:
  - Max Size is 5TB
  - If uploading more than 5GB, must use "multi-part upload"
- Metadata (list of text key / value pairs – system or user metadata)
- Tags (Unicode key / value pair – up to 10) – useful for security / lifecycle
- Version ID (if versioning is enabled)



# AWS S3 - Consistency Model

- **Read after write consistency for PUTS of new objects**
  - As soon as an object is written, we can retrieve it  
ex: (PUT 200 -> GET 200)
  - This is true, **except** if we did a GET before to see if the object existed  
ex: (GET 404 -> PUT 200 -> GET 404) – eventually consistent
- **Eventual Consistency for DELETES and PUTS of existing objects**
  - If we read an object after updating, we might get the older version  
ex: (PUT 200 -> PUT 200 -> GET 200 (might be older version))
  - If we delete an object, we might still be able to retrieve it for a short time  
ex: (DELETE 200 -> GET 200)

# S3 Storage Classes

- Amazon S3 Standard - General Purpose
- Amazon S3 Standard-Infrequent Access (IA)
- Amazon S3 One Zone-Infrequent Access
- Amazon S3 Intelligent Tiering
- Amazon Glacier
- Amazon Glacier Deep Archive
  
- Amazon S3 Reduced Redundancy Storage (deprecated - omitted)

# S3 Standard – General Purpose

- High durability (99.99999999%) of objects across multiple AZ
- If you store 10,000,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000 years
- 99.99% Availability over a given year
- Sustain 2 concurrent facility failures
- Use Cases: Big Data analytics, mobile & gaming applications, content distribution...

# S3 Standard – Infrequent Access (IA)

- Suitable for data that is less frequently accessed, but requires rapid access when needed
- High durability (99.99999999%) of objects across multiple AZs
- 99.9% Availability
- Low cost compared to Amazon S3 Standard
- Sustain 2 concurrent facility failures
- Use Cases: As a data store for disaster recovery, backups...

# S3 One Zone - Infrequent Access (IA)

- Same as IA but data is stored in a single AZ
- High durability (99.99999999%) of objects in a single AZ; data lost when AZ is destroyed
- 99.5% Availability
- Low latency and high throughput performance
- Supports SSL for data at transit and encryption at rest
- Low cost compared to IA (by 20%)
- Use Cases: Storing secondary backup copies of on-premise data, or storing data you can recreate

# S3 Intelligent Tiering

- Same low latency and high throughput performance of S3 Standard
- Small monthly monitoring and auto-tiering fee
- Automatically moves objects between two access tiers based on changing access patterns
- Designed for durability of 99.99999999% of objects across multiple Availability Zones
- Resilient against events that impact an entire Availability Zone
- Designed for 99.9% availability over a given year

# Amazon Glacier

- Low cost object storage meant for archiving / backup
- Data is retained for the longer term (10s of years)
- Alternative to on-premise magnetic tape storage
- Average annual durability is 99.99999999%
- Cost per storage per month (\$0.004 / GB) + retrieval cost
- Each item in Glacier is called “**Archive**” (up to 40TB)
- Archives are stored in “**Vaults**”

# Amazon Glacier & Glacier Deep Archive

- Amazon Glacier – 3 retrieval options:
  - Expedited (1 to 5 minutes)
  - Standard (3 to 5 hours)
  - Bulk (5 to 12 hours)
  - Minimum storage duration of 90 days
- Amazon Glacier Deep Archive – for long term storage – cheaper:
  - Standard (12 hours)
  - Bulk (48 hours)
  - Minimum storage duration of 180 days

# S3 Storage Classes Comparison

	S3 Standard	S3 Intelligent-Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
<b>Designed for durability</b>	99.999999999% (11 9's)					
<b>Designed for availability</b>	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
<b>Availability SLA</b>	99.9%	99%	99%	99%	99.9%	99.9%
<b>Availability Zones</b>	≥3	≥3	≥3	1	≥3	≥3
<b>Minimum capacity charge per object</b>	N/A	N/A	128KB	128KB	40KB	40KB
<b>Minimum storage duration charge</b>	N/A	30 days	30 days	30 days	90 days	180 days
<b>Retrieval fee</b>	N/A	N/A	per GB retrieved	per GB retrieved	per GB retrieved	per GB retrieved

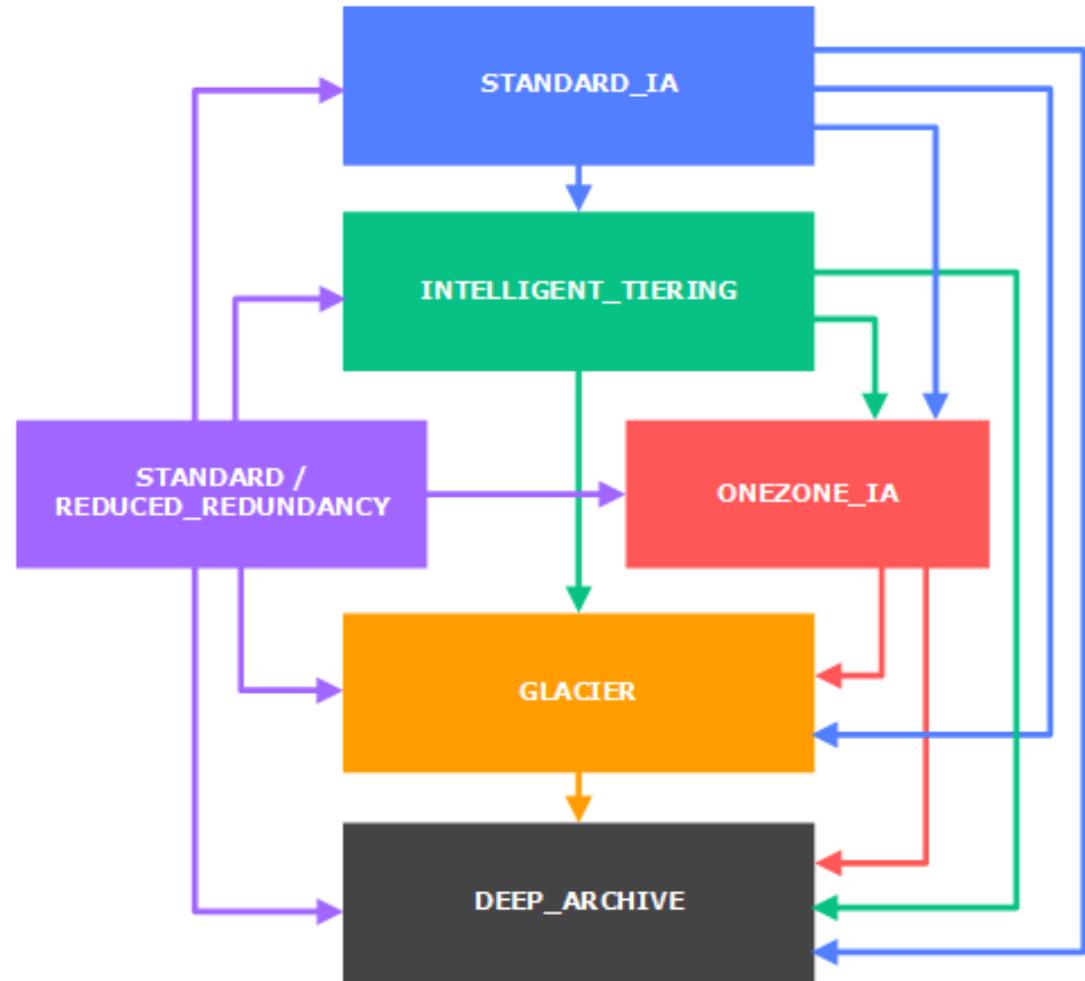
# S3 Storage Classes – Price Comparison

## Example us-east-2

	S3 Standard	S3 Intelligent-Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
<b>Storage Cost (per GB per month)</b>	\$0.023	\$0.0125 - \$0.023	\$0.0125	\$0.01	\$0.004 Minimum 90 days	\$0.00099 Minimum 180 days
<b>Retrieval Cost (per 1000 requests)</b>	GET \$0.0004	GET \$0.0004	GET \$0.001	GET \$0.001	GET \$0.0004 + Expedited - \$10.00 Standard - \$0.05 Bulk - \$0.025	GET \$0.0004 + Standard - \$0.10 Bulk - \$0.025
<b>Time to retrieve</b>	instantaneous	Instantaneous	Instantaneous	Instantaneous	Expedited (1 to 5 minutes) Standard (3 to 5 hours) Bulk (5 to 12 hours)	Standard (12 hours) Bulk (48 hours)
<b>Monitoring Cost (per 1000 objects)</b>		\$0.0025				

# S3 – Moving between storage classes

- You can transition objects between storage classes
- For infrequently accessed object, move them to STANDARD\_IA
- For archive objects you don't need in real-time, GLACIER or DEEP\_ARCHIVE
- Moving objects can be automated using a **lifecycle configuration**



# S3 Lifecycle Rules

- **Transition actions:** It defines when objects are transitioned to another storage class.
  - Move objects to Standard IA class 60 days after creation
  - Move to Glacier for archiving after 6 months
- **Expiration actions:** configure objects to expire (delete) after some time
  - Access log files can be set to delete after a 365 days
  - **Can be used to delete old versions of files (if versioning is enabled)**
  - Can be used to delete incomplete multi-part uploads
- Rules can be created for a certain prefix (ex - s3://mybucket/mp3/\*)
- Rules can be created for certain objects tags (ex - Department: Finance)

# S3 Lifecycle Rules – Scenario 1

- Your application on EC2 creates images thumbnails after profile photos are uploaded to Amazon S3. These thumbnails can be easily recreated, and only need to be kept for 45 days. The source images should be able to be immediately retrieved for these 45 days, and afterwards, the user can wait up to 6 hours. How would you design this?
- S3 source images can be on STANDARD, with a lifecycle configuration to transition them to GLACIER after 45 days.
- S3 thumbnails can be on ONEZONE\_IA, with a lifecycle configuration to expire them (delete them) after 45 days.

# S3 Lifecycle Rules – Scenario 2

- A rule in your company states that you should be able to recover your deleted S3 objects immediately for 15 days, although this may happen rarely. After this time, and for up to 365 days, deleted objects should be recoverable within 48 hours.
- You need to enable S3 versioning in order to have object versions, so that “deleted objects” are in fact hidden by a “delete marker” and can be recovered
- You can transition these “noncurrent versions” of the object to S3\_IA
- You can transition afterwards these “noncurrent versions” to DEEP\_ARCHIVE

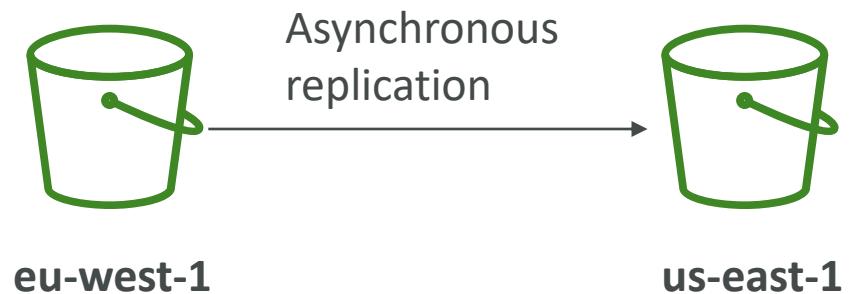
# AWS S3 - Versioning



- You can version your files in AWS S3
- It is enabled at the **bucket level**
- Same key overwrite will increment the “version”: 1, 2, 3....
- It is best practice to version your buckets
  - Protect against unintended deletes (ability to restore a version)
  - Easy roll back to previous version
- Any file that is not versioned prior to enabling versioning will have version “null”
- You can “suspend” versioning

# S3 Cross Region Replication

- Must enable versioning (source and destination)
- Buckets must be in different AWS regions
- Can be in different accounts
- Copying is asynchronous
- Must give proper IAM permissions to S3
- Use cases: compliance, lower latency access, replication across accounts



# AWS S3 – ETag (Entity Tag)

- How do you verify if a file has already been uploaded to S3?
- Names work, but how are you sure the file is **exactly** the same?
- For this, you can use AWS ETags:
  - For simple uploads (less than 5GB), it's the **MD5** hash
  - For multi-part uploads, it's more complicated, no need to know the algorithm
- Using ETag, we can ensure **integrity** of files

# AWS S3 Performance – Key Names

## Historic fact and current exam

- When you had > 100 TPS (transaction per second), S3 performance could degrade
- Behind the scene, each object goes to an S3 partition and for the best performance, we want the highest partition distribution
- In the exam, and historically, it was recommended to have random characters in front of your key name to optimise performance:
  - <my\_bucket>/5r4d\_my\_folder/my\_file1.txt
  - <my\_bucket>/a91e\_my\_folder/my\_file2.txt
  - ...
- It was recommended **never to use dates to prefix keys**:
  - <my\_bucket>/2018\_09\_09\_my\_folder/my\_file1.txt
  - <my\_bucket>/2018\_09\_10\_my\_folder/my\_file2.txt

# AWS S3 Performance – Key Names

## Current performance (not yet exam)

- <https://aws.amazon.com/about-aws/whats-new/2018/07/amazon-s3-announces-increased-request-rate-performance/>
- As of July 17<sup>th</sup> 2018, we can scale up to 3500 RPS for PUT and 5500 RPS for GET for EACH PREFIX
- “This S3 request rate performance increase removes any previous guidance to randomize object prefixes to achieve faster performance”
- It’s a “good to know”, until the exam gets updated ☺

# AWS S3 Performance

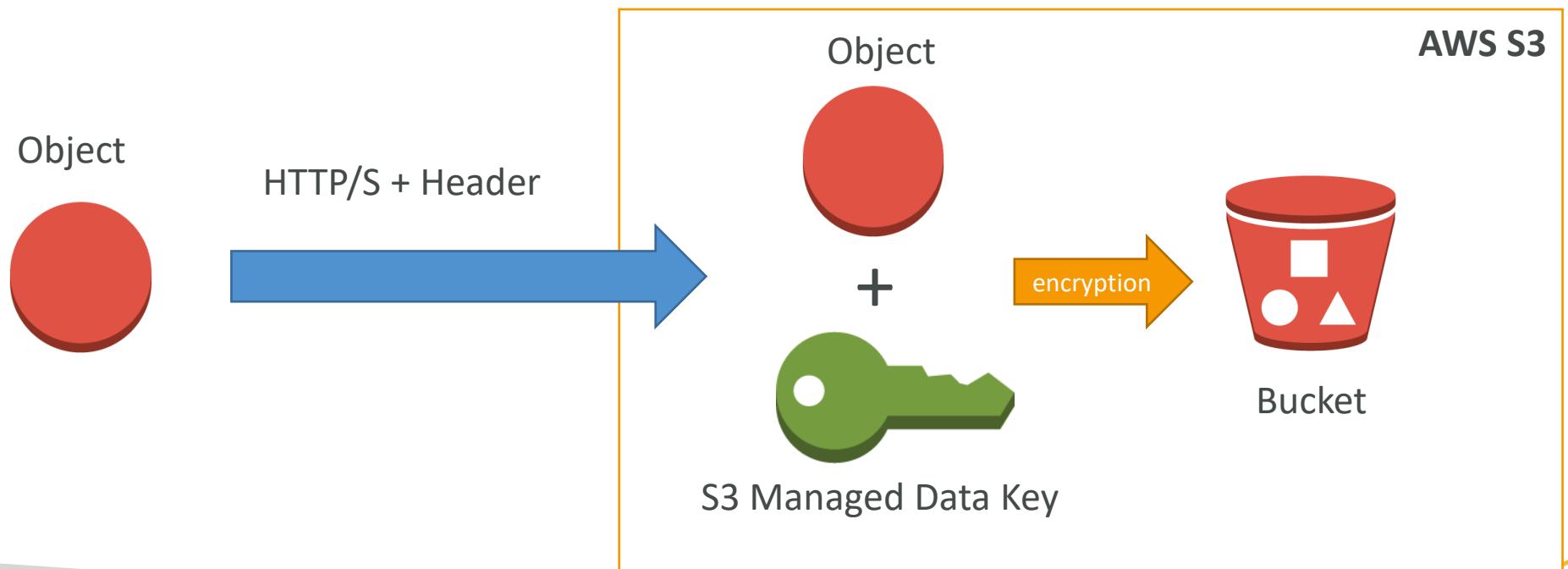
- Faster upload of large objects (>5GB), use multipart upload:
  - parallelizes PUTs for greater throughput
  - maximize your network bandwidth
  - decrease time to retry in case a part fails
- Use CloudFront to cache S3 objects around the world (improves reads)
- S3 Transfer Acceleration (uses edge locations) – just need to change the endpoint you write to, not the code.
- If using SSE-KMS encryption, you may be limited to your AWS limits for KMS usage (~100s – 1000s downloads / uploads per second)

# S3 Encryption for Objects

- There are 4 methods of encrypting objects in S3
  - SSE-S3: encrypts S3 objects using keys handled & managed by AWS
  - SSE-KMS: leverage AWS Key Management Service to manage encryption keys
  - SSE-C: when you want to manage your own encryption keys
  - Client Side Encryption
- It's important to understand which ones are adapted to which situation for the exam

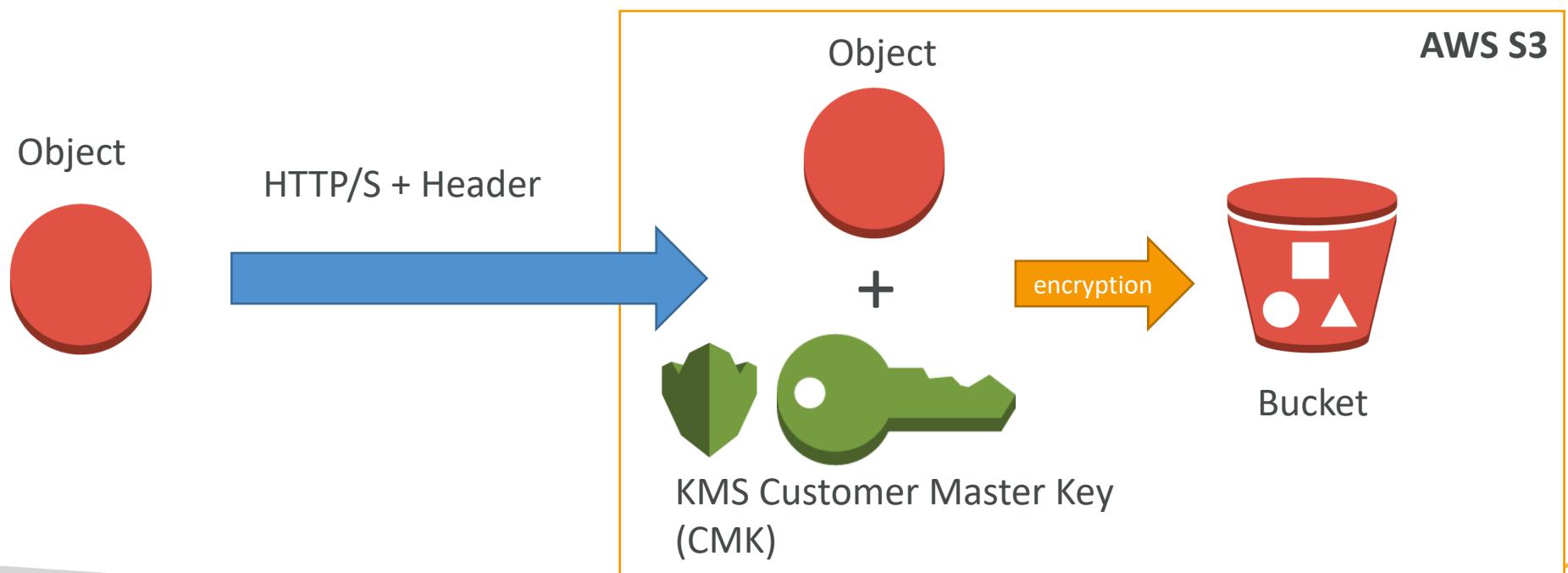
# SSE-S3

- SSE-S3: encryption using keys handled & managed by AWS S3
- Object is encrypted server side
- AES-256 encryption type
- Must set header: “**x-amz-server-side-encryption**”: "AES256"



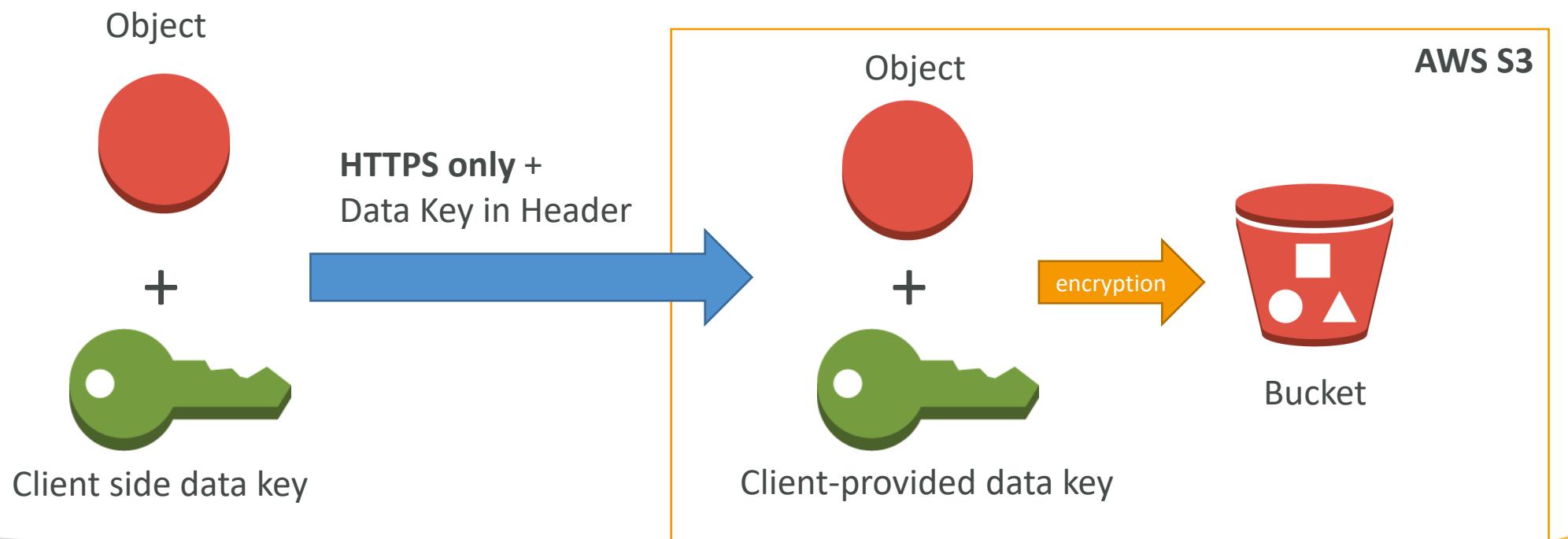
# SSE-KMS

- SSE-KMS: encryption using keys handled & managed by KMS
- KMS Advantages: user control + audit trail
- Object is encrypted server side
- Must set header: “**x-amz-server-side-encryption**”: “aws:kms”



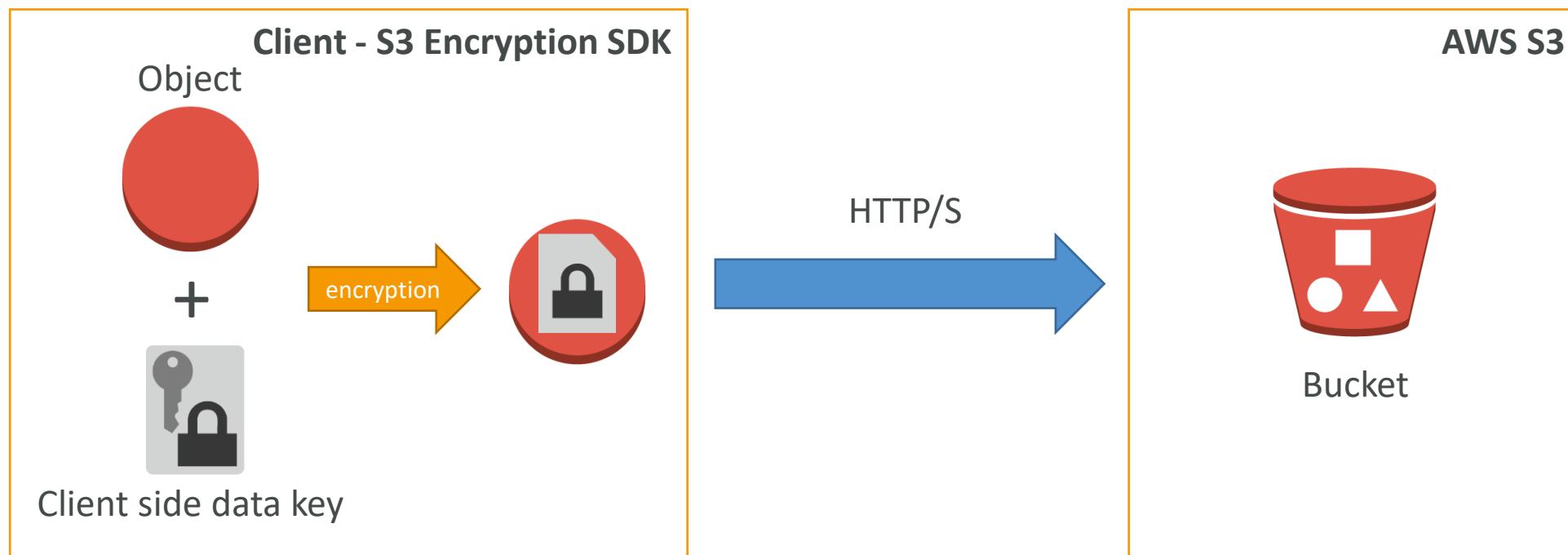
# SSE-C

- SSE-C: server-side encryption using data keys fully managed by the customer outside of AWS
- Amazon S3 does not store the encryption key you provide
- **HTTPS must be used**
- Encryption key must provided in HTTP headers, for every HTTP request made



# Client Side Encryption

- Client library such as the Amazon S3 Encryption Client
- Clients must encrypt data themselves before sending to S3
- Clients must decrypt data themselves when retrieving from S3
- Customer fully manages the keys and encryption cycle



# Encryption in transit (SSL)

- AWS S3 exposes:
  - HTTP endpoint: non encrypted
  - HTTPS endpoint: encryption in flight
- You're free to use the endpoint you want, but HTTPS is recommended
- HTTPS is mandatory for SSE-C
- Encryption in flight is also called SSL / TLS

# S3 CORS (Cross-Origin Resource Sharing)

- If you request data from another website, you need to enable CORS
- Cross Origin Resource Sharing allows you to limit the number of websites that can request your files in S3 (and limit your costs)
- It's a popular exam question



# S3 Access Logs

- For audit purpose, you may want to log all access to S3 buckets
- Any request made to S3, from any account, authorized or denied, will be logged into another S3 bucket
- That data can be analyzed using data analysis tools...
- Or Amazon Athena as we'll see later in this course!
- The log format is at:  
<https://docs.aws.amazon.com/AmazonS3/latest/dev/LoggingFormat.html>



# S3 Security

- User based
  - IAM policies - which API calls should be allowed for a specific user from IAM console
- Resource Based
  - Bucket Policies - bucket wide rules from the S3 console - allows cross account
  - Object Access Control List (ACL) – finer grain
  - Bucket Access Control List (ACL) – less common

# S3 Bucket Policies

- JSON based policies
  - Resources: buckets and objects
  - Actions: Set of API to Allow or Deny
  - Effect: Allow / Deny
  - Principal: The account or user to apply the policy to
- Use S3 bucket for policy to:
  - Grant public access to the bucket
  - Force objects to be encrypted at upload
  - Grant access to another account (Cross Account)

# S3 Default Encryption vs Bucket Policies

- The old way to enable default encryption was to use a bucket policy and refuse any HTTP command without the proper headers:

```
{  
    "Version": "2012-10-17",  
    "Id": "PutObjPolicy",  
    "Statement": [  
        {  
            "Sid": "DenyIncorrectEncryptionHeader",  
            "Effect": "Deny",  
            "Principal": "*",  
            "Action": "s3:PutObject",  
            "Resource": "arn:aws:s3:::<bucket_name>/*",  
            "Condition": {  
                "StringNotEquals": {  
                    "s3:x-amz-server-side-encryption": "AES256"  
                }  
            }  
        }  
    ],  
}.
```

```
{  
    "Sid": "DenyUnEncryptedObjectUploads",  
    "Effect": "Deny",  
    "Principal": "*",  
    "Action": "s3:PutObject",  
    "Resource": "arn:aws:s3:::<bucket_name>/*",  
    "Condition": {  
        "Null": {  
            "s3:x-amz-server-side-encryption": true  
        }  
    }  
}
```

- The new way is to use the “default encryption” option in S3
- Note: Bucket Policies are evaluated before “default encryption”

# S3 Security - Other

- Networking:
  - Supports VPC Endpoints (for instances in VPC without www internet)
- Logging and Audit:
  - S3 access logs can be stored in other S3 bucket
  - API calls can be logged in AWS CloudTrail
- User Security:
  - MFA (multi factor authentication) can be required in versioned buckets to delete objects
  - Signed URLs: URLs that are valid only for a limited time (ex: premium video service for logged in users)

# Glacier



- Low cost object storage meant for archiving / backup
- Data is retained for the longer term (10s of years)
- Alternative to on-premise magnetic tape storage
- Average annual durability is 99.99999999%
- Cost per storage per month (\$0.004 / GB) + retrieval cost
- Each item in Glacier is called “**Archive**” (up to 40TB)
- Archives are stored in ”**Vaults**”
- Exam tip: archival from S3 after XXX days => use Glacier

# Glacier Operations

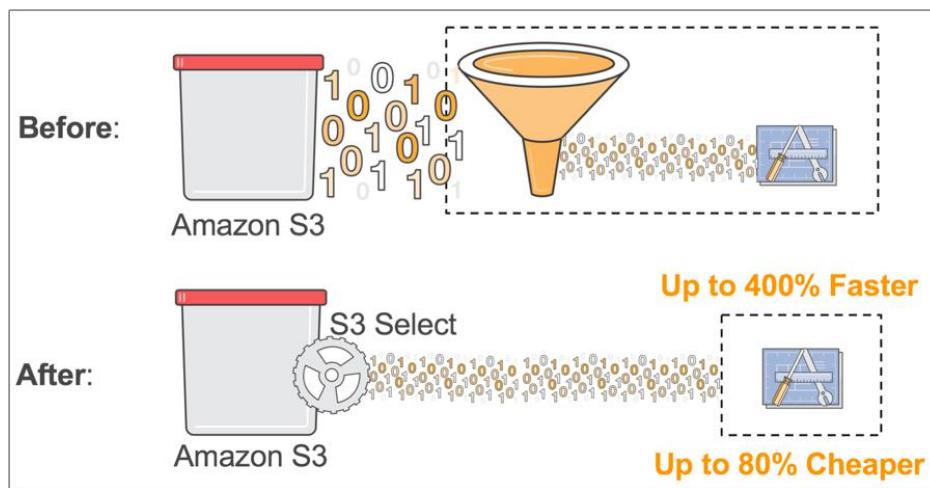
- Restore links have an expiry date
- 3 retrieval options:
  - Expedited (1 to 5 minutes retrieval) – \$0.03 per GB and \$0.01 per request
  - Standard (3 to 5 hours) - \$0.01 per GB and 0.05 per 1000 requests
  - Bulk (5 to 12 hours) - \$0.0025 per GB and \$0.025 per 1000 requests

# Glacier - Vault Policies & Vault Lock

- Vault is a collection of archives
- Each Vault has:
  - ONE vault access policy
  - ONE vault lock policy
- Vault Policies are written in JSON
- Vault Access Policy is similar to bucket policy (restrict user / account permissions)
- Vault Lock Policy is a policy you lock, for regulatory and compliance requirements.
  - The policy is immutable, **it can never be changed (that's why it's call LOCK)**
  - Example 1: forbid deleting an archive if less than 1 year old
  - Example 2: implement WORM policy (write once read many)

# S3 Select & Glacier Select

- Retrieve less data using SQL by performing **server side filtering**
- Can filter by rows & columns (simple SQL statements)
- Less network transfer, less CPU cost client-side

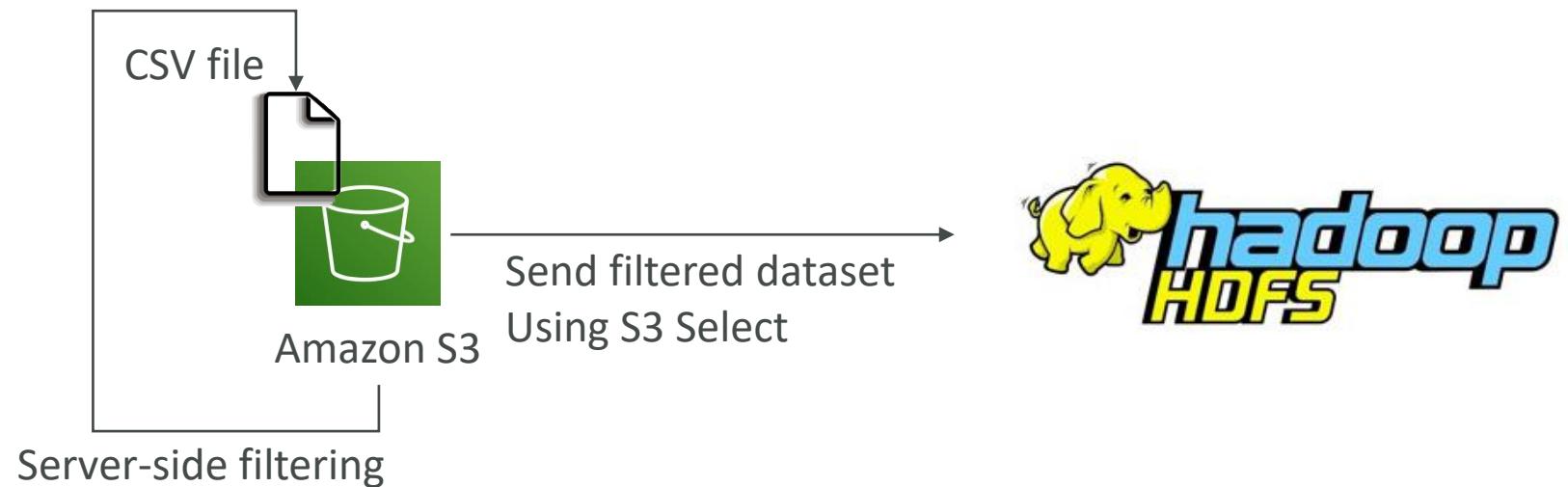


<https://aws.amazon.com/blogs/aws/s3-glacier-select/>



# S3 Select with Hadoop

- Transfer some data from S3 before analyzing it with your cluster
- Load less data into Hadoop, save network costs, transfer the data faster



# DynamoDB



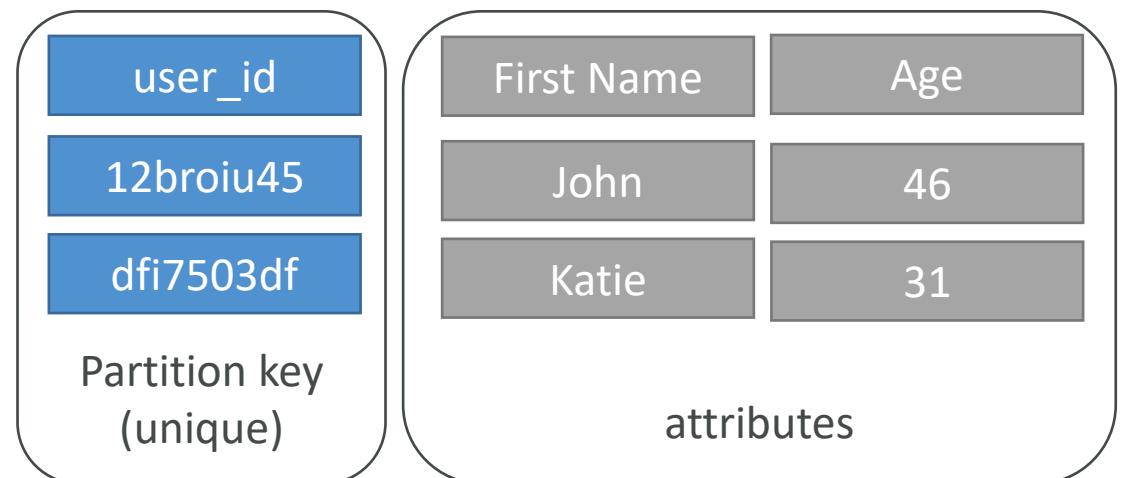
- Fully Managed, Highly available with replication across 3 AZ
- NoSQL database - not a relational database
- Scales to massive workloads, distributed database
- Millions of requests per seconds, trillions of row, 100s of TB of storage
- Fast and consistent in performance (low latency on retrieval)
- Integrated with IAM for security, authorization and administration
- Enables event driven programming with DynamoDB Streams
- Low cost and auto scaling capabilities

# DynamoDB - Basics

- DynamoDB is made of **tables**
- Each table has a **primary key** (must be decided at creation time)
- Each table can have an infinite number of items (= rows)
- Each item has **attributes** (can be added over time – can be null)
- Maximum size of a item is **400KB**
- Data types supported are:
  - Scalar Types: String, Number, Binary, Boolean, Null
  - Document Types: List, Map
  - Set Types: String Set, Number Set, Binary Set

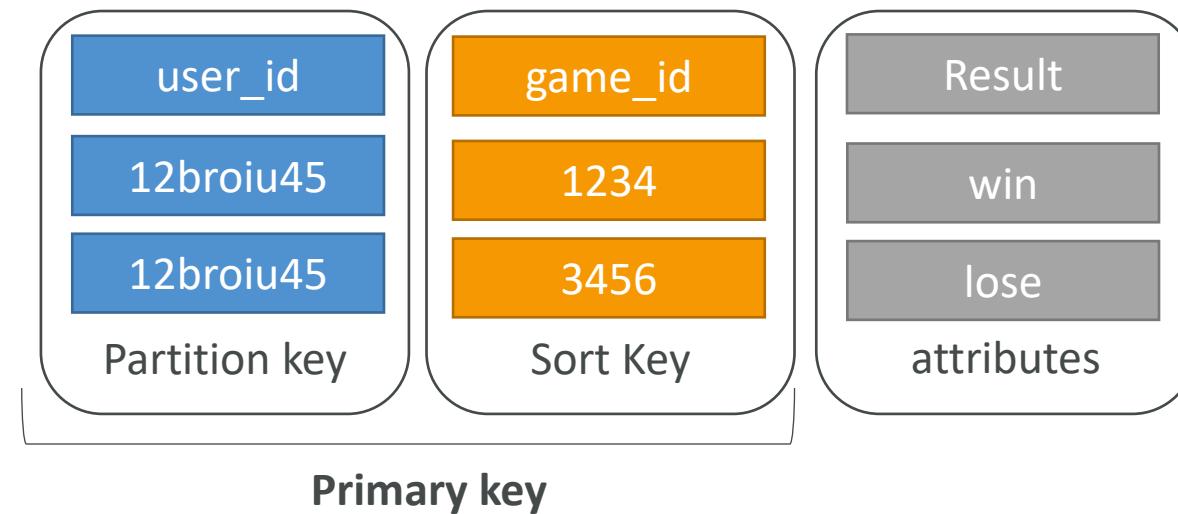
# DynamoDB – Primary Keys

- **Option 1: Partition key only (HASH)**
- Partition key must be unique for each item
- Partition key must be “diverse” so that the data is distributed
- Example: user\_id for a users table



# DynamoDB – Primary Keys

- **Option 2: Partition key + Sort Key**
- The combination must be unique
- Data is grouped by partition key
- Sort key == range key
- Example: users-games table
  - user\_id for the partition key
  - game\_id for the sort key



# DynamoDB – Partition Keys exercise

- We're building a movie database
- What is the best partition key to maximize data distribution?
  - movie\_id
  - producer\_name
  - leader\_actor\_name
  - movie\_language
- movie\_id has the highest cardinality so it's a good candidate
- moving\_language doesn't take many values and may be skewed towards English so it's not a great partition key

# DynamoDB in Big Data

- Common use cases include:
  - Mobile apps
  - Gaming
  - Digital ad serving
  - Live voting
  - Audience interaction for live events
  - Sensor networks
  - Log ingestion
  - Access control for web-based content
  - Metadata storage for Amazon S3 objects
  - E-commerce shopping carts
  - Web session management
- Anti Pattern
  - Prewritten application tied to a traditional relational database: use RDS instead
  - Joins or complex transactions
  - Binary Large Object (BLOB) data: store data in S3 & metadata in DynamoDB
  - Large data with low I/O rate: use S3 instead

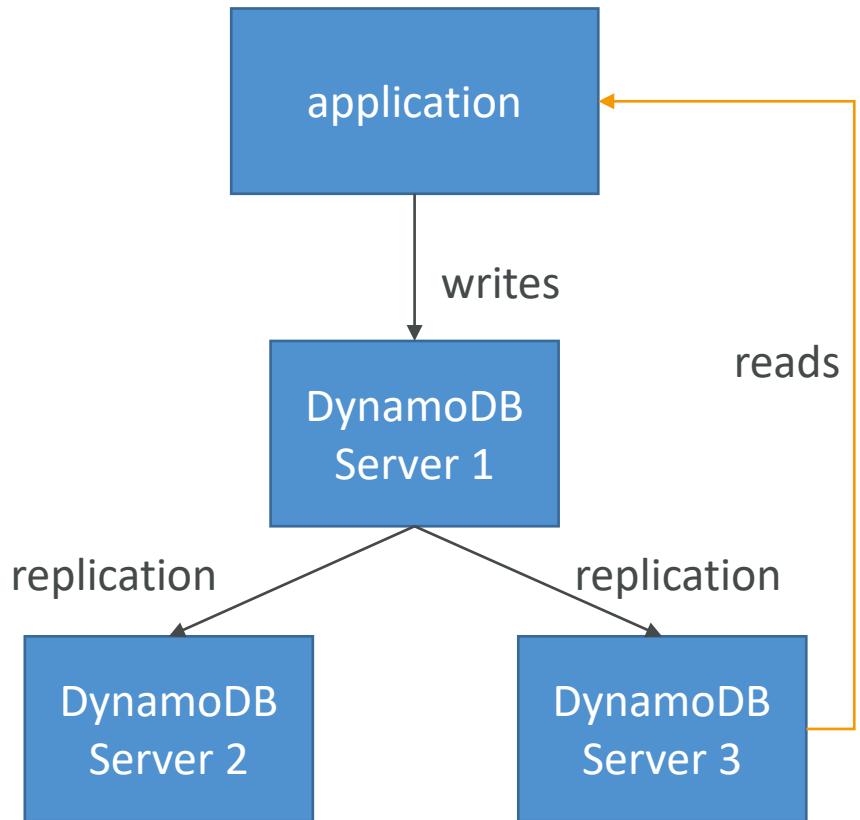
# DynamoDB – Provisioned Throughput

- Table must have provisioned read and write capacity units
- **Read Capacity Units (RCU)**: throughput for reads
- **Write Capacity Units (WCU)**: throughput for writes
- Option to setup auto-scaling of throughput to meet demand
- Throughput can be exceeded temporarily using “burst credit”
- If burst credit are empty, you’ll get a “ProvisionedThroughputException”.
- It’s then advised to do an exponential back-off retry

# DynamoDB – Write Capacity Units

- One *write capacity unit* represents one write per second for an item up to 1 KB in size.
- If the items are larger than 1 KB, more WCU are consumed
- **Example 1:** we write 10 objects per seconds of 2 KB each.
  - We need  $2 * 10 = 20$  WCU
- **Example 2:** we write 6 objects per second of 4.5 KB each
  - We need  $6 * 5 = 30$  WCU (4.5 gets rounded to the upper KB)
- **Example 3:** we write 120 objects per minute of 2 KB each
  - We need  $120 / 60 * 2 = 4$  WCU

# Strongly Consistent Read vs Eventually Consistent Read



- **Eventually Consistent Read:** If we read just after a write, it's possible we'll get unexpected response because of replication
- **Strongly Consistent Read:** If we read just after a write, we will get the correct data
- **By default:** DynamoDB uses Eventually Consistent Reads, but GetItem, Query & Scan provide a "ConsistentRead" parameter you can set to True

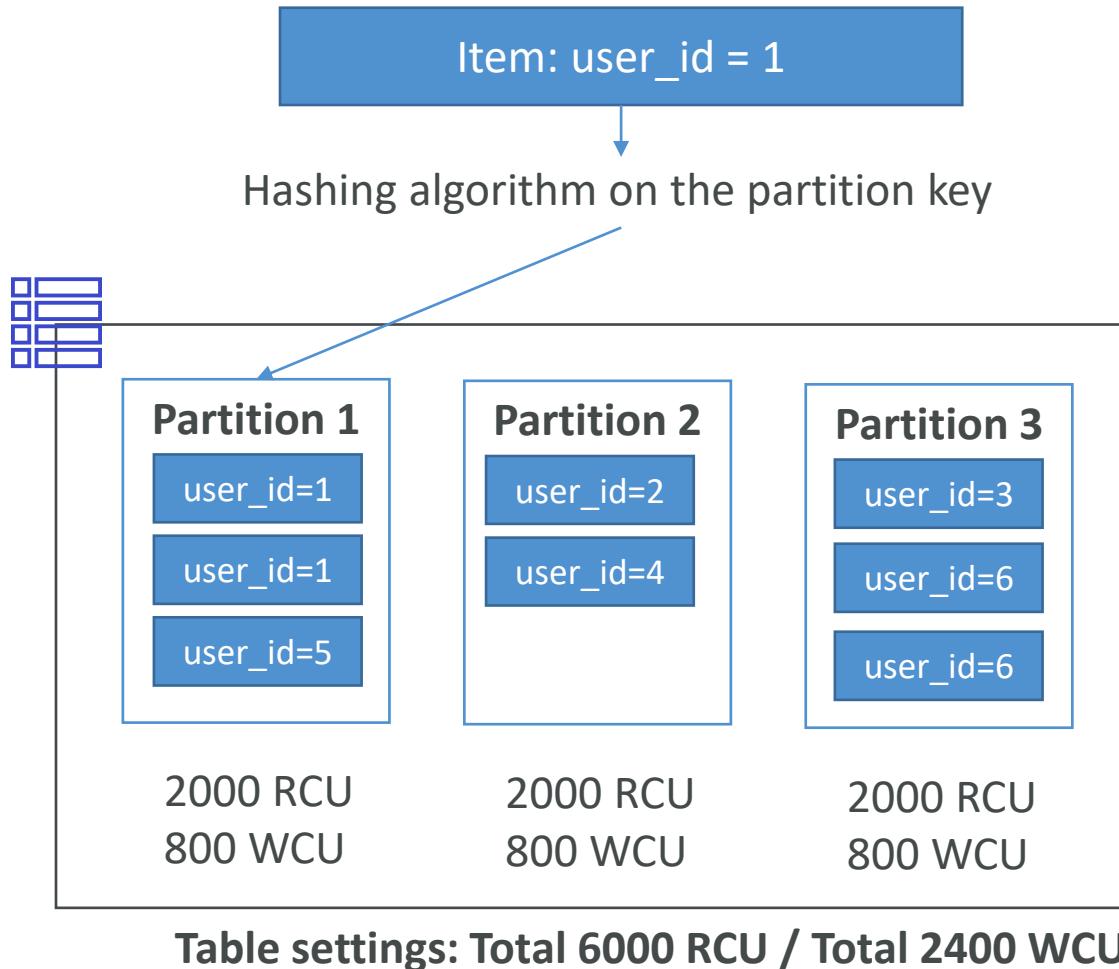
# DynamoDB – Read Capacity Units

- One *read capacity unit* represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size.
- If the items are larger than 4 KB, more RCU are consumed
- **Example 1:** 10 strongly consistent reads per seconds of 4 KB each
  - We need  $10 * 4 \text{ KB} / 4 \text{ KB} = 10 \text{ RCU}$
- **Example 2:** 16 eventually consistent reads per seconds of 12 KB each
  - We need  $(16 / 2) * (12 / 4) = 24 \text{ RCU}$
- **Example 3:** 10 strongly consistent reads per seconds of 6 KB each
  - We need  $10 * 8 \text{ KB} / 4 = 20 \text{ RCU}$  (we have to round up 6 KB to 8 KB)

# DynamoDB - Throttling

- If we exceed our RCU or WCU, we get **ProvisionedThroughputExceededExceptions**
- Reasons:
  - Hot keys / partitions: one partition key is being read too many times (popular item for ex)
  - Very large items: remember RCU and WCU depends on size of items
- Solutions:
  - Exponential back-off when exception is encountered (already in SDK)
  - Distribute partition keys as much as possible
  - If RCU issue, we can use DynamoDB Accelerator (DAX)

# DynamoDB – Partitions Internal



- You start with one partition
- Each partition:
  - Max of 3000 RCU / 1000 WCU
  - Max of 10GB
- To compute the number of partitions:
  - By capacity:  $(\text{TOTAL RCU} / 3000) + (\text{TOTAL WCU} / 1000)$
  - By size:  $\text{Total Size} / 10 \text{ GB}$
  - Total partitions =  $\text{CEILING}(\text{MAX}(\text{Capacity}, \text{Size}))$
- **WCU and RCU are spread evenly between partitions**

# DynamoDB – Writing Data

- **PutItem** - Write data to DynamoDB (create data or full replace)
  - Consumes WCU
- **UpdateItem** – Update data in DynamoDB (partial update of attributes)
  - Possibility to use Atomic Counters and increase them
- **Conditional Writes:**
  - Accept a write / update only if conditions are respected, otherwise reject
  - Helps with concurrent access to items
  - No performance impact

# DynamoDB – Deleting Data

- **DeleteItem**
  - Delete an individual row
  - Ability to perform a conditional delete
- **DeleteTable**
  - Delete a whole table and all its items
  - Much quicker deletion than calling DeleteItem on all items

# DynamoDB – Batching Writes

- **BatchWriteItem**
  - Up to 25 **PutItem** and / or **DeleteItem** in one call
  - Up to 16 MB of data written
  - Up to 400 KB of data per item
- Batching allows you to save in latency by reducing the number of API calls done against DynamoDB
- Operations are done in parallel for better efficiency
- It's possible for part of a batch to fail, in which case we have the try the failed items (using exponential back-off algorithm)

# DynamoDB – Reading Data

- **GetItem:**

- Read based on Primary key
- Primary Key = HASH or HASH-RANGE
- Eventually consistent read by default
- Option to use strongly consistent reads (more RCU - might take longer)
- **ProjectionExpression** can be specified to include only certain attributes

- **BatchGetItem:**

- Up to 100 items
- Up to 16 MB of data
- Items are retrieved in parallel to minimize latency

# DynamoDB – Query

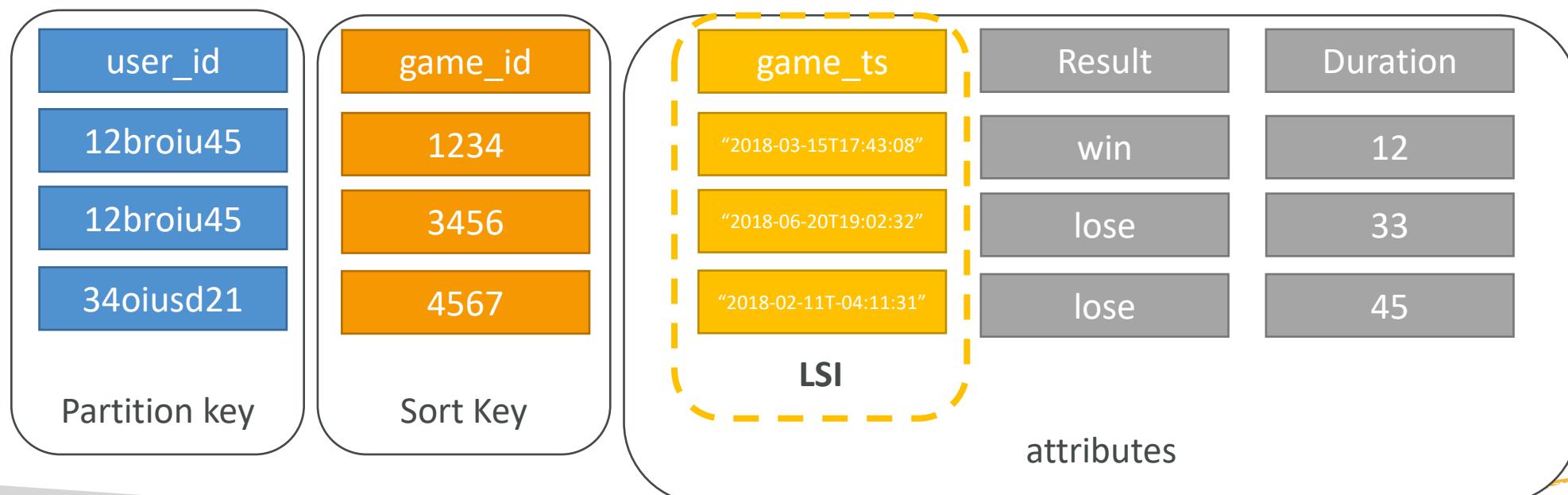
- **Query** returns items based on:
  - PartitionKey value (**must be = operator**)
  - SortKey value (=, <, <=, >, >=, Between, Begin) – optional
  - FilterExpression to further filter (client side filtering)
- Returns:
  - Up to 1 MB of data
  - Or number of items specified in **Limit**
- Able to do pagination on the results
- Can query table, a local secondary index, or a global secondary index

# DynamoDB - Scan

- **Scan** the entire table and then filter out data (inefficient)
- Returns up to 1 MB of data – use pagination to keep on reading
- Consumes a lot of RCU
- Limit impact using Limit or reduce the size of the result and pause
- For faster performance, use **parallel scans**:
  - Multiple instances scan multiple partitions at the same time
  - Increases the throughput and RCU consumed
  - Limit the impact of parallel scans just like you would for Scans
- Can use a **ProjectionExpression + FilterExpression** (no change to RCU)

# DynamoDB – LSI (Local Secondary Index)

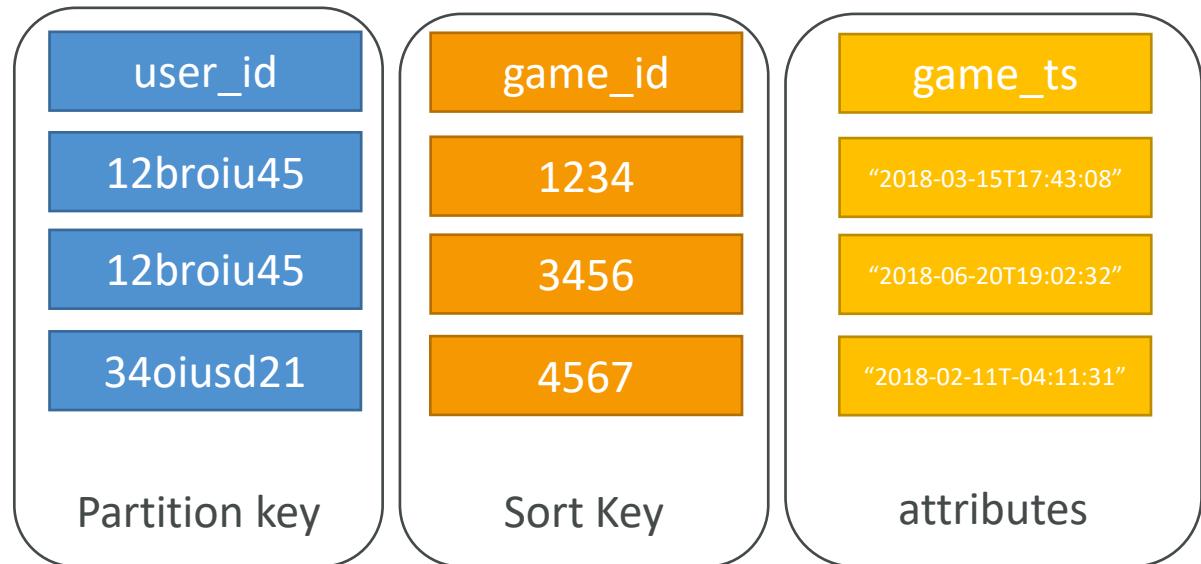
- Alternate range key for your table, **local to the hash key**
- Up to five local secondary indexes per table.
- The sort key consists of exactly one scalar attribute.
- The attribute that you choose must be a scalar String, Number, or Binary
- **LSI must be defined at table creation time**



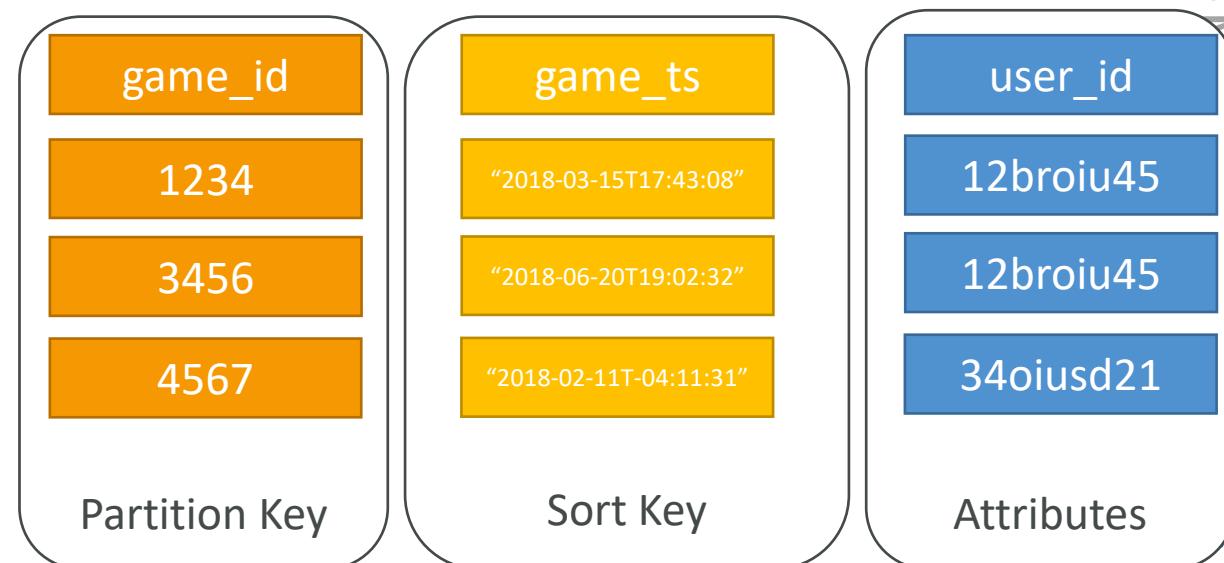
# DynamoDB – GSI (Global Secondary Index)

- To speed up queries on non-key attributes, use a Global Secondary Index
- GSI = partition key + optional sort key
- The index is a new “table” and we can project attributes on it
  - The partition key and sort key of the original table are always projected (KEYS\_ONLY)
  - Can specify extra attributes to project (INCLUDE)
  - Can use all attributes from main table (ALL)
- Must define RCU / WCU for the index
- **Possibility to add / modify GSI (not LSI)**

# DynamoDB – GSI (Global Secondary Index)

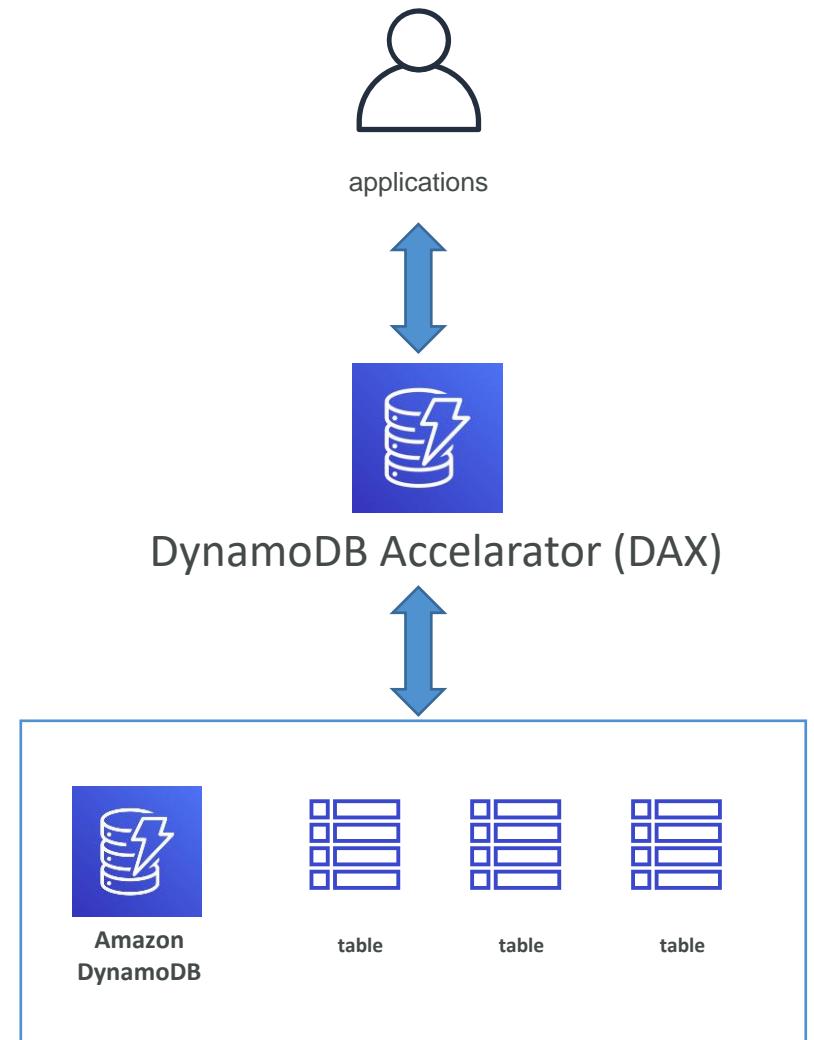


**INDEX – queries by game\_id**



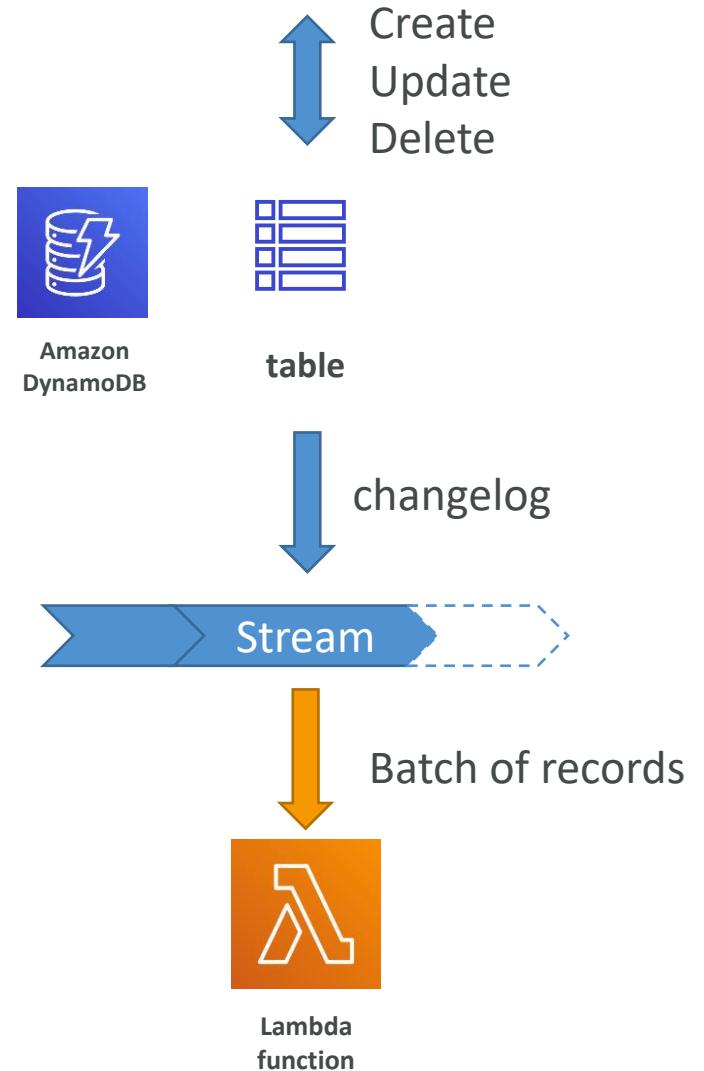
# DynamoDB - DAX

- DAX = DynamoDB Accelerator
- Seamless cache for DynamoDB, no application re-write
- Writes go through DAX to DynamoDB
- Micro second latency for cached reads & queries
- Solves the Hot Key problem (too many reads)
- 5 minutes TTL for cache by default
- Up to 10 nodes in the cluster
- Multi AZ (3 nodes minimum recommended for production)
- Secure (Encryption at rest with KMS, VPC, IAM, CloudTrail...)



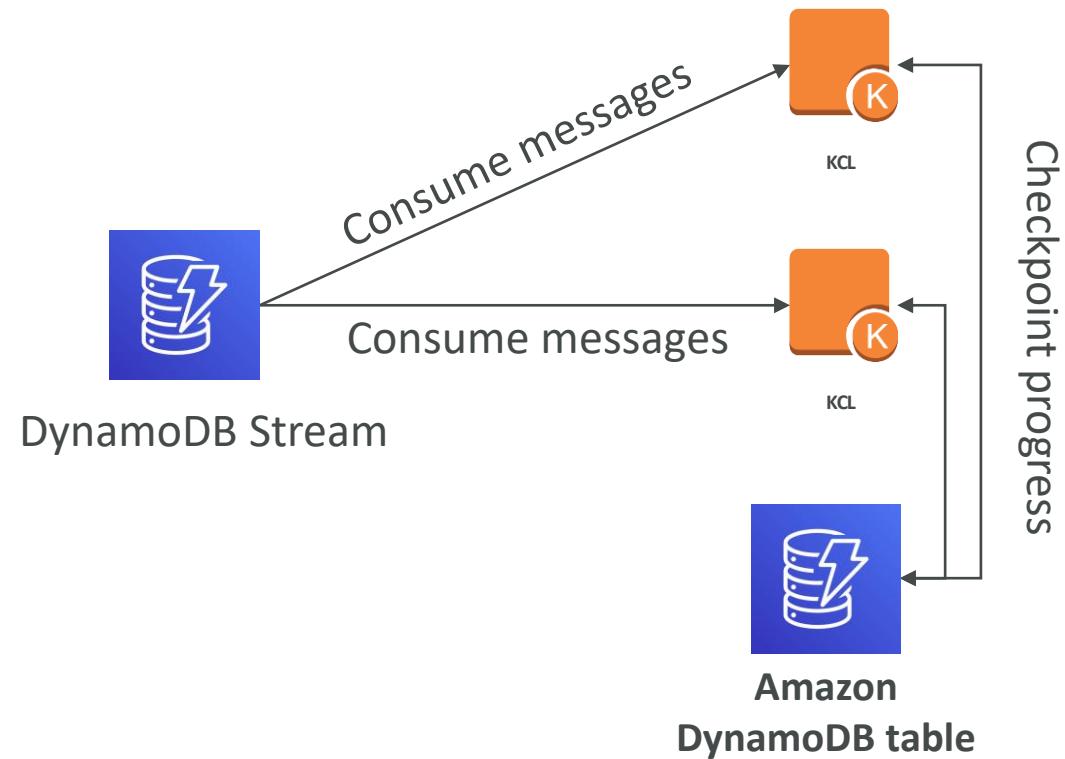
# DynamoDB Streams

- Changes in DynamoDB (Create, Update, Delete) can end up in a DynamoDB Stream
- This stream can be read by AWS Lambda, and we can then do:
  - React to changes in real time (welcome email to new users)
  - Create derivative tables / views
  - Insert into ElasticSearch
- Could implement Cross Region Replication using Streams
- Stream has 24 hours of data retention
- Configurable batch size (up to 1,000 rows, 6 MB)



# DynamoDB Streams Kinesis Adapter

- Use the KCL library to directly consume from DynamoDB Streams
- You just need to add a “Kinesis Adapter” library
- The interface and programming is exactly the same as Kinesis Streams
- That’s the alternative to using AWS Lambda



# DynamoDB TTL (Time to Live)

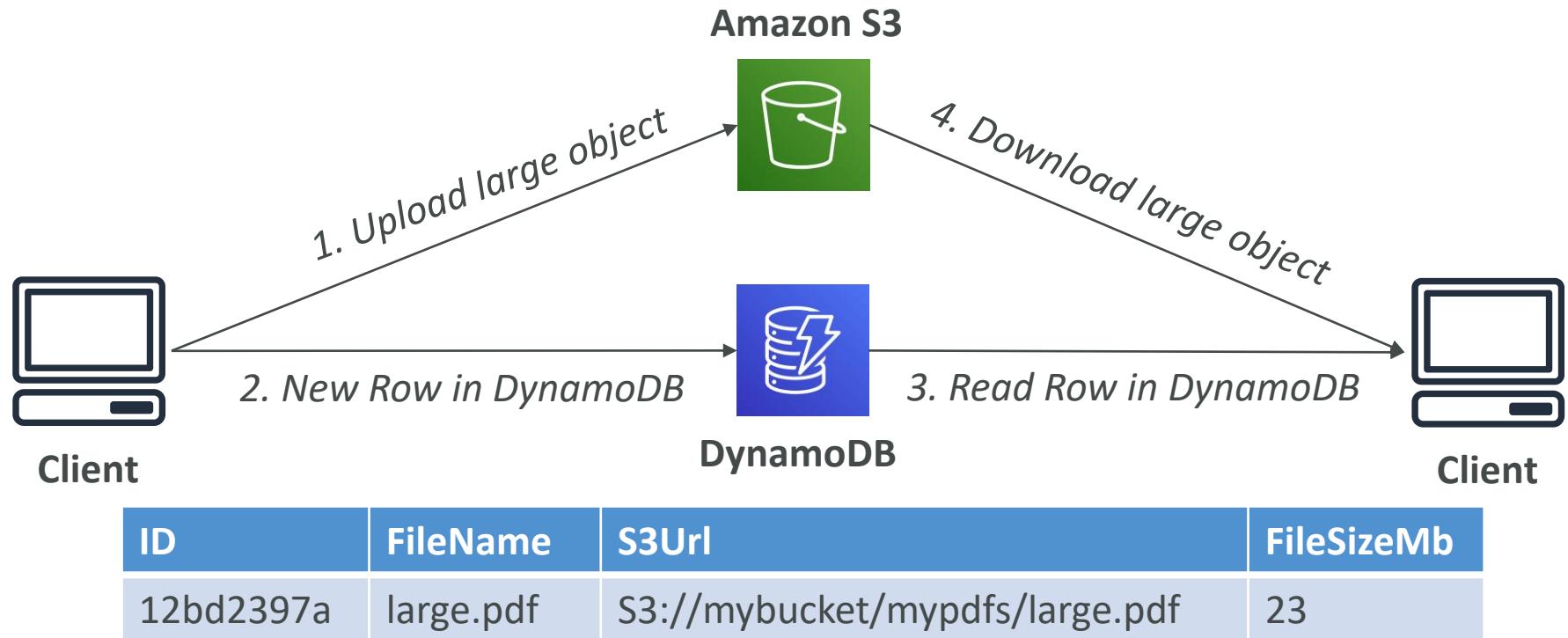
- TTL = automatically delete an item after an expiry date / time
- TTL is provided at no extra cost, deletions do not use WCU / RCU
- TTL is a background task operated by the DynamoDB service itself
- Helps reduce storage and manage the table size over time
- Helps adhere to regulatory norms
- TTL is enabled per row (you define a TTL column, and add a date there)
- DynamoDB typically deletes expired items within 48 hours of expiration
- Deleted items due to TTL are also deleted in GSI / LSI
- DynamoDB Streams can help recover expired items

# DynamoDB – Security & Other Features

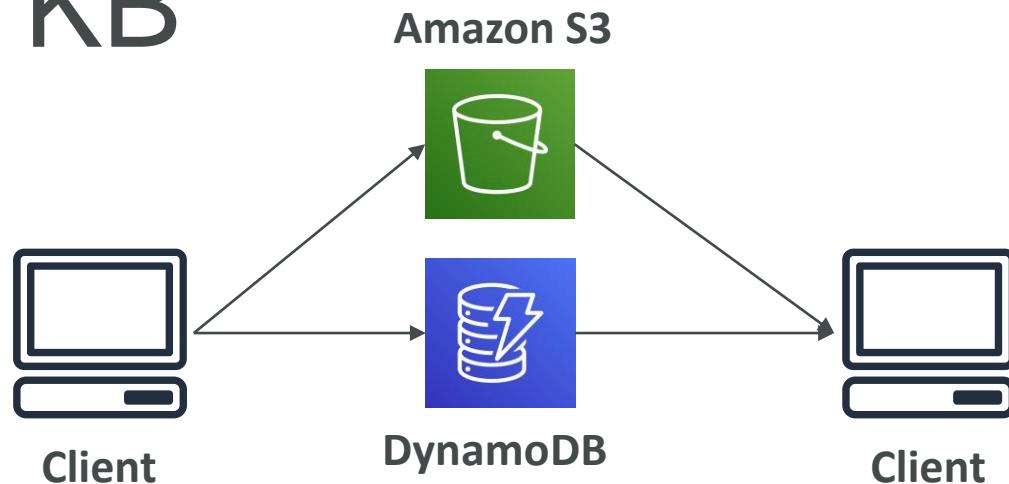
- Security:
  - VPC Endpoints available to access DynamoDB without internet
  - Access fully controlled by IAM
  - Encryption at rest using KMS
  - Encryption in transit using SSL / TLS
- Backup and Restore feature available
  - Point in time restore like RDS
  - No performance impact
- Global Tables
  - Multi region, fully replicated, high performance
- Amazon Database Migration Service (DMS) can be used to migrate to DynamoDB (from Mongo, Oracle, MySQL, S3, etc...)
- You can launch a local DynamoDB on your computer for development purposes

# DynamoDB – Storing large objects

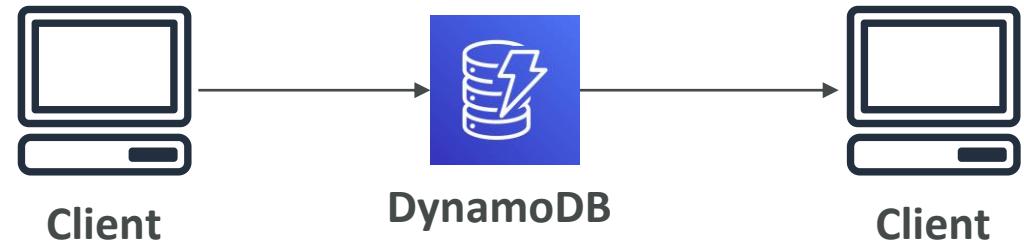
- Max size of an item in DynamoDB = 400 KB
- For large objects, store them in S3 and reference them in DynamoDB



# DynamoDB - Price comparison for 300 KB



- Amazon S3 (300KB of storage)**
  - \$0.0000069 storage per month
  - \$0.0000050 initial PUT
  - \$0.0000004 per GET
- DynamoDB (< 1 KB of storage)**
  - \$0.0006500 for one WCU per month
  - \$0.0001300 for one RCU per month
  - \$0.00000025 storage per month
- Assuming 1 write, 100 reads per month:**
  - \$0.00119215 per month



- DynamoDB (300 KB of storage)**
  - \$0.195 for 300 WCU per month
  - \$0.004940 for 38 RCU per month
  - \$0.000075 storage per month
- Assuming 1 write, 100 reads per month:**
  - Storage is 11x more expensive
  - WCU + RCU are under-used
- Even for items that fit in DynamoDB, if under-used, S3 + DynamoDB is a solution

# AWS ElastiCache Overview



- The same way RDS is to get managed Relational Databases...
- ElastiCache is to get managed Redis or Memcached
- Caches are in-memory databases with really high performance, low latency
- Helps reduce load off of databases for read intensive workloads
- Helps make your application stateless
- Write Scaling using sharding
- Read Scaling using Read Replicas
- Multi AZ with Failover Capability
- AWS takes care of OS maintenance / patching, optimizations, setup, configuration, monitoring, failure recovery and backups

# Redis Overview

- Redis is an in-memory key-value store
- Super low latency (sub ms)
- Cache survive reboots by default (it's called persistence)
- Great to host
  - User sessions
  - Leaderboard (for gaming)
  - Distributed states
  - Relieve pressure on databases (such as RDS)
  - Pub / Sub capability for messaging
- Multi AZ with Automatic Failover for disaster recovery if you don't want to lose your cache data
- Support for Read Replicas

# Memcached Overview

- Memcached is an in-memory object store
- Cache doesn't survive reboots
- Use cases:
  - Quick retrieval of objects from memory
  - Cache often accessed objects
- Overall, Redis has largely grown in popularity and has better feature sets than Memcached.
- I would personally only use Redis for caching needs.

# AWS Lambda

## Serverless data processing

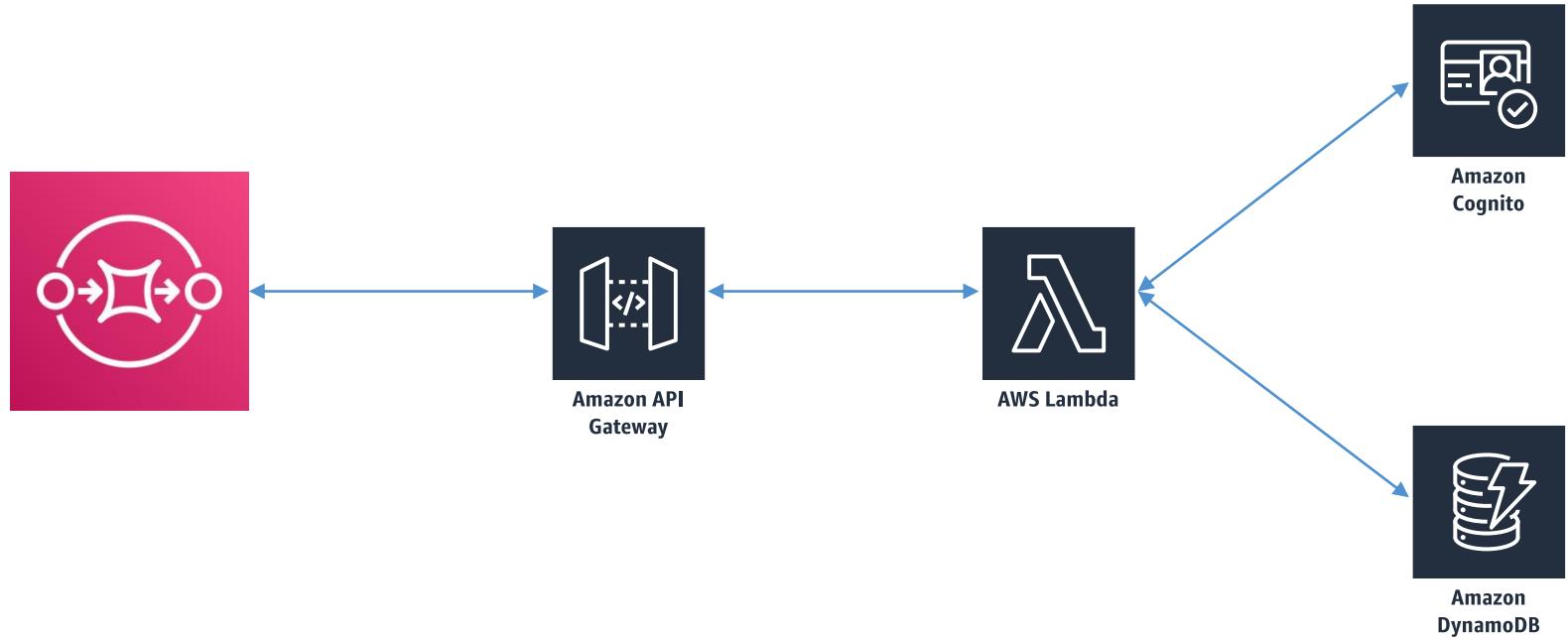
# What is Lambda?

- A way to run code snippets “in the cloud”
  - Serverless
  - Continuous scaling
- Often used to process data as it’s moved around

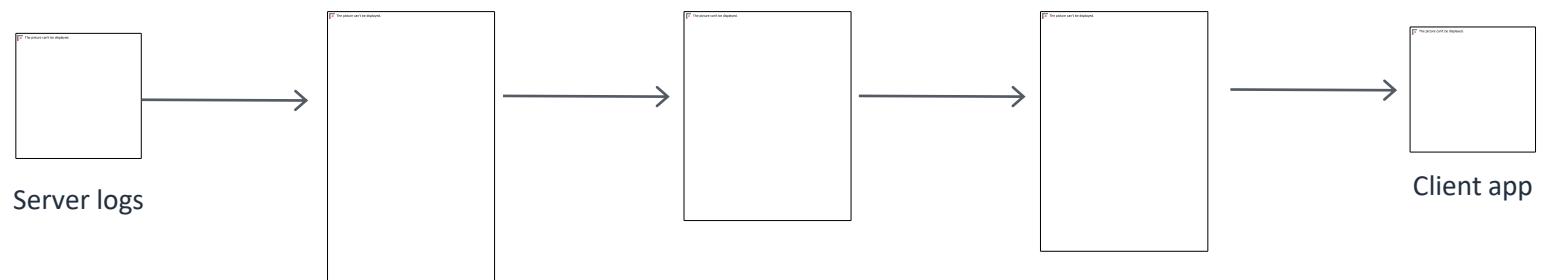


**AWS Lambda**

# Example: Serverless Website

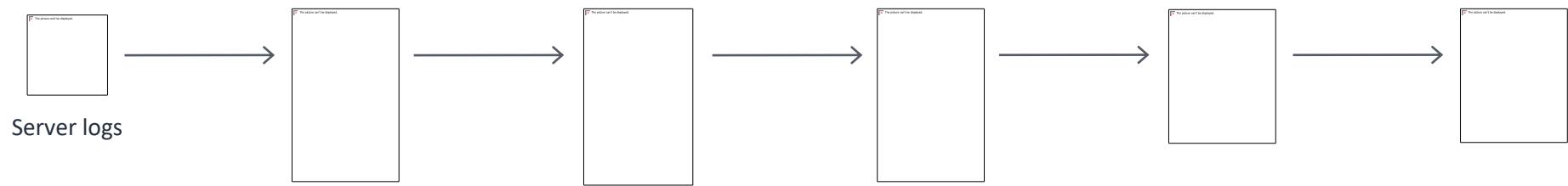


# Example: Order history app



# Example:

## Transaction rate alarm



# Why not just run a server?

- Server management (patches, monitoring, hardware failures, etc.)
- Servers can be cheap, but scaling gets expensive really fast
- You don't pay for processing time you don't use
- Easier to split up development between front-end and back-end

# Main uses of Lambda

- Real-time file processing
- Real-time stream processing
- ETL
- Cron replacement
- Process AWS events



# Supported languages

- Node.js
- Python
- Java
- C#
- Go
- Powershell
- Ruby



# Lambda triggers



Amazon S3



Amazon Simple Email Service



Amazon Kinesis Data Firehose



Amazon Kinesis Data Streams



Amazon DynamoDB



Amazon SNS



Amazon SQS



AWS Config



AWS IoT  
Button



Amazon Lex



Amazon  
CloudWatch



AWS  
CloudFormation



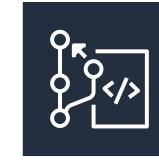
Amazon API  
Gateway



Amazon  
CloudFront



Amazon  
Cognito



AWS  
CodeCommit

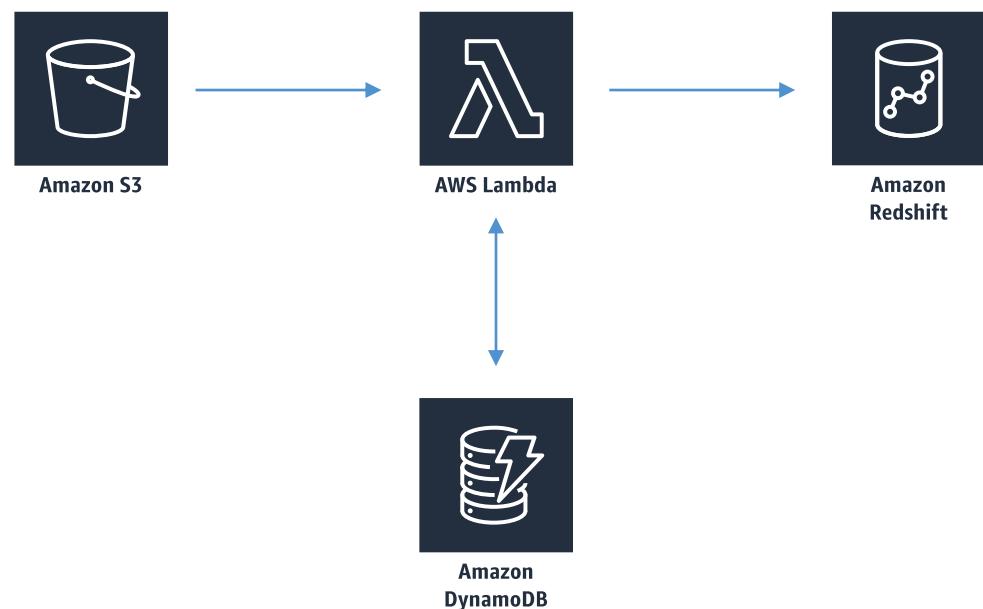
# Lambda and Amazon Elasticsearch Service



# Lambda and Data Pipeline

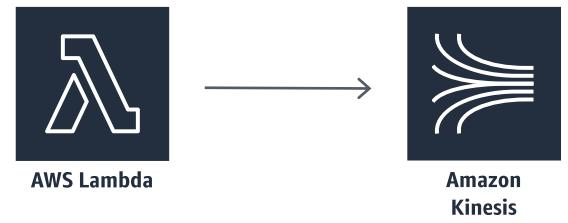


# Lambda and Redshift



# Lambda + Kinesis

- Your Lambda code receives an event with a **batch** of stream records
  - You specify a batch size when setting up the trigger (up to 10,000 records)
  - Too large a batch size can cause timeouts!
  - Batches may also be split beyond Lambda's payload limit (6 MB)
- Lambda will retry the batch until it succeeds or the data expires
  - This can stall the shard if you don't handle errors properly
  - Use more shards to ensure processing isn't totally held up by errors
- Lambda processes shard data synchronously



# Cost Model

- “Pay for what you use”
- Generous free tier (1M requests / month, 400K GB-seconds compute time)
- \$0.20 / million requests
- \$.00001667 per GB/second



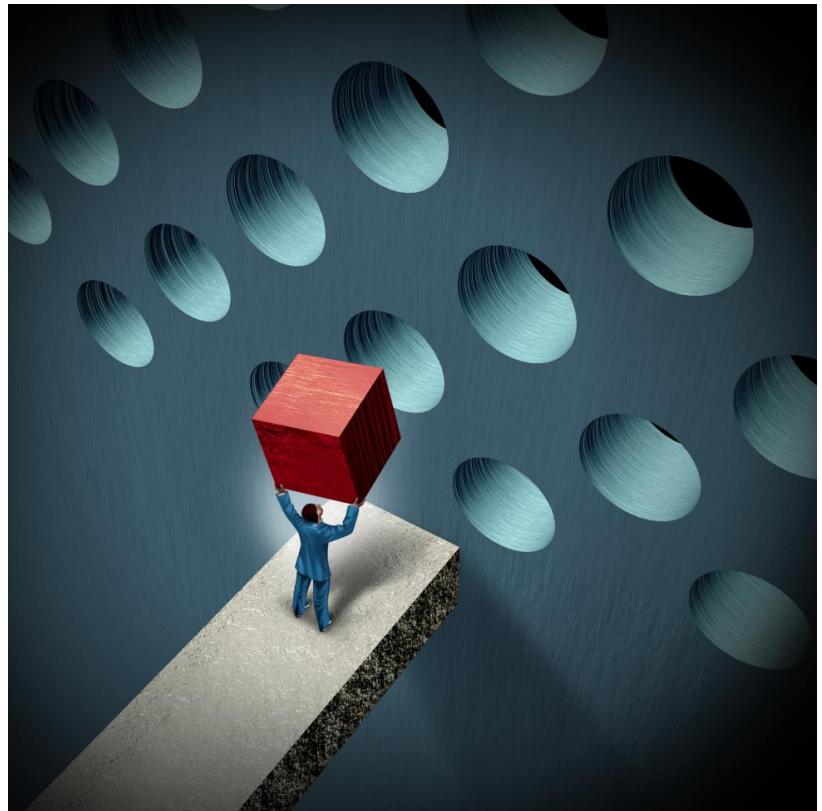
# Other promises

- High availability
- Unlimited scalability\*
- High performance
  - But you do specify a timeout!  
This can cause problems.  
Max is 900 seconds.



# Anti-patterns

- Long-running applications
- Dynamic websites
- Stateful applications



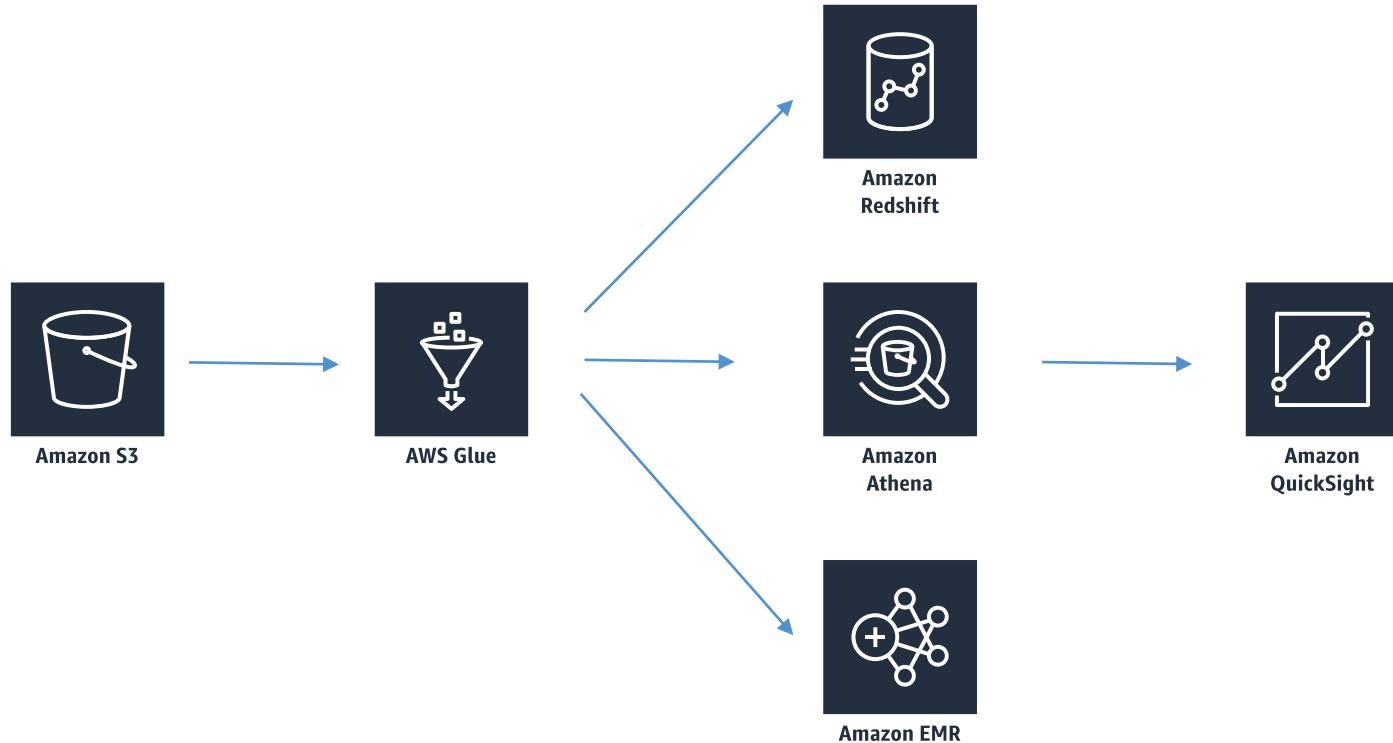
# AWS Glue

Table definitions and ETL

# What is Glue?

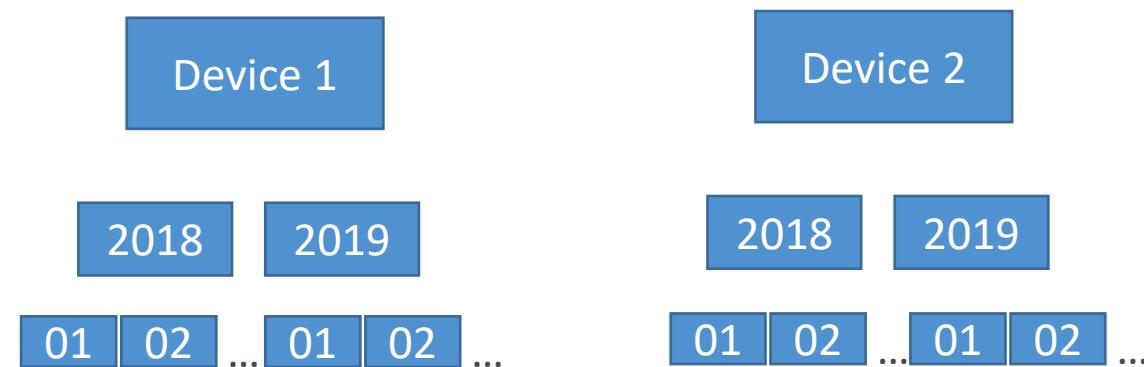
- Serverless discovery and definition of table definitions and schema
  - S3 “data lakes”
  - RDS
  - Redshift
  - Most other SQL databases
- Custom ETL jobs
  - Trigger-driven, on a schedule, or on demand
  - Fully managed

# Glue Crawler / Data Catalog



# Glue and S3 Partitions

- Glue crawler will extract partitions based on how your S3 data is organized
- Think up front about how you will be querying your data lake in S3
- Example: devices send sensor data every hour
- Do you query primarily by time ranges?
  - If so, organize your buckets as yyyy/mm/dd/device
- Do you query primarily by device?
  - If so, organize your buckets as device/yyyy/mm/dd



# Glue + Hive



# Glue ETL

- Automatic code generation
- Scala or Python
- Encryption
  - Server-side (at rest)
  - SSL (in transit)
- Can be event-driven
- Can provision additional “DPU’s” (data processing units) to increase performance of underlying Spark jobs
- Errors reported to CloudWatch

# Glue ETL

- Transform data, Clean Data, Enrich Data (before doing analysis)
  - Generate ETL code in Python or Scala, you can modify the code
  - Can provide your own Spark or PySpark scripts
  - Target can be S3, JDBC (RDS, Redshift), or in Glue Data Catalog
- Fully managed, cost effective, pay only for the resources consumed
- Jobs are run on a serverless Spark platform
- Glue Scheduler to schedule the jobs
- Glue Triggers to automate job runs based on “events”

# Glue ETL - Transformations

- Bundled Transformations:
  - DropFields, DropNullFields – remove (null) fields
  - Filter – specify a function to filter records
  - Join – to enrich data
  - Map - add fields, delete fields, perform external lookups
- Machine Learning Transformations:
  - **FindMatches ML:** identify duplicate or matching records in your dataset, even when the records do not have a common unique identifier and no fields match exactly.
  - Format conversions: CSV, JSON, Avro, Parquet, ORC, XML
  - Apache Spark transformations (example: K-Means)

# AWS Glue Development Endpoints

- Develop ETL scripts using a notebook
  - Then create an ETL job that runs your script (using Spark and Glue)
- Endpoint is in a VPC controlled by security groups, connect via:
  - Apache Zeppelin on your local machine
  - Zeppelin notebook server on EC2 (via Glue console)
  - SageMaker notebook
  - Terminal window
  - PyCharm professional edition
  - Use Elastic IP's to access a private endpoint address



# Running Glue jobs

- Time-based schedules (cron style)
- Job bookmarks
  - Persists state from the job run
  - Prevents reprocessing of old data
  - Allows you to process new data only when re-running on a schedule
  - Works with S3 sources in a variety of formats
  - Works with relational databases via JDBC (if PK's are in sequential order)
    - Only handles new rows, not updated rows
- CloudWatch Events
  - Fire off a Lambda function or SNS notification when ETL succeeds or fails
  - Invoke EC2 run, send event to Kinesis, activate a Step Function



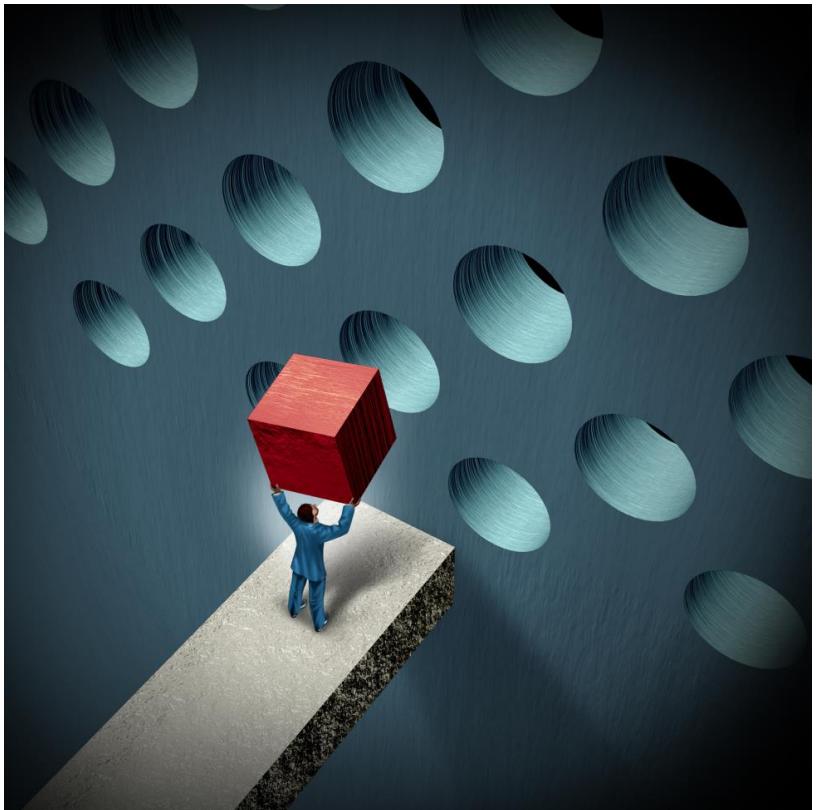
# Glue cost model

- Billed by the minute for crawler and ETL jobs
- First million objects stored and accesses are free for the Glue Data Catalog
- Development endpoints for developing ETL code charged by the minute



# Glue Anti-patterns

- Multiple ETL engines



# No longer an anti-pattern: streaming

- As of April 2020, Glue ETL supports serverless streaming ETL
  - Consumes from Kinesis or Kafka
  - Clean & transform in-flight
  - Store results into S3 or other data stores
- Runs on Apache Spark Structured Streaming

# EMR

## Elastic MapReduce

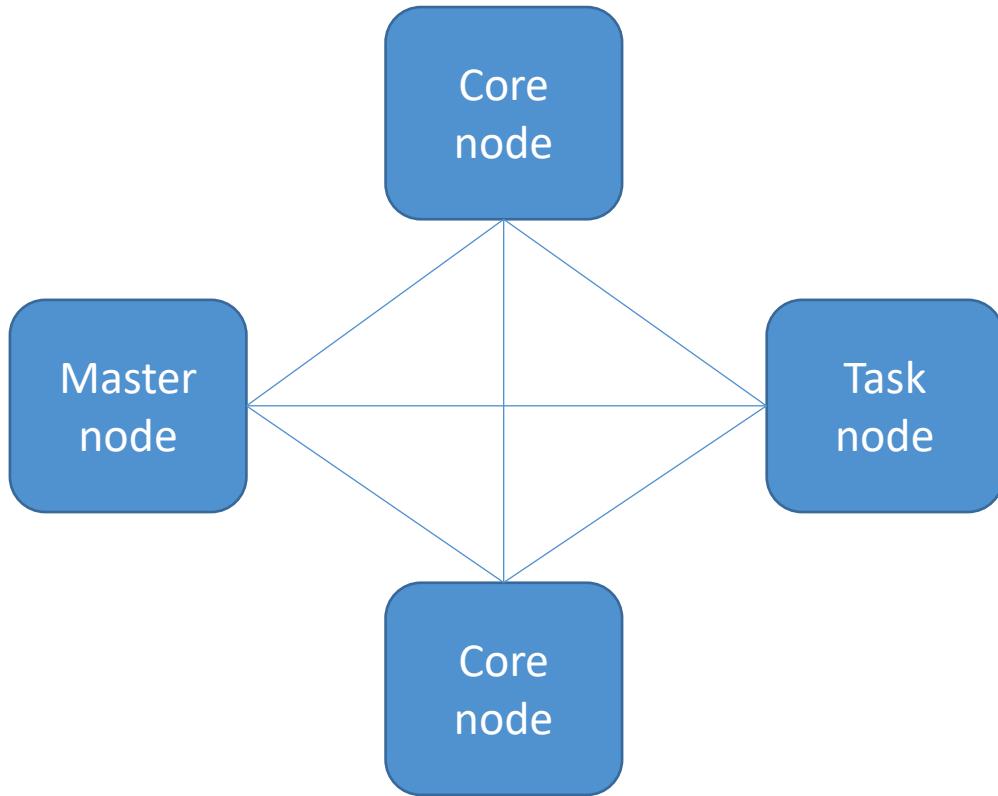
# What is EMR?

- Elastic MapReduce
- Managed Hadoop framework on EC2 instances
- Includes Spark, HBase, Presto, Flink, Hive & more
- EMR Notebooks
- Several integration points with AWS



**Amazon EMR**

# An EMR Cluster



- **Master node:** manages the cluster
  - Single EC2 instance
- **Core node:** Hosts HDFS data and runs tasks
  - Can be scaled up & down, but with some risk
- **Task node:** Runs tasks, does not host data
  - No risk of data loss when removing
  - Good use of **spot instances**

# EMR Usage

- Transient vs Long-Running Clusters
  - Can spin up task nodes using Spot instances for temporary capacity
  - Can use reserved instances on long-running clusters to save \$
- Connect directly to master to run jobs
- Submit ordered steps via the console

# EMR / AWS Integration

- Amazon EC2 for the instances that comprise the nodes in the cluster
- Amazon VPC to configure the virtual network in which you launch your instances
- Amazon S3 to store input and output data
- Amazon CloudWatch to monitor cluster performance and configure alarms
- AWS IAM to configure permissions
- AWS CloudTrail to audit requests made to the service
- AWS Data Pipeline to schedule and start your clusters

# EMR Storage

- HDFS
- EMRFS: access S3 as if it were HDFS
  - **EMRFS Consistent View** – Optional for S3 consistency
    - Uses DynamoDB to track consistency
- Local file system
- EBS for HDFS



# EMR promises

- EMR charges by the hour
  - Plus EC2 charges
- Provisions new nodes if a core node fails
- Can add and remove tasks nodes on the fly
- Can resize a running cluster's core nodes



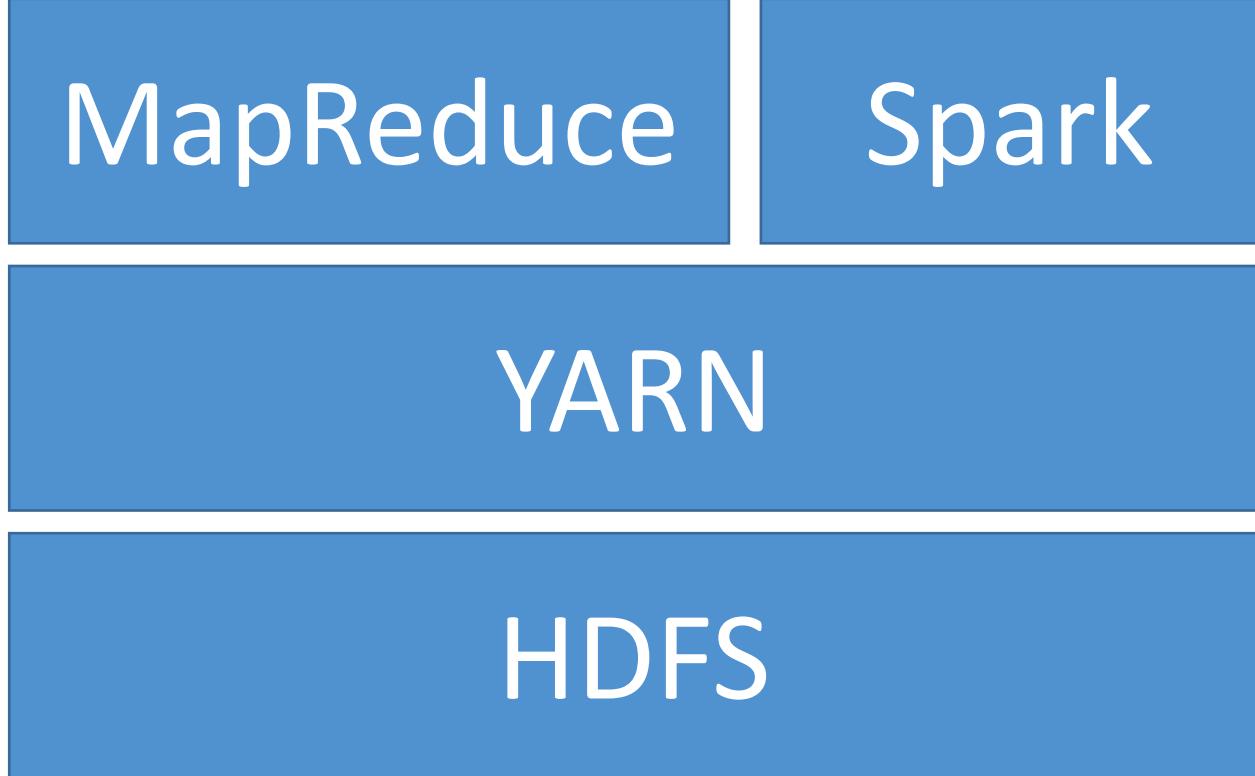
# So... what's Hadoop?

MapReduce

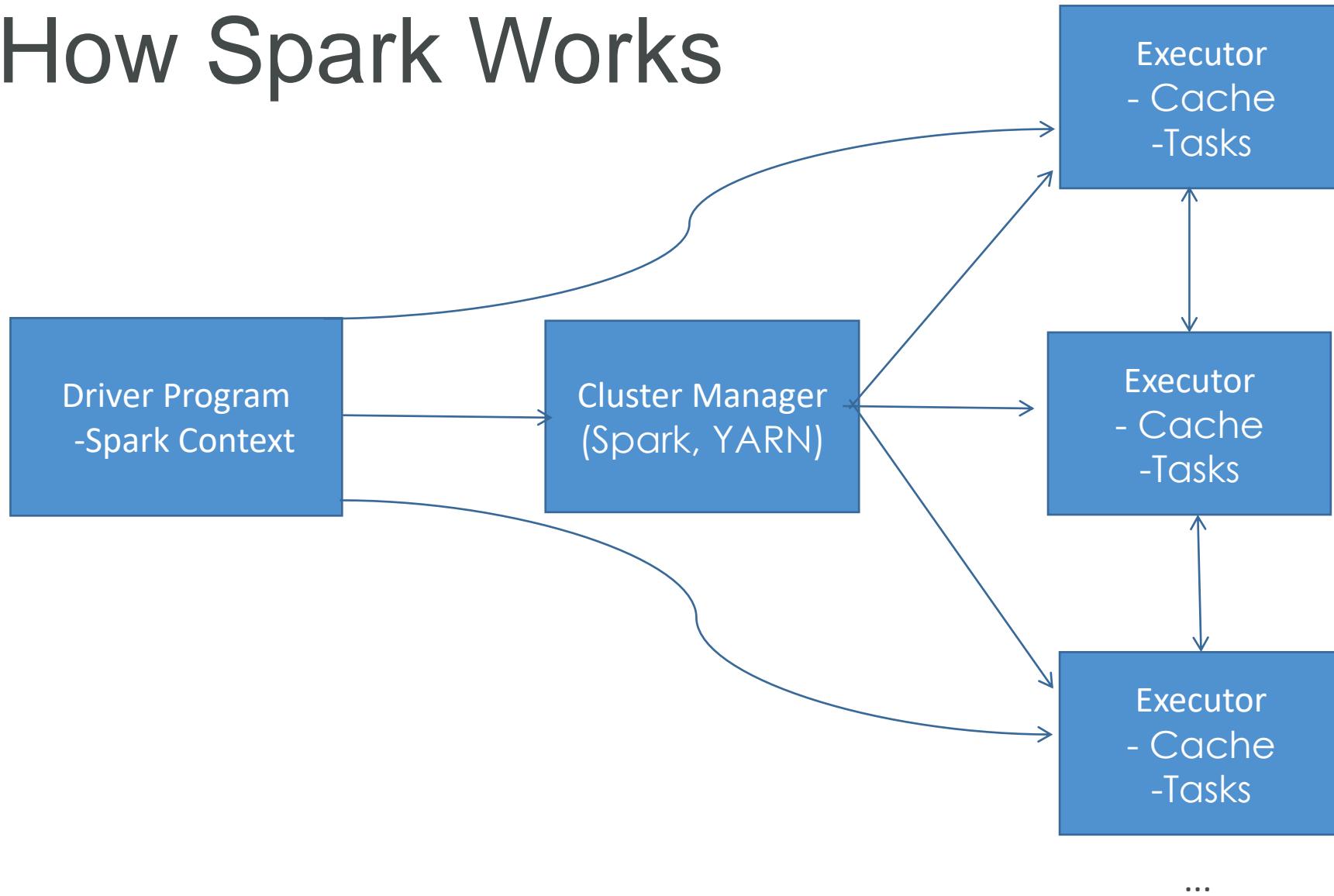
YARN

HDFS

# Apache Spark



# How Spark Works



# Spark Components

Spark Streaming

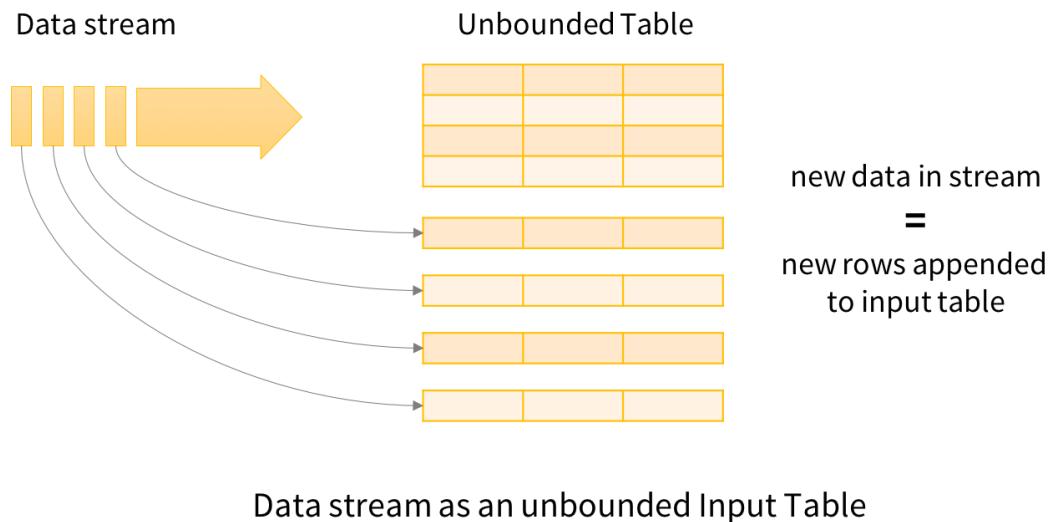
Spark SQL

MLLib

GraphX

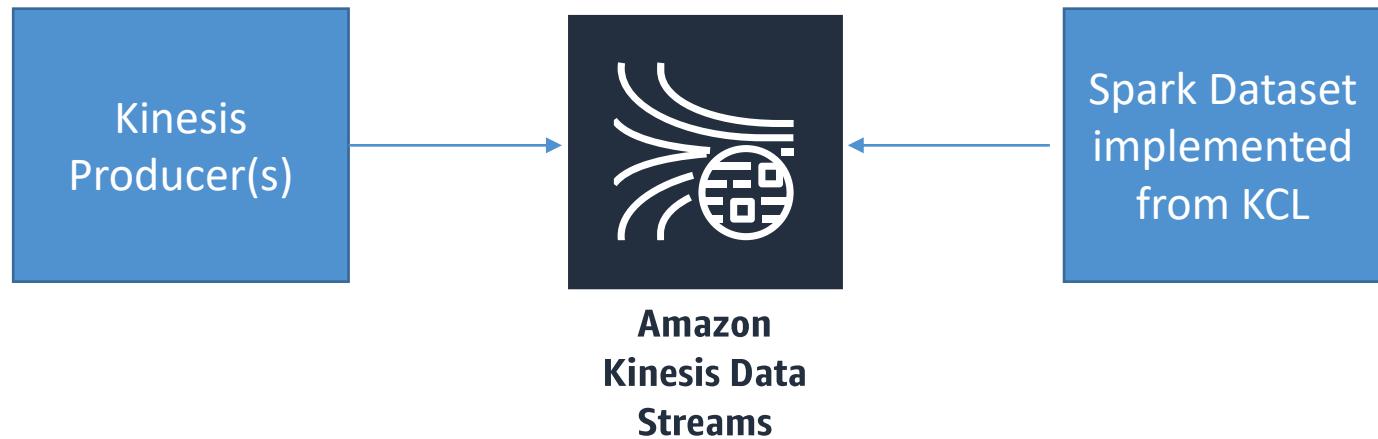
SPARK CORE

# Spark Structured Streaming



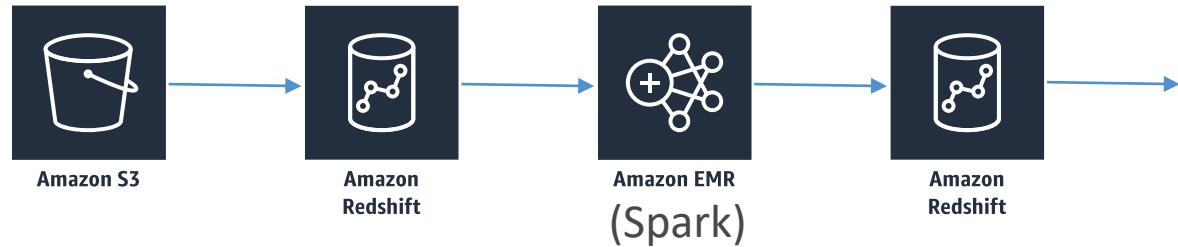
```
val inputDF = spark.readStream.json("s3://logs")
inputDF.groupBy($"action", window($"time", "1 hour")).count()
.writeStream.format("jdbc").start("jdbc:mysql//...")
```

# Spark Streaming + Kinesis

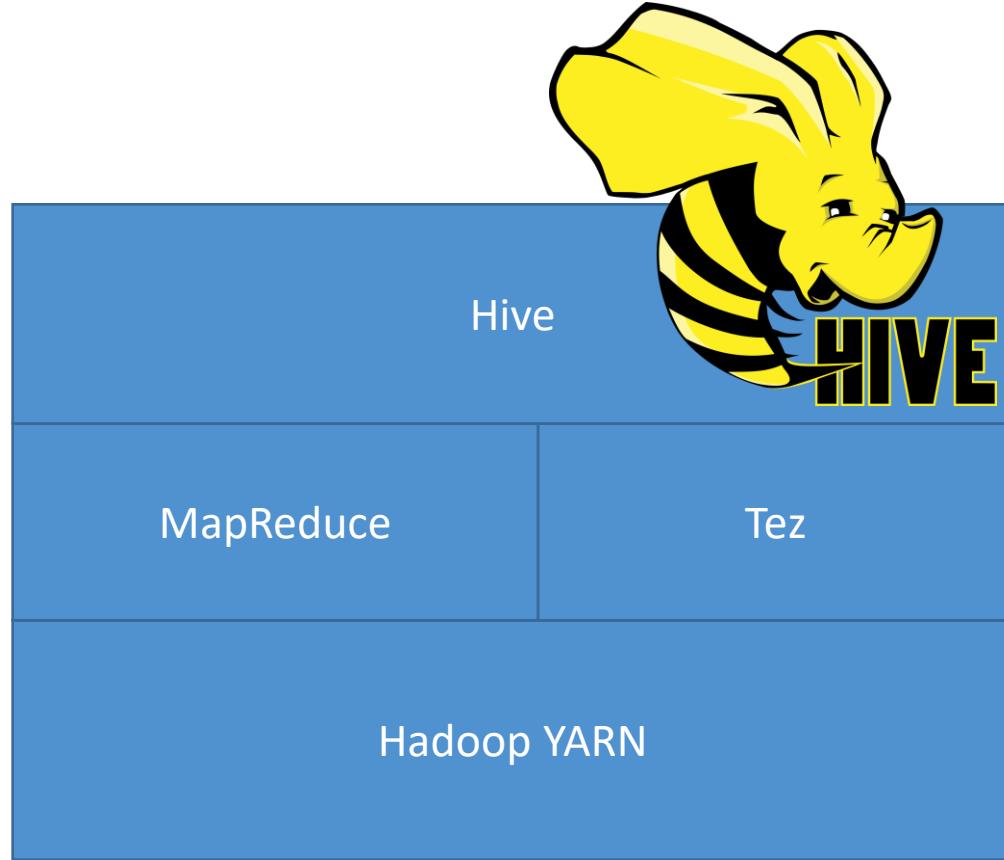


# Spark + Redshift

- spark-redshift package allows Spark datasets from Redshift
  - It's a Spark SQL data source
- Useful for ETL using Spark

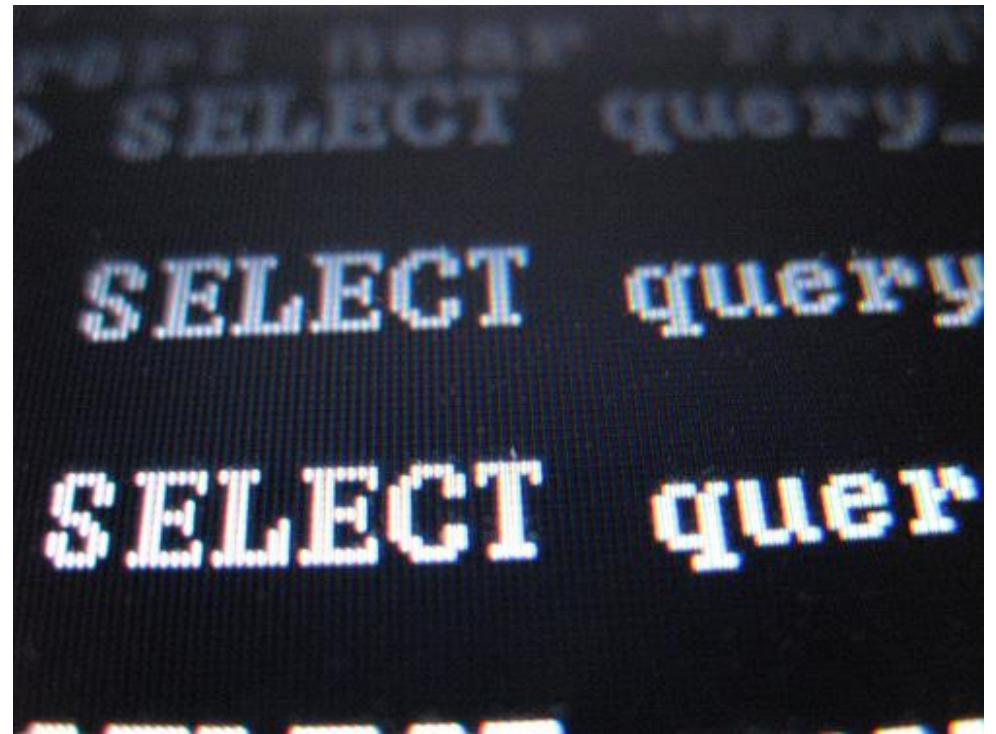


# Apache Hive



# Why Hive?

- Uses familiar SQL syntax (HiveQL)
- Interactive
- Scalable – works with “big data” on a cluster
  - Really most appropriate for data warehouse applications
- Easy OLAP queries – WAY easier than writing MapReduce in Java
- Highly optimized
- Highly extensible
  - User defined functions
  - Thrift server
  - JDBC / ODBC driver



# The Hive Metastore

- Hive maintains a “metastore” that imparts a structure you define on the unstructured data that is stored on HDFS etc.

```
CREATE TABLE ratings (
    userID INT,
    movieID      INT,
    rating INT,
    time   INT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE;

LOAD DATA LOCAL INPATH '${env:HOME}/ml-100k/u.data'
OVERWRITE INTO TABLE ratings;
```

# External Hive Metastores

- Metastore is stored in MySQL on the master node by default
- External metastores offer better resiliency / integration
  - AWS Glue Data Catalog
  - Amazon RDS



# Other Hive / AWS integration points

- Load table partitions from S3
- Write tables in S3
- Load scripts from S3
- DynamoDB as an external table



# Apache Pig

- Writing mappers and reducers by hand takes a long time.
- Pig introduces *Pig Latin*, a scripting language that lets you use SQL-like syntax to define your map and reduce steps.
- Highly extensible with user-defined functions (UDF's)



# How Pig Works



```

ratings = LOAD '/user/maria_dev/ml-100k/u.data' AS (userID:int, movieID:int, rating:int, ratingTime:int);

metadata = LOAD '/user/maria_dev/ml-100k/u.item' USING PigStorage('|')
    AS (movieID:int, movieTitle:chararray, releaseDate:chararray, videoRelease:chararray, imdbLink:chararray);

nameLookup = FOREACH metadata GENERATE movieID, movieTitle,
    ToUnixTime(ToDate(releaseDate, 'dd-MMM-yyyy')) AS releaseTime;

ratingsByMovie = GROUP ratings BY movieID;

avgRatings = FOREACH ratingsByMovie GENERATE group AS movieID, AVG(ratings.rating) AS avgRating;

fiveStarMovies = FILTER avgRatings BY avgRating > 4.0;

fiveStarsWithData = JOIN fiveStarMovies BY movieID, nameLookup BY movieID;

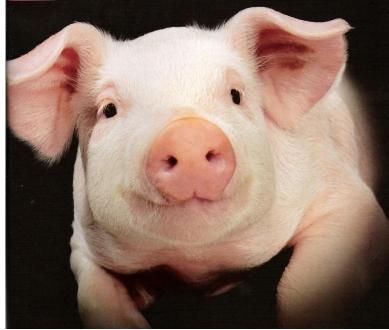
oldestFiveStarMovies = ORDER fiveStarsWithData BY nameLookup::releaseTime;

DUMP oldestFiveStarMovies;

```

# Pig / AWS Integration

- Ability to use multiple file systems (not just HDFS)
  - i.e., query data in S3
- Load JAR's and scripts from S3



Amazon S3

# HBase

- Non-relational, petabyte-scale database
- Based on Google's BigTable, on top of HDFS
- In-memory
- Hive integration



# Sounds a lot like DynamoDB

- Both are NoSQL databases intended for the same sorts of things
- But if you're all-in with AWS anyhow, DynamoDB has advantages
  - Fully managed (auto-scaling)
  - More integration with other AWS services
  - Glue integration
- HBase has some advantages though:
  - Efficient storage of sparse data
  - Appropriate for high frequency counters (consistent reads & writes)
  - High write & update throughput
  - More integration with Hadoop

# HBase / AWS integration

- Can store data (StoreFiles and metadata) on S3 via EMRFS
- Can back up to S3



Amazon S3

# Presto

- It can connect to many different “big data” databases and data stores at once, and query across them
- **Interactive queries at petabyte scale**
- Familiar SQL syntax
- Optimized for OLAP – analytical queries, data warehousing
- Developed, and still partially maintained by Facebook
- This is what Amazon Athena uses under the hood
- Exposes JDBC, Command-Line, and Tableau interfaces



# Presto connectors

- HDFS
- S3
- Cassandra
- MongoDB
- HBase
- SQL
- Redshift
- Teradata



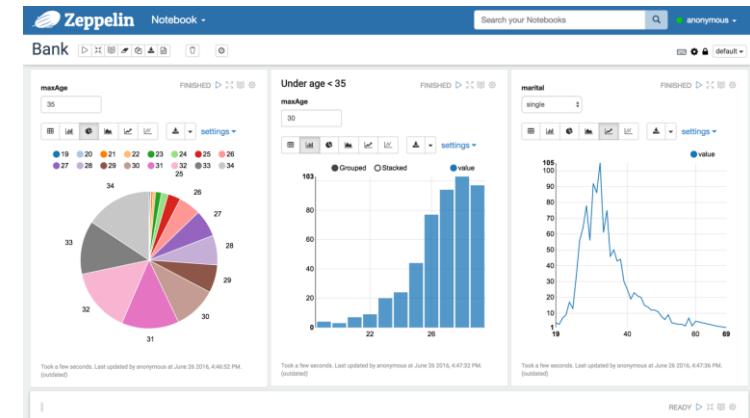
# Apache Zeppelin



- If you're familiar with iPython notebooks – it's like that
  - Lets you interactively run scripts / code against your data
  - Can interleave with nicely formatted notes
  - Can share notebooks with others on your cluster
- Spark, Python, JDBC, HBase, Elasticsearch + more

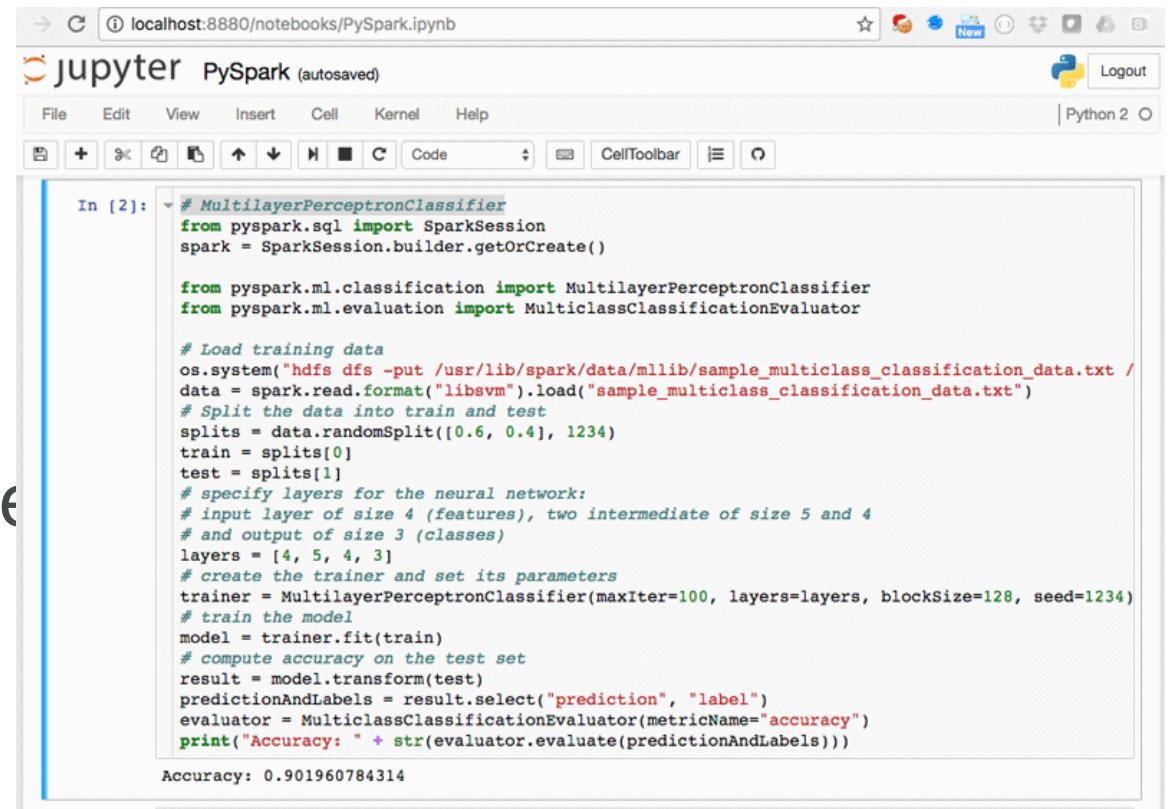
# Zeppelin + Spark

- Can run Spark code interactively (like you can in the Spark shell)
  - This speeds up your development cycle
  - And allows easy experimentation and exploration of your big data
- Can execute SQL queries directly against SparkSQL
- Query results may be visualized in charts and graphs
- Makes Spark feel more like a data science tool!



# EMR Notebook

- Similar concept to Zeppelin, with more AWS integration
- Notebooks backed up to S3
- Provision clusters from the notebook!
- Hosted inside a VPC
- Accessed only via AWS console



The screenshot shows a Jupyter Notebook interface with the title "PySpark" in the header. The notebook URL is "localhost:8880/notebooks/PySpark.ipynb". The code in cell [2] is as follows:

```
# MultilayerPerceptronClassifier
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()

from pyspark.ml.classification import MultilayerPerceptronClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Load training data
os.system("hdfs dfs -put /usr/lib/spark/data/mllib/sample_multiclass_classification_data.txt /")
data = spark.read.format("libsvm").load("sample_multiclass_classification_data.txt")

# Split the data into train and test
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]

# specify layers for the neural network:
# input layer of size 4 (features), two intermediate of size 5 and 4
# and output of size 3 (classes)
layers = [4, 5, 4, 3]

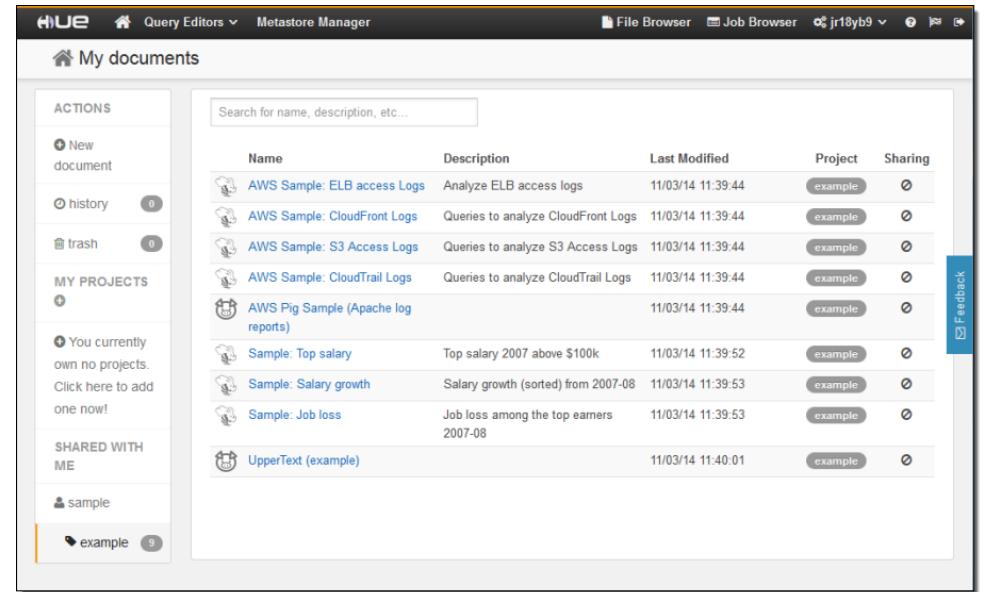
# create the trainer and set its parameters
trainer = MultilayerPerceptronClassifier(maxIter=100, layers=layers, blockSize=128, seed=1234)
# train the model
model = trainer.fit(train)
# compute accuracy on the test set
result = model.transform(test)
predictionAndLabels = result.select("prediction", "label")
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Accuracy: " + str(evaluator.evaluate(predictionAndLabels)))
```

The output of the code is:

```
Accuracy: 0.901960784314
```

# Hue

- Hadoop User Experience
- Graphical front-end for applications on your EMR cluster
- IAM integration: Hue Super-users inherit IAM roles
- S3: Can browse & move data between HDFS and S3

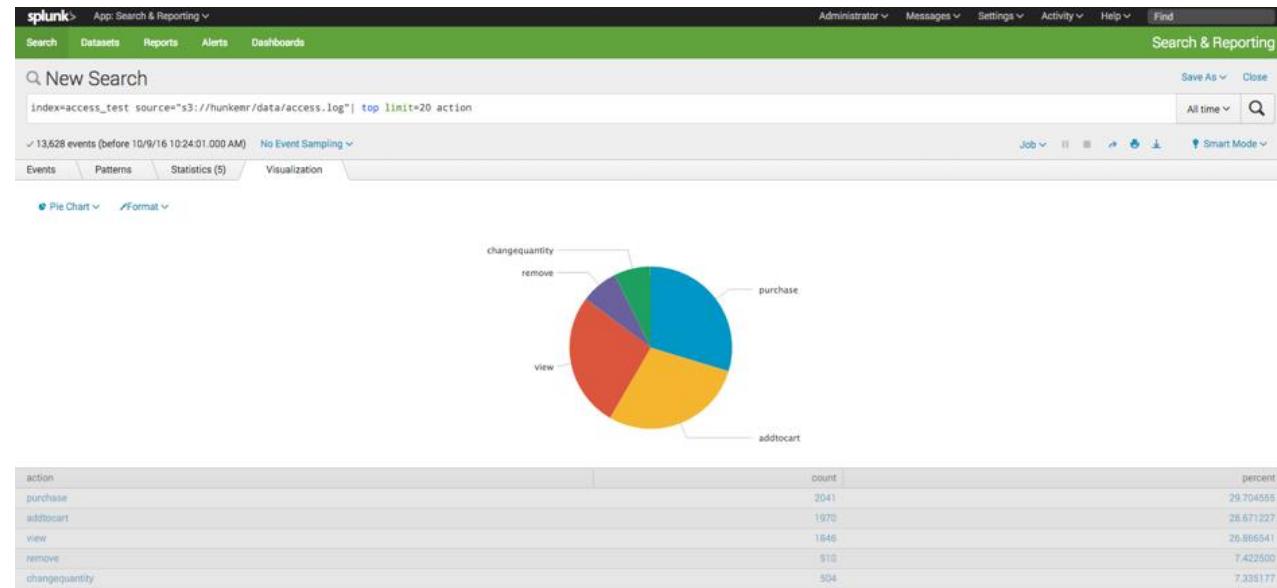


The screenshot shows the Hue web interface with the title 'HUE' at the top. The main area is titled 'My documents' and contains a table of data. The table has columns for Name, Description, Last Modified, Project, and Sharing. The data includes various sample queries and logs from AWS services like ELB, CloudFront, S3, and CloudTrail, along with some Pig samples and salary-related queries. A sidebar on the left shows options for 'New document', 'history', 'trash', 'MY PROJECTS', and 'SHARED WITH ME'.

Name	Description	Last Modified	Project	Sharing
AWS Sample: ELB access Logs	Analyze ELB access logs	11/03/14 11:39:44	example	0
AWS Sample: CloudFront Logs	Queries to analyze CloudFront Logs	11/03/14 11:39:44	example	0
AWS Sample: S3 Access Logs	Queries to analyze S3 Access Logs	11/03/14 11:39:44	example	0
AWS Sample: CloudTrail Logs	Queries to analyze CloudTrail Logs	11/03/14 11:39:44	example	0
AWS Pig Sample (Apache log reports)		11/03/14 11:39:44	example	0
Sample: Top salary	Top salary 2007 above \$100k	11/03/14 11:39:52	example	0
Sample: Salary growth	Salary growth (sorted) from 2007-08	11/03/14 11:39:53	example	0
Sample: Job loss	Job loss among the top earners 2007-08	11/03/14 11:39:53	example	0
UpperText (example)		11/03/14 11:40:01	example	0

# Splunk

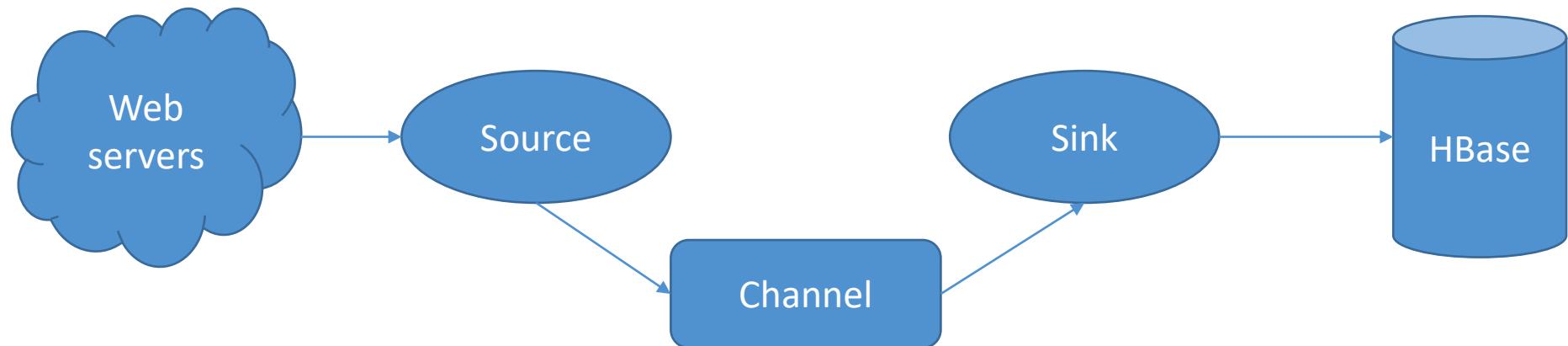
- Splunk / Hunk “makes machine data accessible, usable, and valuable to everyone”
- Operational tool – can be used to visualize EMR and S3 data using your EMR Hadoop cluster.



# Flume



- Another way to stream data into your cluster
- Made from the start with Hadoop in mind
  - Built-in sinks for HDFS and HBase
- Originally made to handle log aggregation



# MXNet

- Like Tensorflow, a library for building and accelerating neural networks
- Included on EMR



# S3DistCP

- Tool for copying large amounts of data
  - From S3 into HDFS
  - From HDFS into S3
- Uses MapReduce to copy in a distributed manner
- Suitable for parallel copying of large numbers of objects
  - Across buckets, across accounts



# Other EMR / Hadoop Tools

- **Ganglia** (monitoring)
- **Mahout** (machine learning)
- **Accumulo** (another NoSQL database)
- **Sqoop** (relational database connector)
- **HCatalog** (table and storage management for Hive metastore)
- **Kinesis Connector** (directly access Kinesis streams in your scripts)
- **Tachyon** (accelerator for Spark)
- **Derby** (open-source relational DB in Java)
- **Ranger** (data security manager for Hadoop)
- Install whatever you want

# EMR Security

- IAM policies
- Kerberos
- SSH
- IAM roles



# EMR: Choosing Instance Types

- Master node:
  - m4.large if < 50 nodes, m4.xlarge if > 50 nodes
- Core & task nodes:
  - m4.large is usually good
  - If cluster waits a lot on external dependencies (i.e. a web crawler), t2.medium
  - Improved performance: m4.xlarge
  - Computation-intensive applications: high CPU instances
  - Database, memory-caching applications: high memory instances
  - Network / CPU-intensive (NLP, ML) – cluster computer instances
- Spot instances
  - Good choice for task nodes
  - Only use on core & master if you're testing or very cost-sensitive; you're risking partial data loss

# Amazon Machine Learning

ML with linear and logistic regression

# Machine Learning 101

- Machine learning systems predict some unknown property of an item, given its other properties
- Examples:
  - How much will this house sell for?
  - What is this a picture of?
  - Is this biopsy result malignant?
  - Is this financial transaction fraudulent?



# Supervised Learning

- Supervised machine learning systems are **trained**
  - The property we want to predict is called a **label**
  - Our **training data set** contains labels known to be correct, together with the other attributes of the data (i.e., known house sale price given its location, # of bedrooms, square feet, etc.)
  - This training data is used to build a **model** that can then make **predictions** of unknown labels



# Train / Test

- Your training data can be randomly split into a **training set** and a **test set**
- Only the training set is used to train the model
- The model is then used on the test set
- We can then measure the **accuracy** of the predicted labels vs. their actual labels



# Types of models in Amazon ML

Example	Model type
What price will this house sell for?	Regression
What is this a picture of?	Multiclass Classification
Is this biopsy result malignant?	Binary Classification
Is this financial transaction fraudulent?	Binary Classification



# Confusion Matrix

- A way to visualize the accuracy of multiclass classification predictive models

Predicted label	Dog	Cat	Fish
True label	1.00	0.00	0.00
Dog	0.00	0.62	0.38
Cat	0.00	0.00	1.00
Fish	0.00	0.00	1.00



# Hyperparameters

- Machine learning models often depend on tuning the parameters of the model itself
  - This is called **hyperparameter tuning**
- Parameters in Amazon ML include:
  - Learning rate
  - Model size
  - Number of passes
  - Data shuffling
  - Regularization



# Amazon Machine Learning (ML)

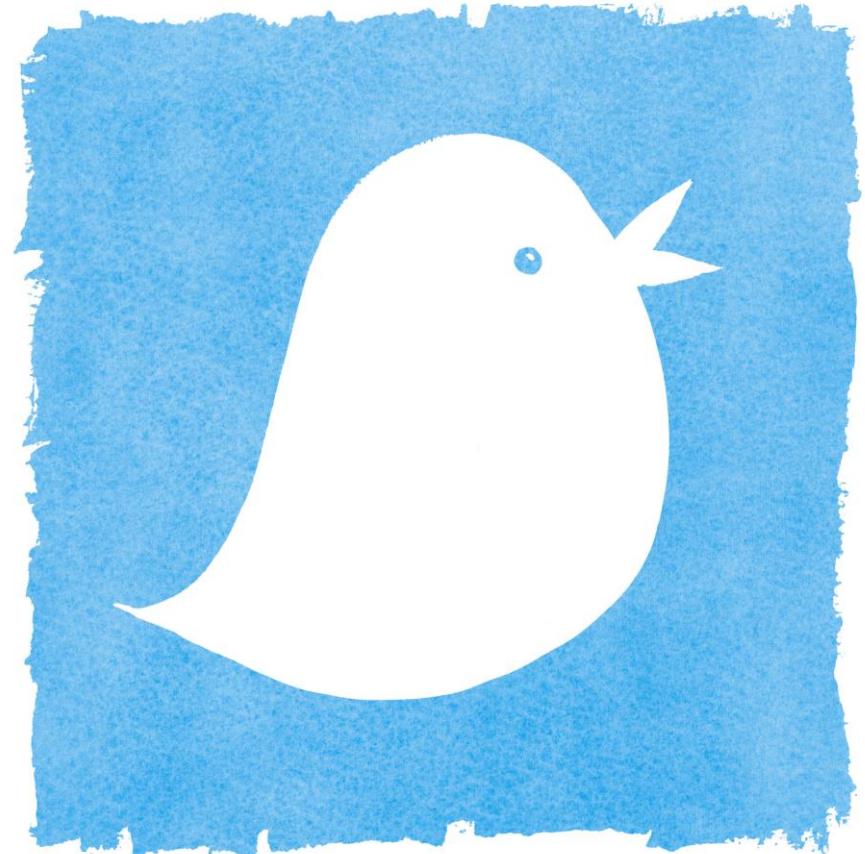
- Provides visualization tools & wizards to make creating a model easy
- You point it to training data in S3, Redshift, or RDS
- It builds a model than can make predictions using batches or a low-latency API
- Can do train/test and evaluate your model
- Fully managed
- Honestly it's a bit outdated now



Amazon  
Machine  
Learning

# “Ideal Usage Patterns”

- Flag suspicious transactions (fraud detection)
- Forecasting product demand
- Personalization – predict items a user will be interested in
- Predict user activity (we'll do this)
- Classify social media (does this Tweet require my attention?)



# Amazon ML: Cost Model

- “Pay for what you use”
- Charged for compute time
- Number of predictions
- Memory used to run your model
- Compute-hours for training

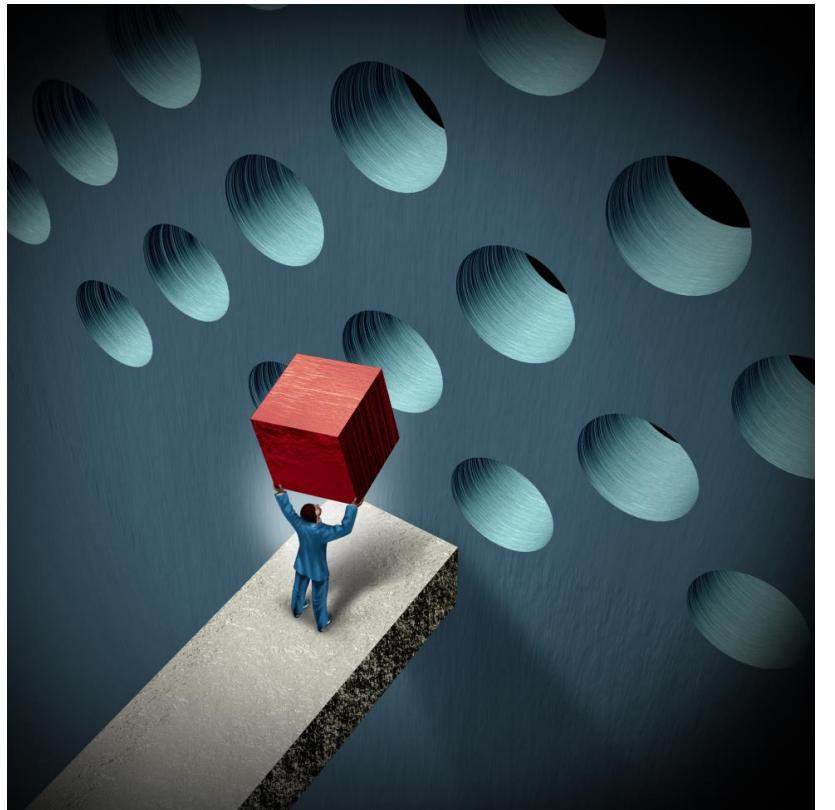
# Amazon ML: Promises & Limitations

- No downtime
- Up to 100GB training data  
(more via support ticket)
- Up to 5 simultaneous jobs  
(more via support ticket)



# Amazon ML: Anti-Patterns

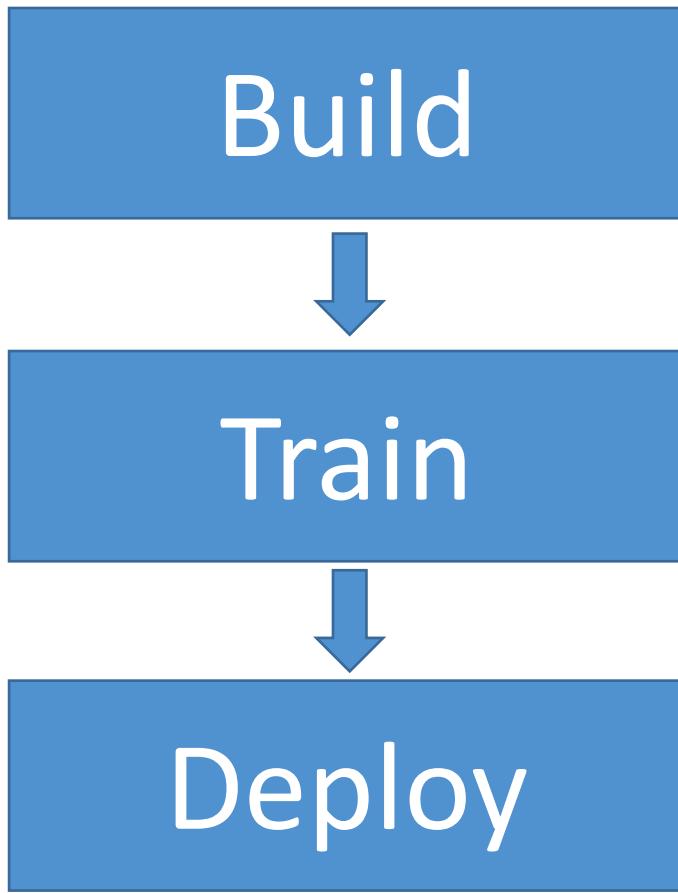
- Terabyte-scale data
- Unsupported learning tasks
  - Sequence prediction
  - Unsupervised clustering
  - Deep learning
- EMR / Spark is an (unmanaged) alternative.



# Amazon SageMaker

Scalable, fully-managed machine learning

# SageMaker modules



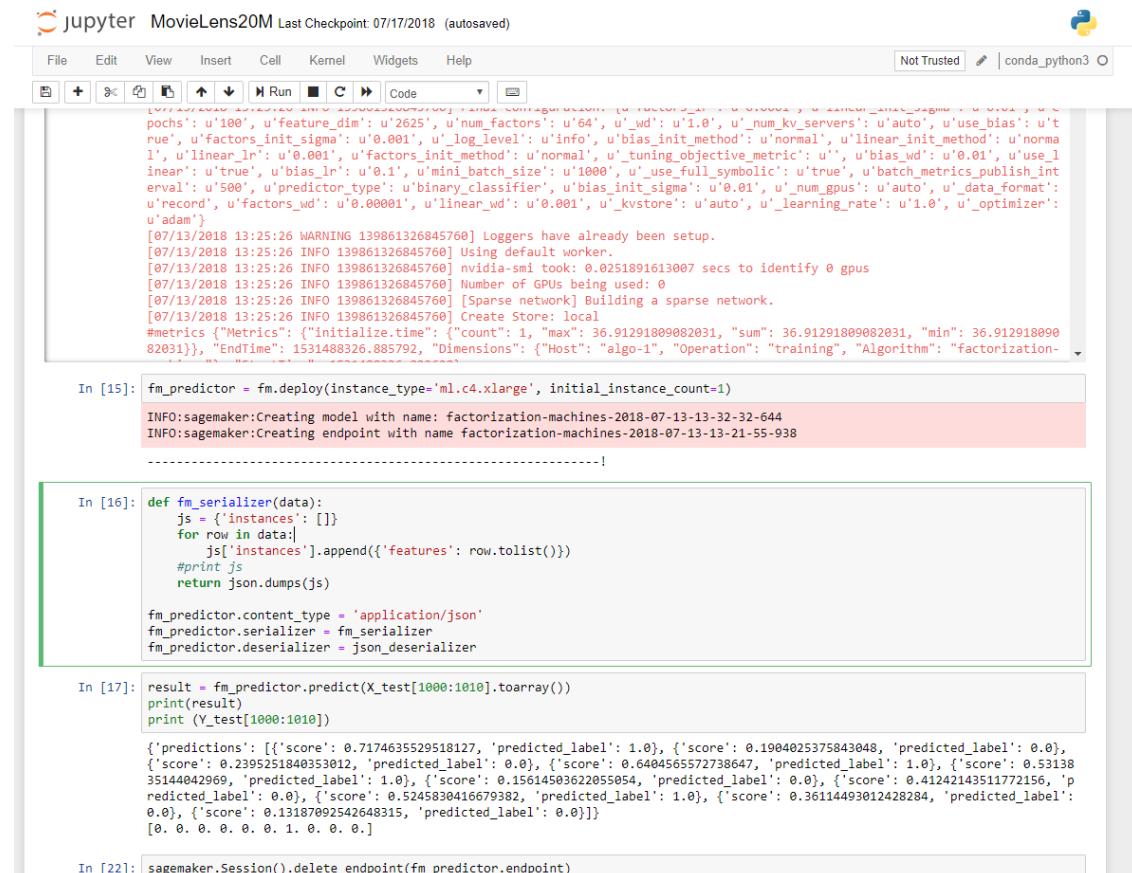
Amazon  
SageMaker

# SageMaker is powerful

- Tensorflow
- Apache MXNet
- GPU accelerated deep learning
- Scaling effectively unlimited
- Hyperparameter tuning jobs



# Jupyter Notebooks



The screenshot shows a Jupyter Notebook interface with the title "jupyter MovieLens20M Last Checkpoint: 07/17/2018 (autosaved)". The notebook has tabs for File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A "Not Trusted" warning is visible. The main area contains several code cells and their outputs.

```

In [15]: fm_predictor = fm.deploy(instance_type='ml.c4.xlarge', initial_instance_count=1)
INFO:sagemaker:Creating model with name: factorization-machines-2018-07-13-13-32-644
INFO:sagemaker:Creating endpoint with name factorization-machines-2018-07-13-13-21-55-938
-----!
```

```

In [16]: def fm_serializer(data):
    js = {'instances': []}
    for row in data:
        js['instances'].append({'features': row.tolist()})
    #print js
    return json.dumps(js)

fm_predictor.content_type = 'application/json'
fm_predictor.serializer = fm_serializer
fm_predictor.deserializer = json_deserializer

```

```

In [17]: result = fm_predictor.predict(X_test[1000:1010].toarray())
print(result)
print(Y_test[1000:1010])

```

```

{'predictions': [{'score': 0.7174635529518127, 'predicted_label': 1.0}, {'score': 0.1904025375843048, 'predicted_label': 0.0}, {'score': 0.2395251840353012, 'predicted_label': 0.0}, {'score': 0.6404565572738647, 'predicted_label': 1.0}, {'score': 0.5313835144042969, 'predicted_label': 1.0}, {"score": 0.15614503622055054, "predicted_label": 0.0}, {"score": 0.41242143511772156, "predicted_label": 0.0}, {"score": 0.5245830416679382, "predicted_label": 1.0}, {"score": 0.36114493012428284, "predicted_label": 0.0}, {"score": 0.13187092542648315, "predicted_label": 0.0}]}
[0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

```

In [22]: sagemaker.Session().delete_endpoint(fm_predictor.endpoint)

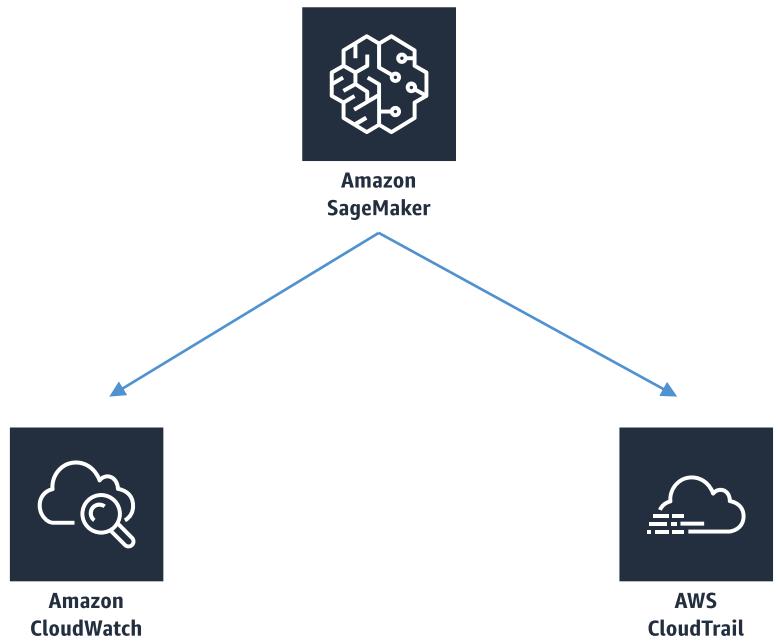
```

# SageMaker Security

- Code stored in “ML storage volumes”
  - Controlled by security groups
  - Optionally encrypted at rest
- All artifacts encrypted in transit and at rest
- API & console secured by SSL
- IAM roles
- Encrypted S3 buckets for data
- KMS integration for SageMaker notebooks, training jobs, endpoints



# SageMaker Operations



# Deep Learning 101

## And AWS Best Practices

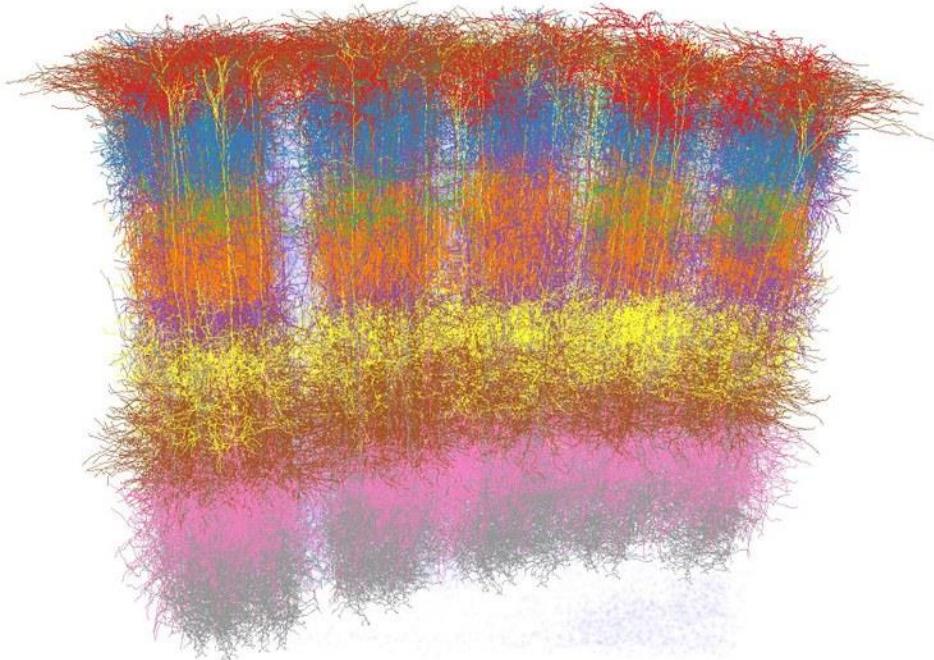
# The biological inspiration

- Neurons in your cerebral cortex are connected via axons
- A neuron “fires” to the neurons it’s connected to, when enough of its input signals are activated.
- Very simple at the individual neuron level – but layers of neurons connected in this way can yield learning behavior.
- Billions of neurons, each with thousands of connections, yields a mind



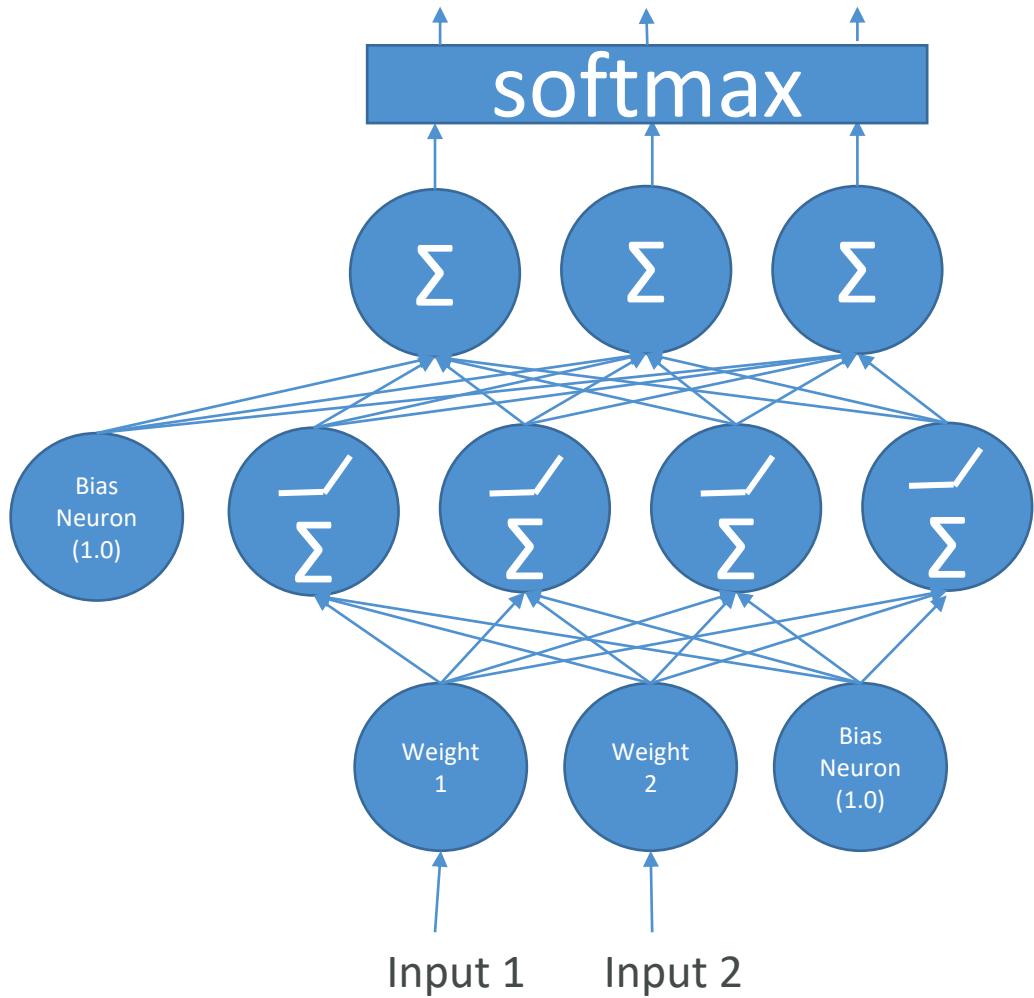
# Cortical columns

- Neurons in your cortex seem to be arranged into many stacks, or “columns” that process information in parallel
- “mini-columns” of around 100 neurons are organized into larger “hyper-columns”. There are 100 million mini-columns in your cortex
- This is coincidentally similar to how GPU’s work...



(credit: Marcel Oberlaender et al.)

# Deep Neural Networks



# Deep Learning Frameworks

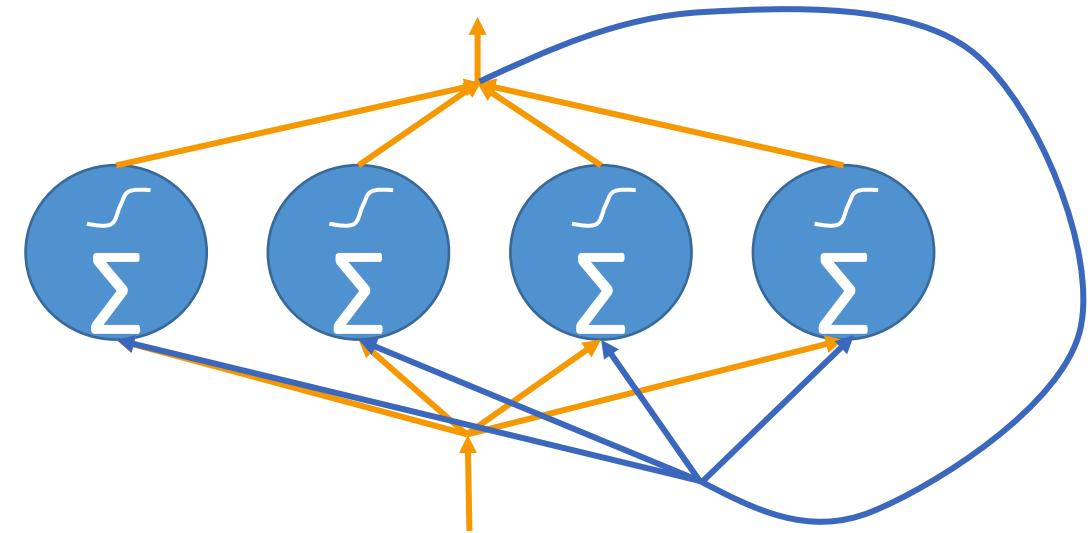
- Tensorflow / Keras
- MXNet

```
model = Sequential()

model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9,
           nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd, metrics=['accuracy'])
```

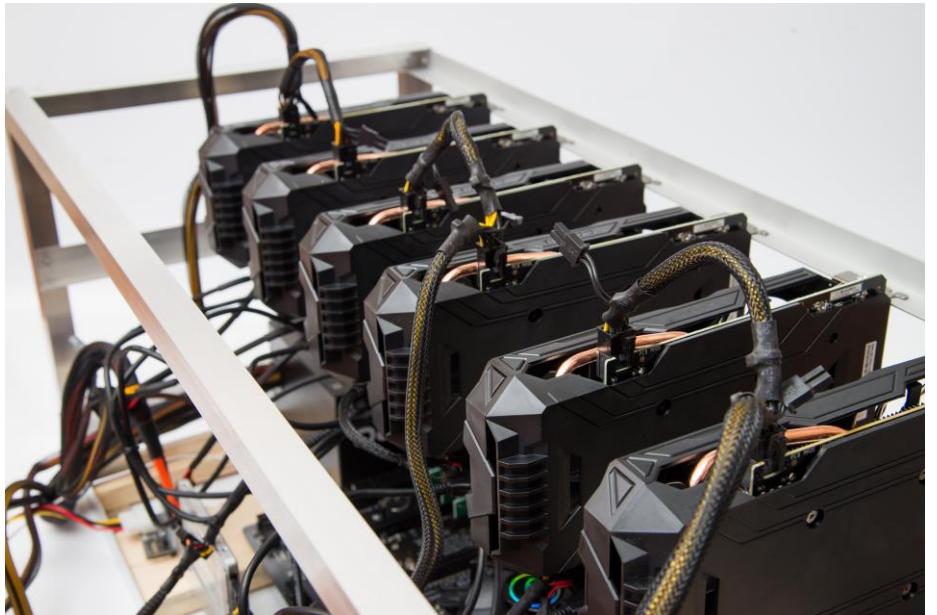
# Types of Neural Networks

- Feedforward Neural Network
- Convolutional Neural Networks (CNN)
  - Image classification (is there a stop sign in this image?)
- Recurrent Neural Networks (RNNs)
  - Deals with sequences in time (predict stock prices, understand words in a sentence, etc)
  - **LSTM, GRU**



# Deep Learning on EC2 / EMR

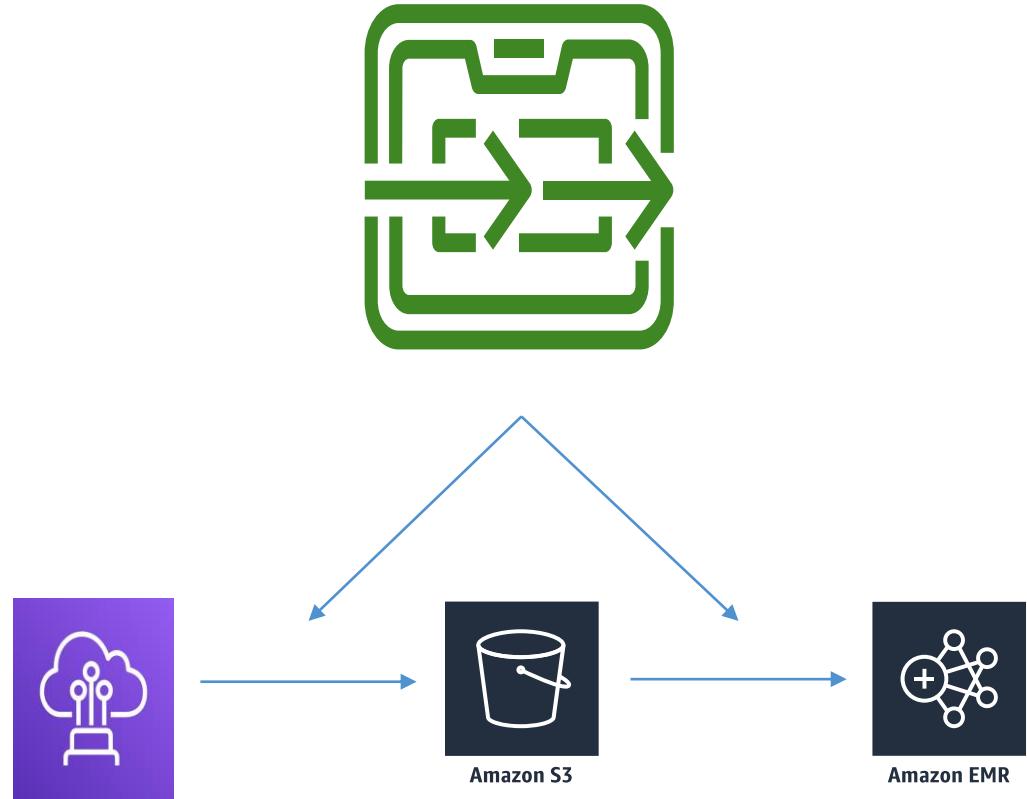
- EMR supports Apache MXNet and GPU instance types
- Appropriate instance types for deep learning:
  - P3: 8 Tesla V100 GPU's
  - P2: 16 K80 GPU's
  - G3: 4 M60 GPU's (all Nvidia chips)
- Deep Learning AMI's



# AWS Data Pipeline

A high-level overview

# Data Pipeline example



# Data Pipeline Features

- Destinations include S3, RDS, DynamoDB, Redshift and EMR
- Manages task dependencies
- Retries and notifies on failures
- Cross-region pipelines
- Precondition checks
- Data sources may be on-premises
- Highly available



# Data Pipeline Activities

- EMR
- Hive
- Copy
- SQL
- Scripts



# AWS Step Functions

A high-level overview

# AWS Step Functions



- Use to design workflows
- Easy visualizations
- Advanced Error Handling and Retry mechanism outside the code
- Audit of the history of workflows
- Ability to “Wait” for an arbitrary amount of time
- Max execution time of a State Machine is 1 year

# Step Functions – Examples

## Train a Machine Learning Model

**Definition**  
Code is pre-configured by the chosen sample project. It can be edited after creation. Step Functions state machines are defined using the JSON-based Amazon States Language (ASL). [Learn more](#)

```

1  {
2    "StartAt": "Generate dataset",
3    "States": {
4      "Generate dataset": {
5        "Resource": "<GENERATE_LAMBDA_FUNCTION_ARN>",
6        "Type": "Task",
7        "Next": "Train model (XGBoost)"
8      },
9      "Train model (XGBoost)": {
10        "Resource": "arn:
<PARTITION>:states:::sagemaker:createTrainingJob.sync",
11        "Parameters": {
12          "AlgorithmSpecification": {
13            "TrainingImage": "<SAGEMAKER_TRAINING_IMAGE>",
14            "TrainingInputMode": "File"
15          },
16          "OutputDataConfig": {
17            "S3OutputPath": "s3://<S3_BUCKET>/models"
18          },
19          "StoppingCondition": {
20            "MaxRuntimeInSeconds": 86400
21          },
22          "ResourceConfig": {
23            "InstanceCount": 1,
24            "InstanceType": "ml.m4.xlarge",

```

```

graph TD
    Start((Start)) --> Generate[Generate dataset]
    Generate --> Train[Train model (XGBoost)]
    Train --> Save[Save Model]
    Save --> Batch[Batch transform]
    Batch --> End((End))

```

# Step Functions – Examples

## Tune a Machine Learning Model

**Definition**

Code is pre-configured by the chosen sample project. It can be edited after creation. Step Functions state machines are defined using the JSON-based Amazon States Language (ASL). [Learn more](#)

`1 {
2 "StartAt": "Generate Training Dataset",
3 "States": {
4 "Generate Training Dataset": {
5 "Resource": "<GENERATE_LAMBDA_FUNCTION_ARN>",
6 "Type": "Task",
7 "Next": "HyperparameterTuning (XGBoost)"
8 },
9 "HyperparameterTuning (XGBoost)": {
10 "Resource": "arn:
<PARTITION>:states:::sagemaker:createHyperParameterTuningJob.sync",
11 "Parameters": {
12 "HyperParameterTuningJobName.$": "
<JOB_NAME_FROM_LAMBDA>",
13 "HyperParameterTuningJobConfig": {
14 "Strategy": "Bayesian",
15 "HyperParameterTuningJobObjective": {
16 "Type": "Minimize",
17 "MetricName": "validation:rmse"
18 },
19 "ResourceLimits": {
20 "MaxNumberOfTrainingJobs": 2,
21 "MaxParallelTrainingJobs": 2
22 },
23 "ParameterRanges": {`

The screenshot shows the AWS Step Functions console interface. On the left, there is a code editor window displaying the ASL code for the state machine. On the right, there is a visual representation of the state machine as a directed graph. The graph starts with a yellow circle labeled 'Start'. An arrow points from 'Start' to a dashed box labeled 'Generate Training Dataset'. Another arrow points from 'Generate Training Dataset' to a dashed box labeled 'HyperparameterTuning (XGBoost)'. From 'HyperparameterTuning (XGBoost)', an arrow points to a dashed box labeled 'Extract Model Path'. From 'Extract Model Path', an arrow points to a dashed box labeled 'HyperparameterTuning - Save Model'. From 'HyperparameterTuning - Save Model', an arrow points to a dashed box labeled 'Extract Model Name'. From 'Extract Model Name', an arrow points to a dashed box labeled 'Batch transform'. Finally, an arrow points from 'Batch transform' to a yellow circle labeled 'End'.

# Step Functions – Examples

## Manage a Batch Job

**Definition**

Code is pre-configured by the chosen sample project. It can be edited after creation. Step Functions state machines are defined using the JSON-based Amazon States Language (ASL). [Learn more](#)

```

1  {
2      "Comment": "An example of the Amazon States Language for
3          notification on an AWS Batch job completion",
4      "StartAt": "Submit Batch Job",
5      "TimeoutSeconds": 3600,
6      "States": {
7          "Submit Batch Job": {
8              "Type": "Task",
9              "Resource": "arn:<PARTITION>:states:::batch:submitJob.sync",
10             "Parameters": {
11                 "JobName": "BatchJobNotification",
12                 "JobQueue": "<BATCH_QUEUE_ARN>",
13                 "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>"
14             },
15             "Next": "Notify Success",
16             "Catch": [
17                 {
18                     "ErrorEquals": [ "States.ALL" ],
19                     "Next": "Notify Failure"
20                 }
21             ],
22             "Notify Success": {
23                 "Type": "Task",
24                 "Resource": "arn:<PARTITION>:states:::sns:publish",
25                 "Parameters": {
26                     "TopicArn": "<TOPIC_ARN>",
27                     "Message": "Batch Job completed successfully"
28                 }
29             }
30         }
31     }
32 }
```

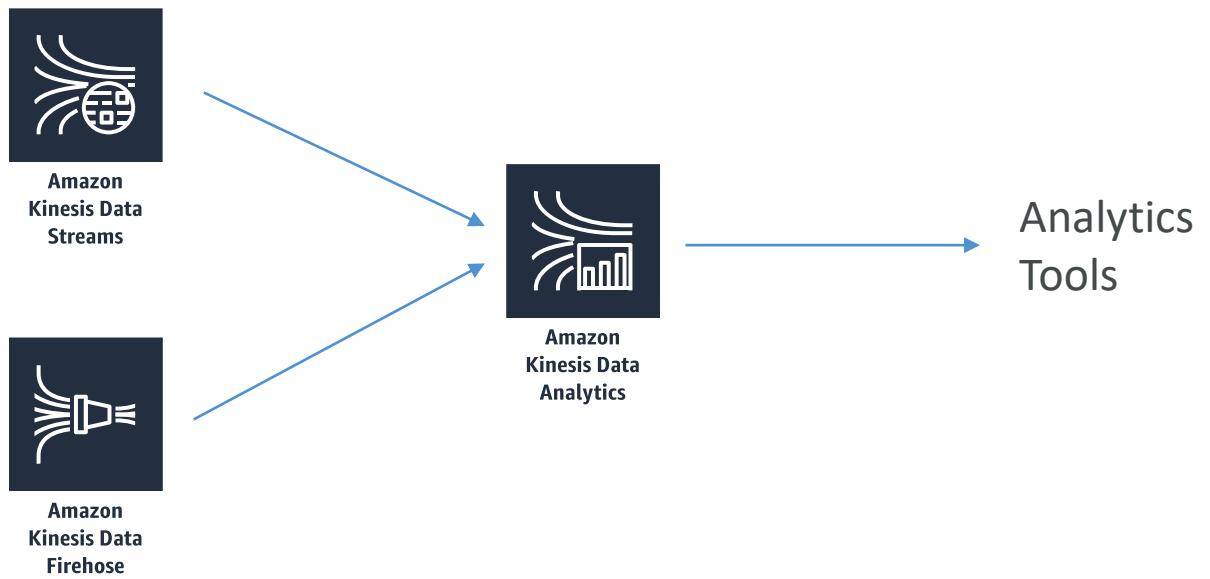
```

graph TD
    Start((Start)) --> Submit[Submit Batch Job]
    Submit --> Success[Notify Success]
    Submit --> Failure[Notify Failure]
    Success --> End((End))
    Failure --> End
    
```

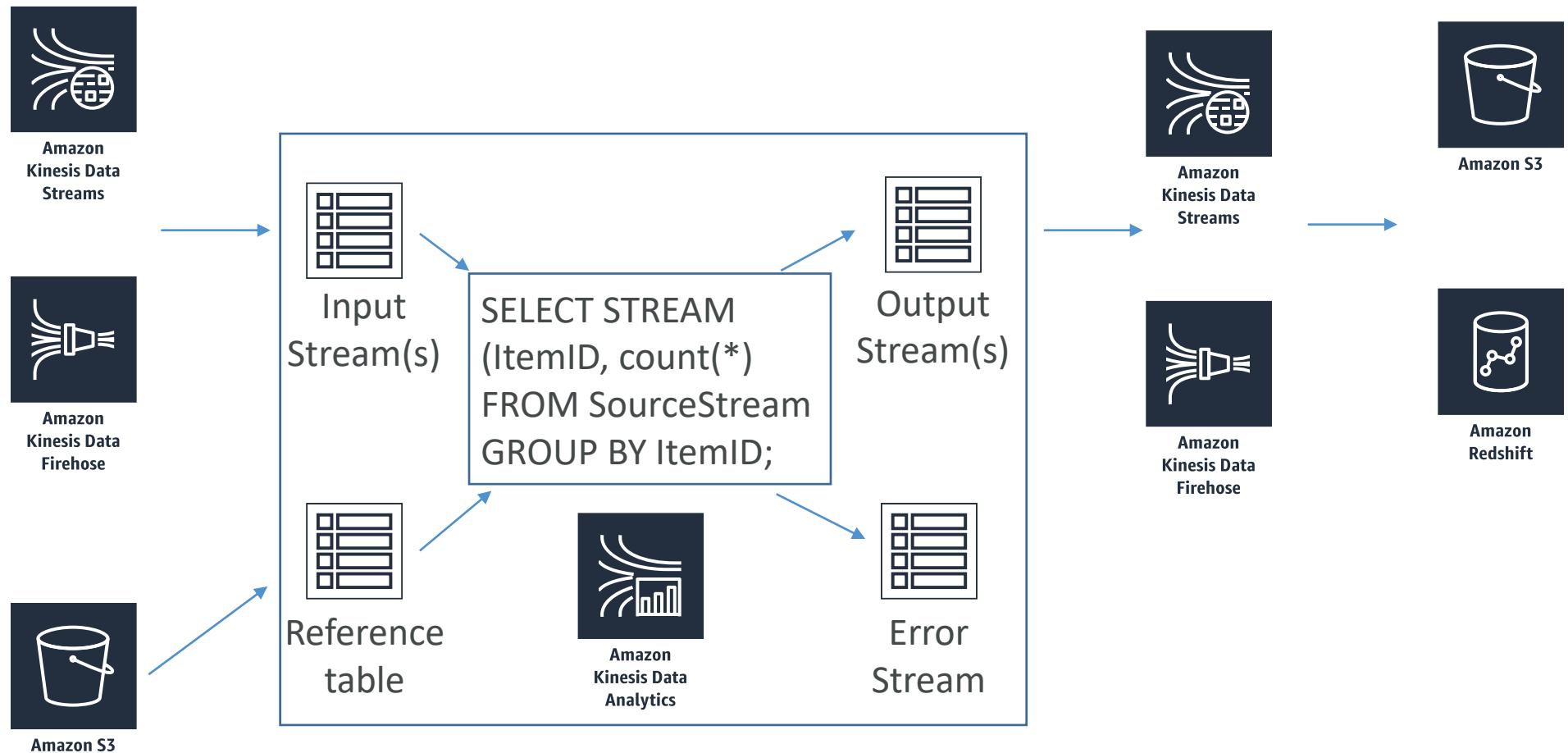
# Kinesis Analytics

Querying streams of data

# Conceptually...



# In more depth...



# Common use-cases

- Streaming ETL
- Continuous metric generation
- Responsive analytics



**Amazon  
Kinesis Data  
Analytics**

# Kinesis Analytics

- Pay only for resources consumed (but it's not cheap)
- Serverless; scales automatically
- Use IAM permissions to access streaming source and destination(s)
- Schema discovery

# RANDOM\_CUT\_FOREST

- SQL function used for anomaly detection on numeric columns in a stream
- They're especially proud of this because they published a paper on it
- It's a novel way to identify outliers in a data set so you can handle them however you need to
- Example: detect anomalous subway ridership during the NYC marathon

---

## Robust Random Cut Forest Based Anomaly Detection On Streams

---

**Sudipto Guha**  
University of Pennsylvania, Philadelphia, PA 19104.

SUDIPTO@CIS.UPENN.EDU

**Nina Mishra**  
Amazon, Palo Alto, CA 94303.

NMISHRA@AMAZON.COM

**Gourav Roy**  
Amazon, Bangalore, India 560055.

GOURAVR@AMAZON.COM

**Okke Schrijvers**  
Stanford University, Palo Alto, CA 94305.

OKKES@CS.STANFORD.EDU

### Abstract

In this paper we focus on the anomaly detection problem for dynamic data streams through the lens of random cut forests. We investigate a robust random cut data structure that can be used as a sketch or synopsis of the input stream. We provide a plausible definition of non-parametric anomalies based on the influence of an unseen point on the remainder of the data, i.e., the externalities of the data points. We show how the sketch can be efficiently updated in a dynamic data stream. We demonstrate the viability of the algorithm on publicly available real data.

### 1. Introduction

Anomaly detection is one of the cornerstone problems in data mining. Even though the problem has been well studied over the last few decades, the emerging explosion of data from the internet of things and sensors lead us to reconsider the problem. In most of these contexts, data is streaming and well-understood prior models do not exist. Furthermore the input streams need not be append-only; there may be corrections, updates and a variety of other dynamic changes. Two central questions in this regard are (1) how do we define anomalies? and (2) what data struc-

a point is data dependent and corresponds to the externalities induced by the point in according to the model of the data. We extend this notion of externality to handle “outlier masking” that often arises from duplicates and near duplicate records. Note that the notion of model complexity has to be amenable to efficient computation in dynamic data streams. This relates question (1) to question (2) which we discuss in greater detail next. However it is worth noting that anomaly detection is not well understood even in the simpler context of static batch processing and (2) remains relevant in the batch setting as well.

For question (2), we explore a randomized approach, akin to (Liu et al., 2012), due in part to the practical success reported in (Emmott et al., 2013). Randomization is a powerful tool and known to be valuable in supervised learning (Breiman, 2001). But its technical exploration in the context of anomaly detection is not well-understood and the same comment applies to the algorithm put forth in (Liu et al., 2012). Moreover that algorithm has several limitations as described in Section 4.1. In particular, we show that the presence of important dimensions, crucial for outlier detection, makes it difficult to extend the algorithm to a stream. Prior work attempted to extend this work to a stream. Prior work attempted to solutions (Tan et al., 2011) that extend to streaming, however those were not found to be effective (Emmott et al., 2013). To address these limitations, we put forward a sketch or synopsis termed *robust random cut forest* (RRCF) formally

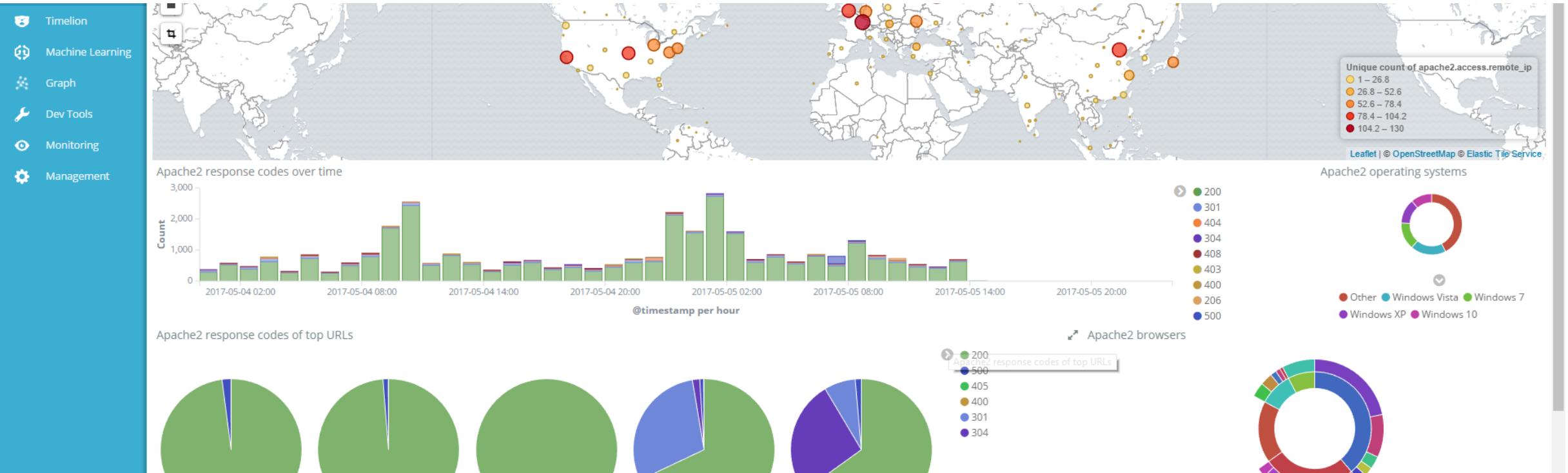
# Amazon Elasticsearch Service

Petabyte-scale analysis and reporting

# What is Elasticsearch?

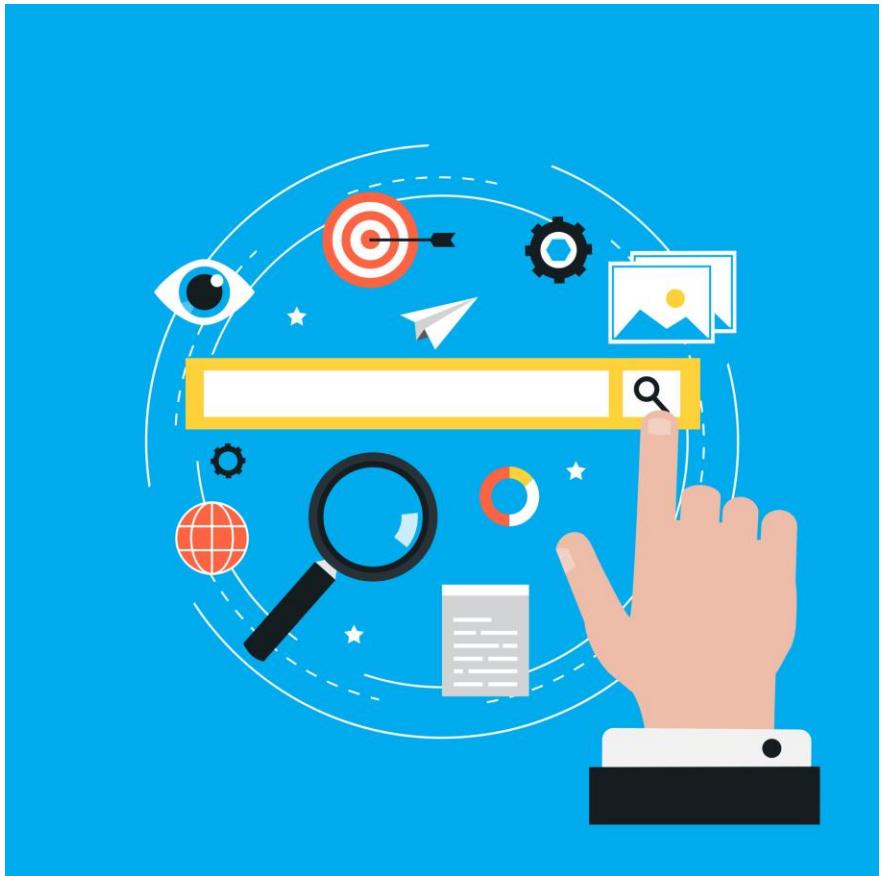
- The Elastic Stack
- A search engine
- An analysis tool
- A visualization tool (Kibana)
- A data pipeline (Beats / Logstash)
  - You can use Kinesis too
- Horizontally scalable

# What is Kibana?



# Elasticsearch applications

- Full-text search
- Log analytics
- Application monitoring
- Security analytics
- Clickstream analytics



# Elasticsearch concepts



## documents

Documents are the things you're searching for. They can be more than text – any structured JSON data works. Every document has a unique ID, and a type.



## types

A type defines the schema and mapping shared by documents that represent the same sort of thing. (A log entry, an encyclopedia article, etc.)

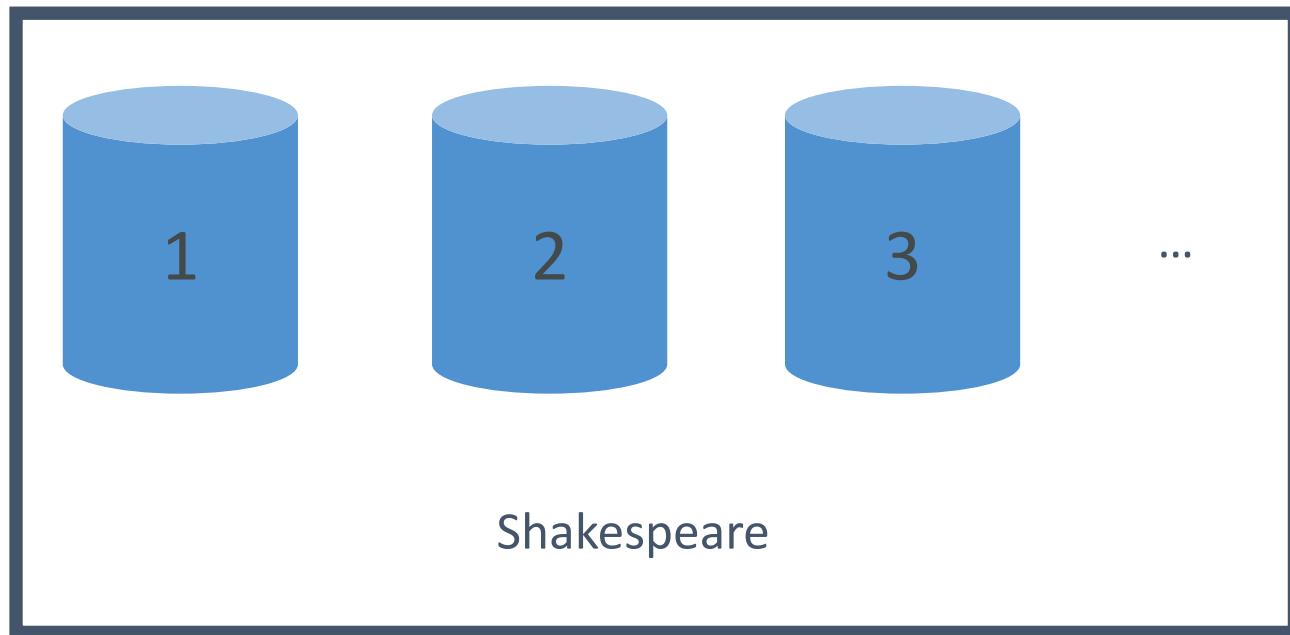


## indices

An index powers search into all documents within a collection of types. They contain inverted indices that let you search across everything within them at once.

# An index is split into shards

Documents are **hashed** to a particular shard.



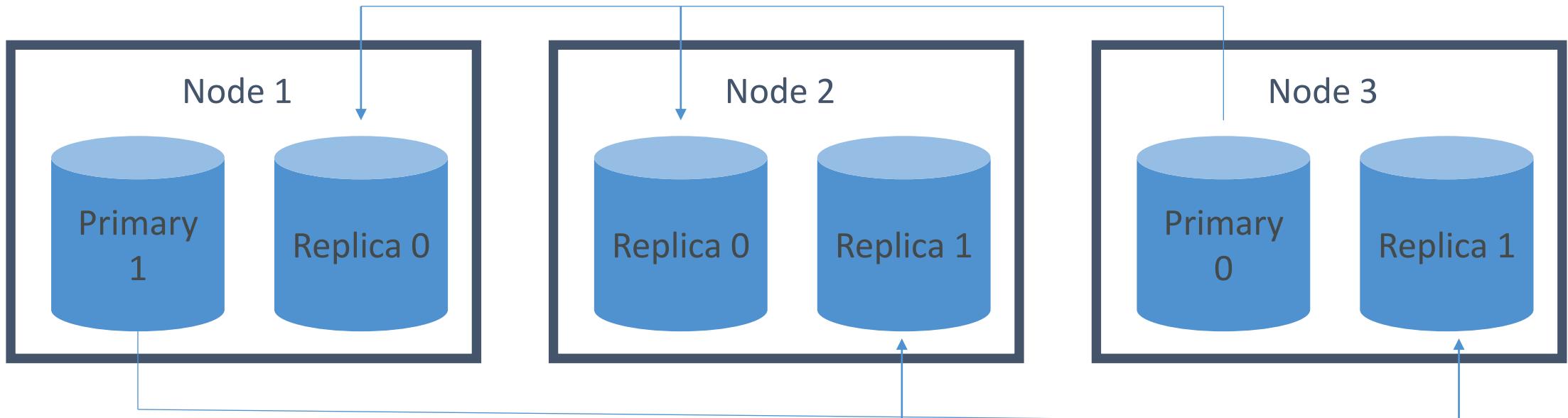
Each shard may be on a different **node** in a cluster.

Every shard is a self-contained Lucene index of its own.

# Redundancy

This index has two primary shards and two replicas.

Your application should round-robin requests amongst nodes.



Write requests are routed to the primary shard, then replicated

Read requests are routed to the primary or any replica

# Amazon Elasticsearch Service

- Fully-managed (but not serverless)
- Scale up or down without downtime
  - But this isn't automatic
- Pay for what you use
  - Instance-hours, storage, data transfer
- Network isolation
- AWS integration
  - S3 buckets (via Lambda to Kinesis)
  - Kinesis Data Streams
  - DynamoDB Streams
  - CloudWatch / CloudTrail
  - Zone awareness

# Amazon ES options

- Dedicated master node(s)
  - Choice of count and instance types
- “Domains”
- Snapshots to S3
- Zone Awareness

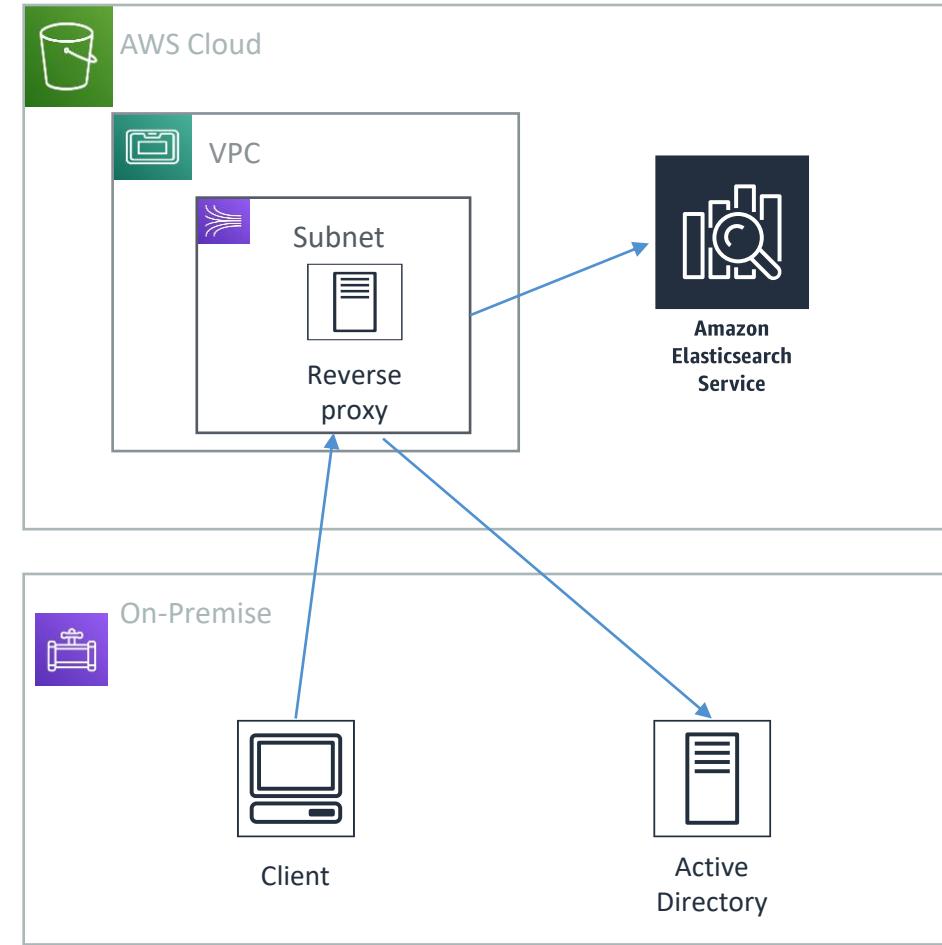
# Amazon ES Security

- Resource-based policies
- Identity-based policies
- IP-based policies
- Request signing
- VPC
- Cognito



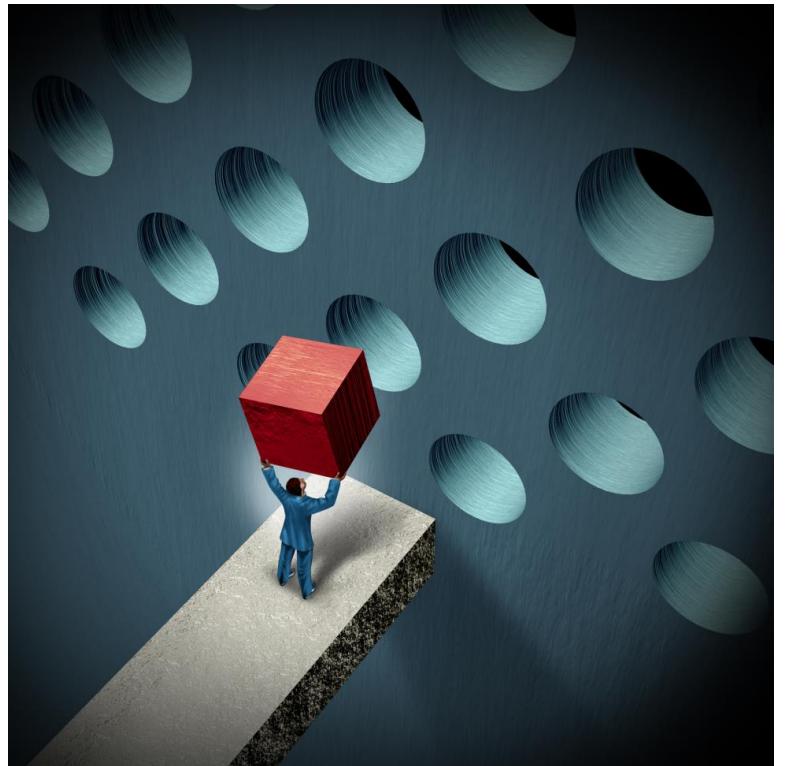
# Securing Kibana

- Cognito
- Getting inside a VPC from outside is hard...
  - Nginx reverse proxy on EC2 forwarding to ES domain
  - SSH tunnel for port 5601
  - VPC Direct Connect
  - VPN



# Amazon ES anti-patterns

- OLTP
  - No transactions
  - RDS or DynamoDB is better
- Ad-hoc data querying
  - Athena is better
- Remember Amazon ES is primarily for search & analytics



# Amazon Athena

Serverless interactive queries of S3 data

# What is Athena?

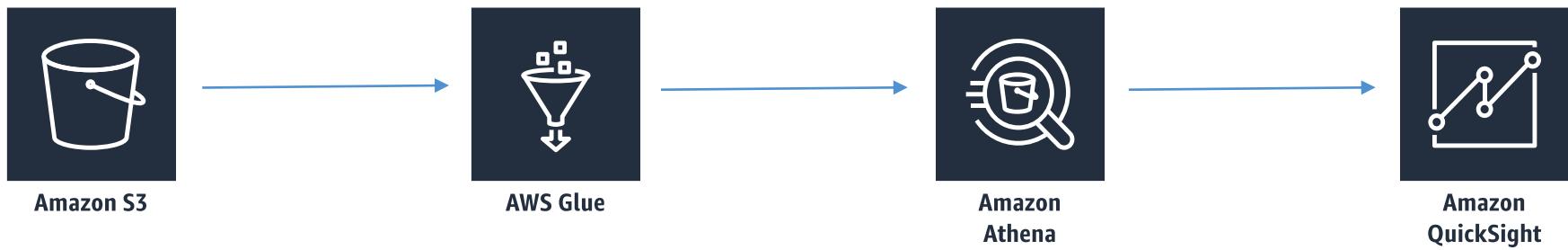
- Interactive query service for S3 (SQL)
  - No need to load data, it stays in S3
- Presto under the hood
- Serverless!
- Supports many data formats
  - CSV (human readable)
  - JSON (human readable)
  - ORC (columnar, splittable)
  - Parquet (columnar, splittable)
  - Avro (splittable)
- Unstructured, semi-structured, or structured

# Some examples

- Ad-hoc queries of web logs
- Querying staging data before loading to Redshift
- Analyze CloudTrail / CloudFront / VPC / ELB etc logs in S3
- Integration with Jupyter, Zeppelin, RStudio notebooks
- Integration with QuickSight
- Integration via ODBC / JDBC with other visualization tools

```
43 USE AdventureworksLT;
44 GO
45 SELECT p.Name AS ProductName,
46 NonDiscountSales = (OrderQty * UnitPrice)
47 Discounts = ((OrderQty * UnitPrice) * TaxRate)
48 FROM Production.Product AS p
49 INNER JOIN Sales.SalesOrderDetail sod
50 ON p.ProductID = sod.ProductID
51 ORDER BY ProductName DESC;
52 GO
```

# Athena + Glue



# Athena cost model

- Pay-as-you-go
  - \$5 per TB scanned
  - Successful or cancelled queries count, failed queries do not.
  - No charge for DDL (CREATE/ALTER/DROP etc.)
- Save LOTS of money by using columnar formats
  - ORC, Parquet
  - Save 30-90%, and get better performance
- Glue and S3 have their own charges



# Athena Security

- Access control
  - IAM, ACLs, S3 bucket policies
  - AmazonAthenaFullAccess / AWSQuicksightAthenaAccess
- Encrypt results at rest in S3 staging directory
  - Server-side encryption with S3-managed key (SSE-S3)
  - Server-side encryption with KMS key (SSE-KMS)
  - Client-side encryption with KMS key (CSE-KMS)
- Cross-account access in S3 bucket policy possible
- Transport Layer Security (TLS) encrypts in-transit (between Athena and S3)



# Athena anti-patterns

- Highly formatted reports / visualization
  - That's what QuickSight is for
- ETL
  - Use Glue instead



# Amazon Redshift

Fully-managed, petabyte-scale data warehouse

# What is Redshift?

- Fully-managed, petabyte scale data warehouse service
- 10X better performance than other DW's
  - Via machine learning, massively parallel query execution, columnar storage
- Designed for OLAP, not OLTP
- Cost effective
- SQL, ODBC, JDBC interfaces
- Scale up or down on demand
- Built-in replication & backups
- Monitoring via CloudWatch / CloudTrail

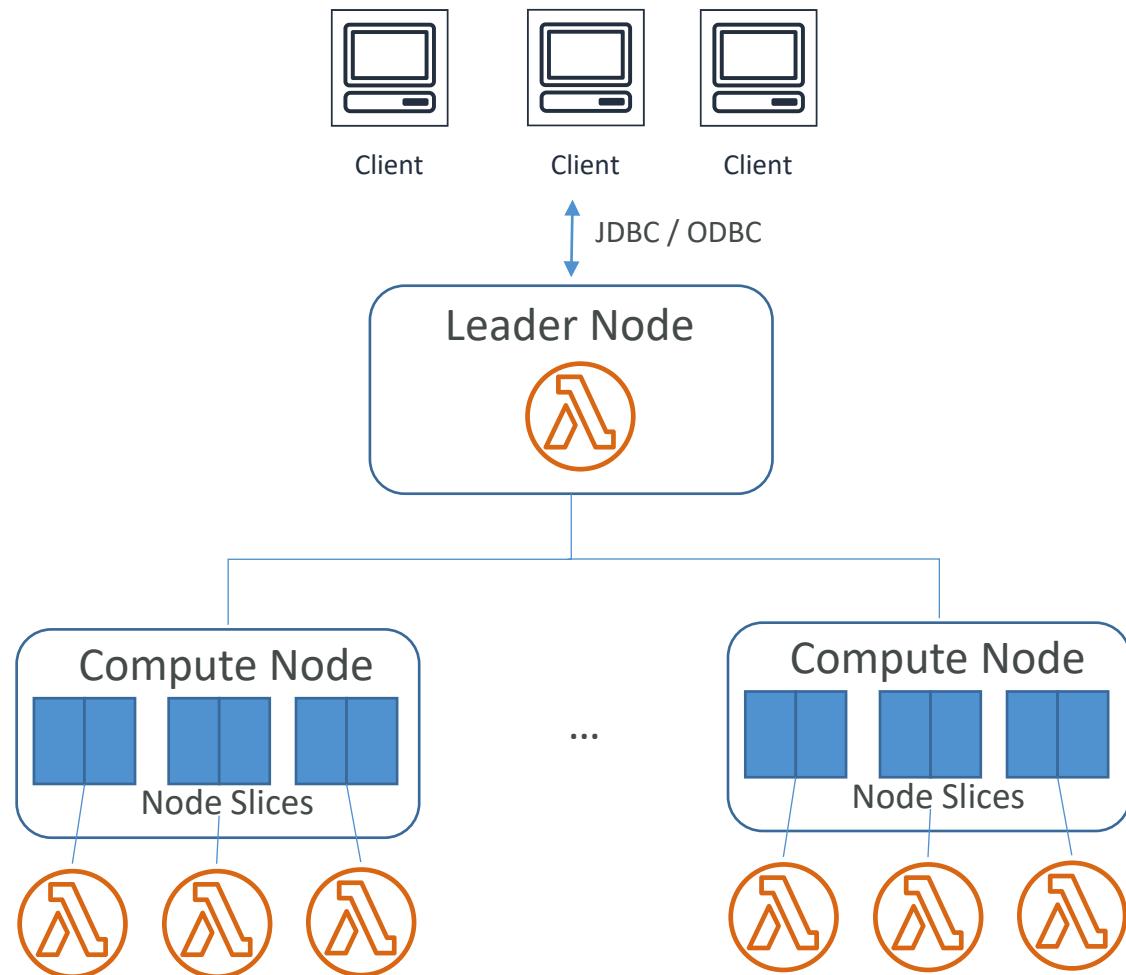


**Amazon  
Redshift**

# Redshift Use-Cases

- Accelerate analytics workloads
- Unified data warehouse & data lake
- Data warehouse modernization
- Analyze global sales data
- Store historical stock trade data
- Analyze ad impressions & clicks
- Aggregate gaming data
- Analyze social trends

# Redshift architecture



# Redshift Spectrum

- Query exabytes of unstructured data in S3 without loading
- Limitless concurrency
- Horizontal scaling
- Separate storage & compute resources
- Wide variety of data formats
- Support of Gzip and Snappy compression



Amazon  
Redshift



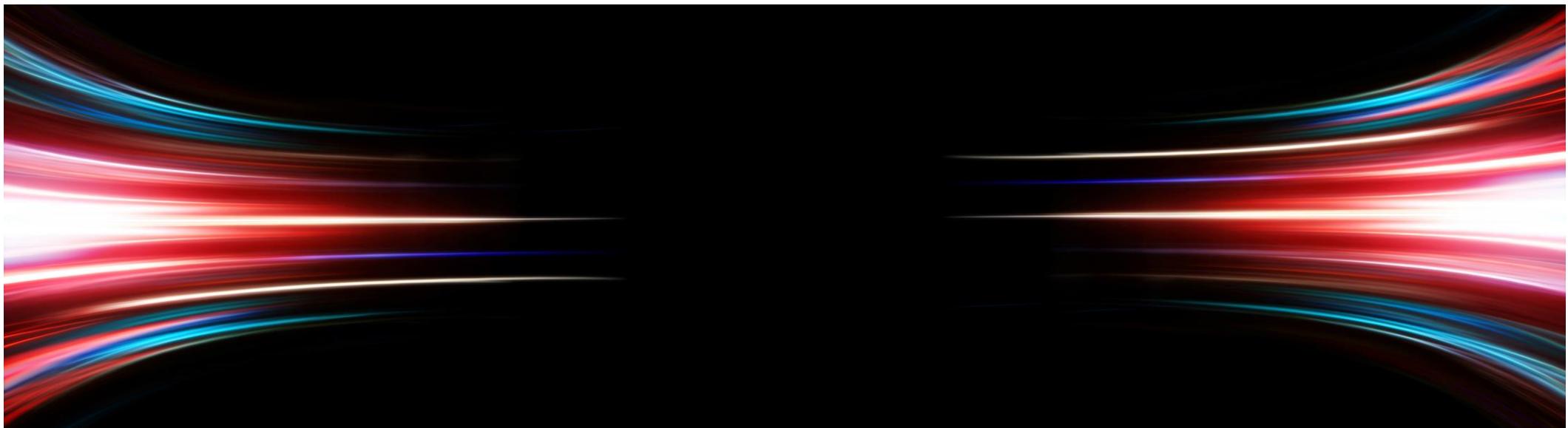
AWS Glue



Amazon S3

# Redshift Performance

- Massively Parallel Processing (MPP)
- Columnar Data Storage
- Column Compression



# Redshift Durability

- Replication within cluster
- Backup to S3
  - Asynchronously replicated to another region
- Automated snapshots
- Failed drives / nodes automatically replaced
- However – limited to a single availability zone (AZ)



# Scaling Redshift

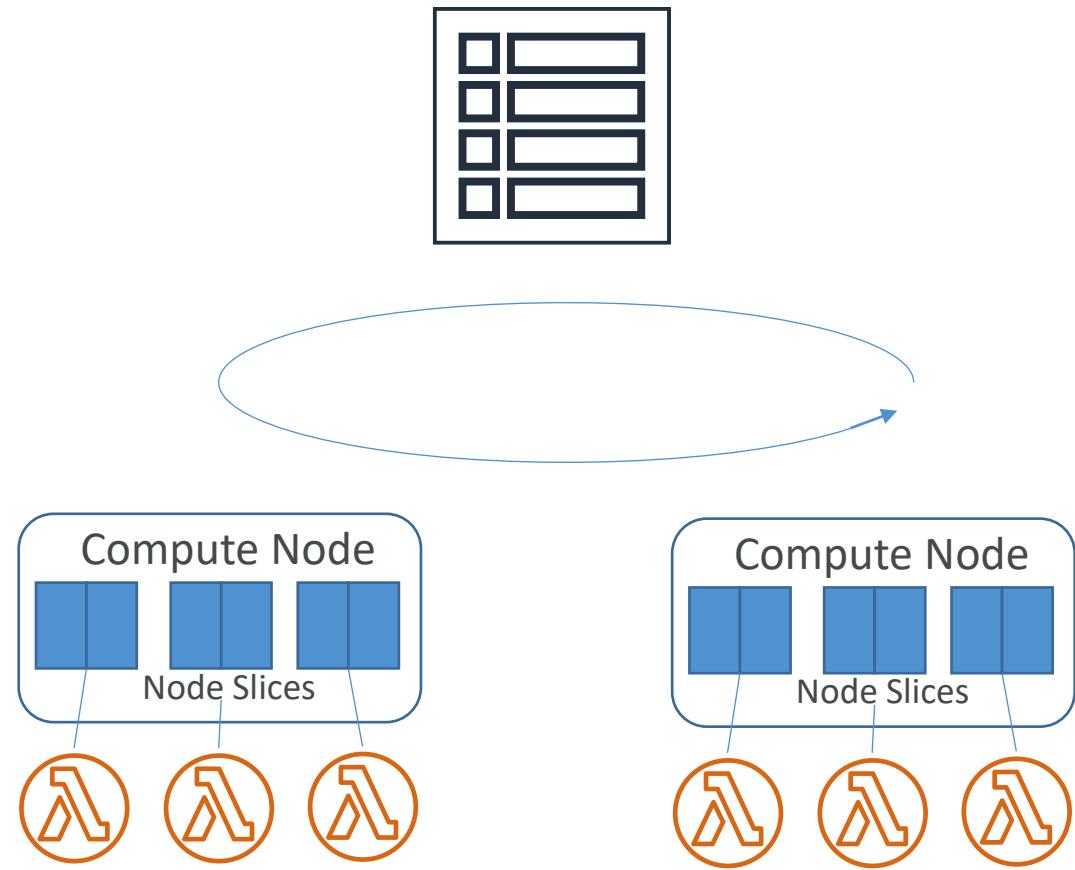
- Vertical and horizontal scaling on demand
- During scaling:
  - A new cluster is created while your old one remains available for reads
  - CNAME is flipped to new cluster (a few minutes of downtime)
  - Data moved in parallel to new compute nodes



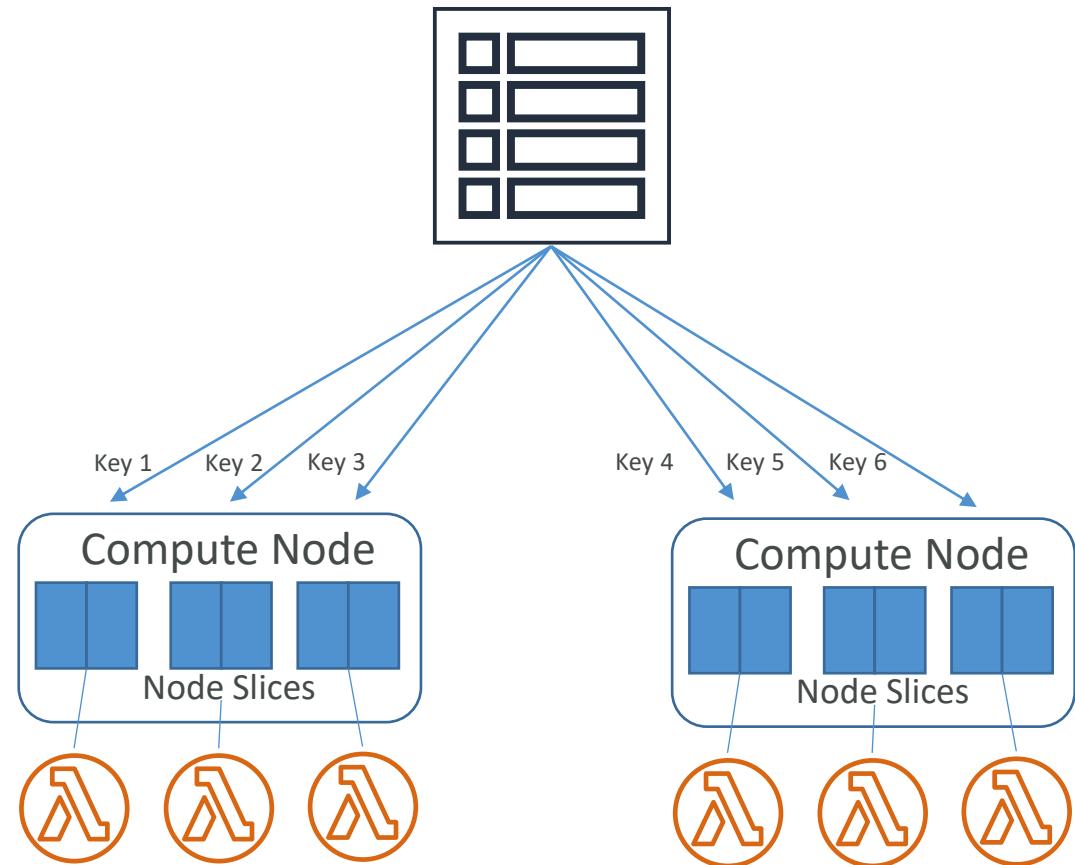
# Redshift Distribution Styles

- AUTO
  - Redshift figures it out based on size of data
- EVEN
  - Rows distributed across slices in round-robin
- KEY
  - Rows distributed based on one column
- ALL
  - Entire table is copied to every node

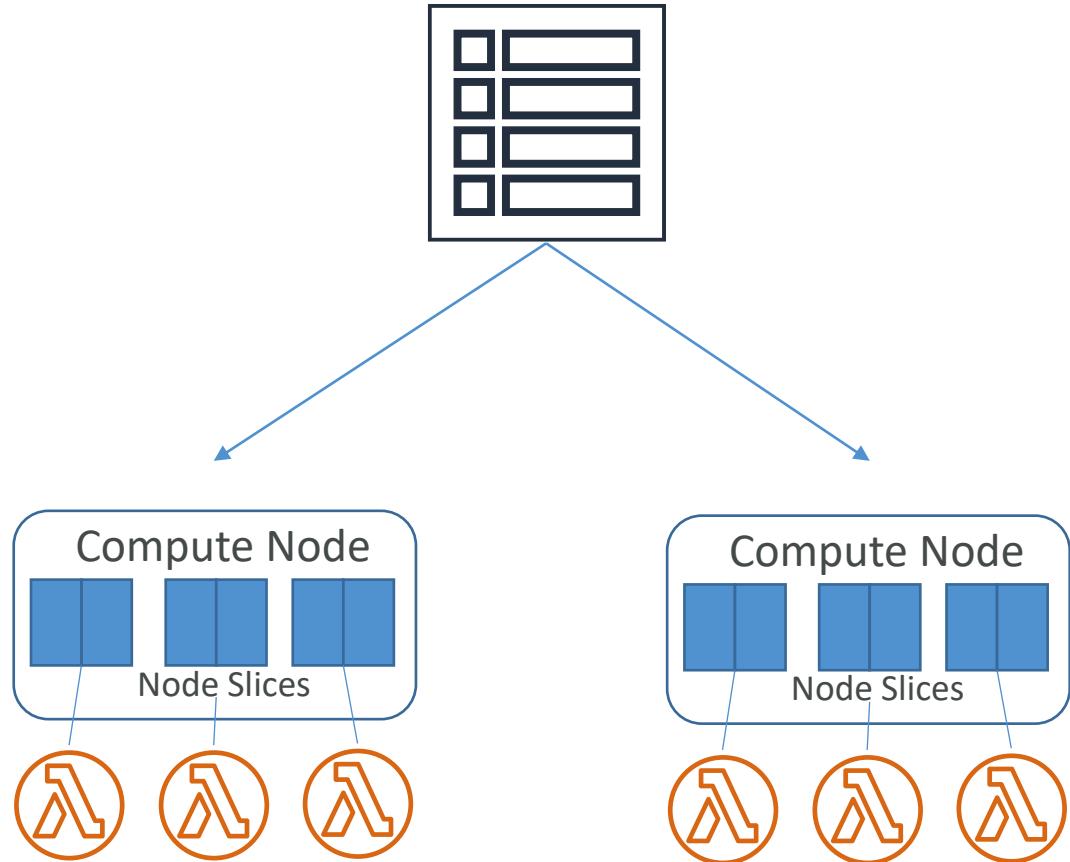
# EVEN distribution



# KEY distribution



# ALL distribution



# Redshift Sort Keys

- Rows are stored on disk in sorted order based on the column you designate as a sort key
- Like an index
- Makes for fast range queries
- Choosing a sort key
  - Recency? Filtering? Joins?
- Single vs. Compound vs Interleaved sort keys



# Sort Keys: Single Column

Date	Genre	Movie
3/18/2019	Comedy	Monty Python and the Holy Grail
3/18/2019	Adventure	Indiana Jones and the Temple of Doom
3/18/2019	Drama	Interstellar
3/18/2019	Drama	The Dark Knight
3/19/2019	Fantasy	The Lord of the Rings
3/19/2019	Drama	12 Angry Men
3/19/2019	Adventure	Inception

# Sort Keys: Compound

Date	Genre	Movie
3/18/2019	Adventure	Indiana Jones and the Temple of Doom
3/18/2019	Comedy	Monty Python and the Holy Grail
3/18/2019	Drama	Interstellar
3/18/2019	Drama	The Dark Knight
3/19/2019	Adventure	Inception
3/19/2019	Drama	12 Angry Men
3/19/2019	Fantasy	The Lord of the Rings

# Sort Keys: Interleaved

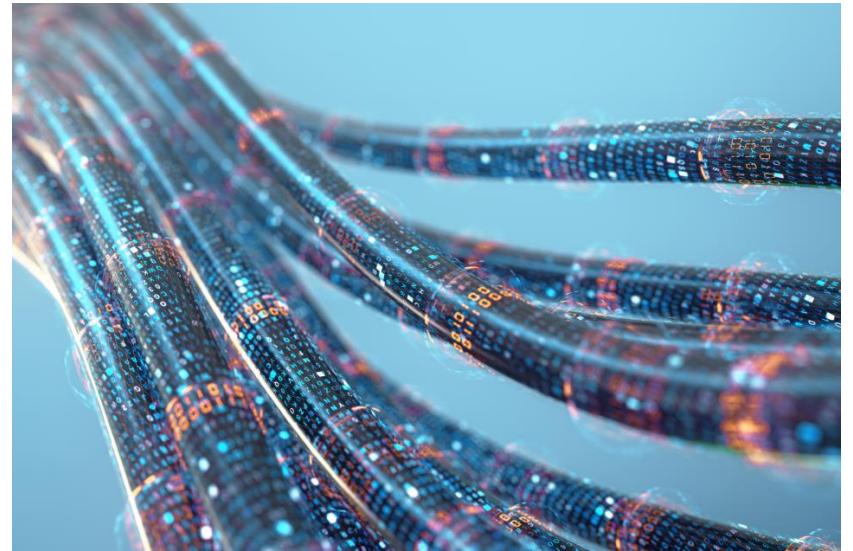
Date	Genre	Movie
3/18/2019	Adventure	Indiana Jones and the Temple of Doom
3/18/2019	Comedy	Monty Python and the Holy Grail
3/18/2019	Drama	Interstellar
3/18/2019	Drama	The Dark Knight
3/19/2019	Adventure	Inception
3/19/2019	Drama	12 Angry Men
3/19/2019	Fantasy	The Lord of the Rings

Date	Genre	Movie
3/19/2019	Drama	12 Angry Men
3/19/2019	Adventure	Inception
3/18/2019	Adventure	Indiana Jones and the Temple of Doom
3/18/2019	Drama	Interstellar
3/18/2019	Comedy	Monty Python and the Holy Grail
3/18/2019	Drama	The Dark Knight
3/19/2019	Fantasy	The Lord of the Rings

Date	Genre	Movie
3/18/2019	Adventure	Indiana Jones and the Temple of Doom
3/19/2019	Adventure	Inception
3/18/2019	Comedy	Monty Python and the Holy Grail
3/18/2019	Drama	Interstellar
3/18/2019	Drama	The Dark Knight
3/19/2019	Drama	12 Angry Men
3/19/2019	Fantasy	The Lord of the Rings

# Importing / Exporting data

- COPY command
  - Parallelized; efficient
  - From S3, EMR, DynamoDB, remote hosts
  - S3 requires a manifest file and IAM role
- UNLOAD command
  - Unload from a table into files in S3
- Enhanced VPC routing



# COPY command: More depth

- Use COPY to load large amounts of data from outside of Redshift
- If your data is already in Redshift in another table,
  - Use INSERT INTO ...SELECT
  - Or CREATE TABLE AS
- COPY can decrypt data as it is loaded from S3
  - Hardware-accelerated SSL used to keep it fast
- Gzip, Izop, and bzip2 compression supported to speed it up further
- Automatic compression option
  - Analyzes data being loaded and figures out optimal compression scheme for storing it
- Special case: narrow tables (lots of rows, few columns)
  - Load with a single COPY transaction if possible
  - Otherwise hidden metadata columns consume too much space

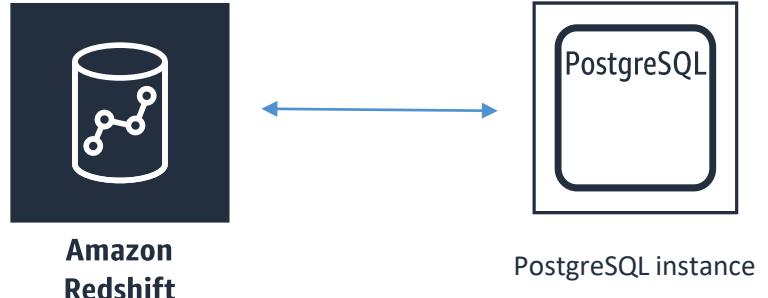


# Redshift copy grants for cross-region snapshot copies

- Let's say you have a KMS-encrypted Redshift cluster and a snapshot of it
- You want to copy that snapshot to another region for backup
- In the destination AWS region:
  - Create a KMS key if you don't have one already
  - Specify a unique name for your snapshot copy grant
  - Specify the KMS key ID for which you're creating the copy grant
- In the source AWS region:
  - Enable copying of snapshots to the copy grant you just created

# DBLINK

- Connect Redshift to PostgreSQL (possibly in RDS)
- Good way to copy and sync data between PostgreSQL and Redshift



PostgreSQL instance

```
CREATE EXTENSION postgres_fdw;
CREATE EXTENSION dblink;
CREATE SERVER foreign_server
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host '<amazon_redshift_ip>', port '<port>', dbname '<database_name>', sslmode
'require');
CREATE USER MAPPING FOR <rds_postgresql_username>
    SERVER foreign_server
    OPTIONS (user '<amazon_redshift_username>', password '<password>');
```

# Integration with other services

- S3
- DynamoDB
- EMR / EC2
- Data Pipeline
- Database Migration Service



Amazon S3



AWS Database  
Migration  
Service



Amazon  
DynamoDB



Amazon EMR

# Redshift Workload Management (WLM)

- Prioritize short, fast queries vs. long, slow queries
- Query queues
- Via console, CLI, or API

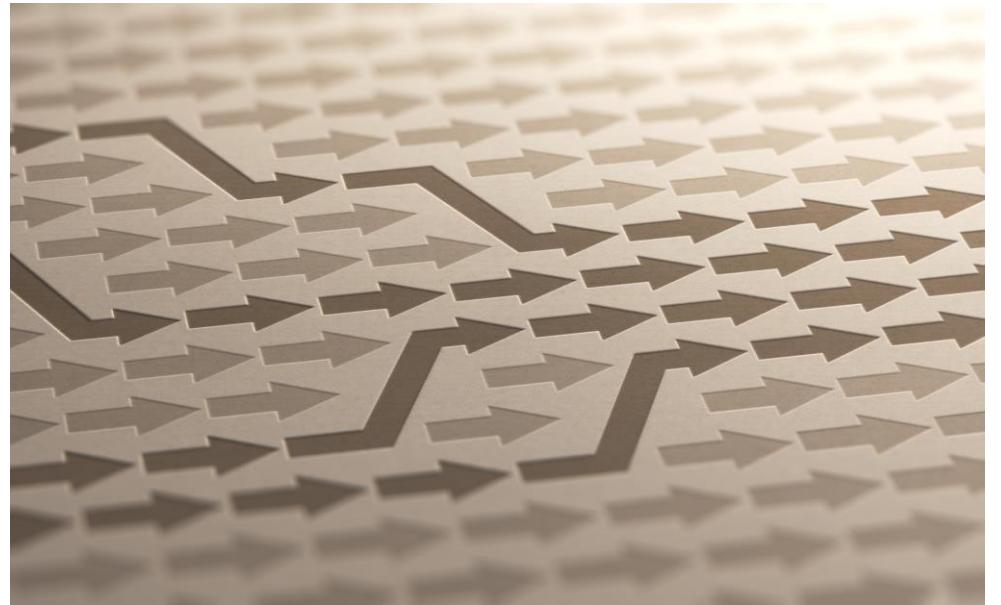
# Concurrency Scaling

- Automatically adds cluster capacity to handle increase in concurrent **read** queries
- Support virtually unlimited concurrent users & queries
- WLM queues manage which queries are sent to the concurrency scaling cluster



# Automatic Workload Management

- Creates up to 8 queues
- Default 5 queues with even memory allocation
- Large queries (ie big hash joins) -> concurrency lowered
- Small queries (ie inserts, scans, aggregations) -> concurrency raised
- Configuring query queues
  - Priority
  - Concurrency scaling mode
  - User groups
  - Query groups
  - Query monitoring rules



# Manual Workload Management

- One default queue with concurrency level of 5 (5 queries at once)
- Superuser queue with concurrency level 1
- Define up to 8 queues, up to concurrency level 50
  - Each can have defined concurrency scaling mode, concurrency level, user groups, query groups, memory, timeout, query monitoring rules
  - Can also enable query queue hopping
    - Timed out queries “hop” to next queue to try again

# Short Query Acceleration (SQA)

- Prioritize short-running queries over longer-running ones
- Short queries run in a dedicated space, won't wait in queue behind long queries
- Can be used in place of WLM queues for short queries
- Works with:
  - CREATE TABLE AS (CTAS)
  - Read-only queries (SELECT statements)
- Uses machine learning to predict a query's execution time
- Can configure how many seconds is “short”

# Resizing Redshift Clusters

- Elastic resize
  - Quickly add or remove nodes of same type
    - (It *\*can\** change node types, but not without dropping connections – it creates a whole new cluster)
  - Cluster is down for a few minutes
  - Tries to keep connections open across the downtime
  - Limited to doubling or halving for some dc2 and ra3 node types.
- Classic resize
  - Change node type and/or number of nodes
  - Cluster is read-only for hours to days
- Snapshot, restore, resize
  - Used to keep cluster available during a classic resize
  - Copy cluster, resize new cluster



# VACUUM command

- Recovers space from deleted rows
- VACUUM FULL
- VACUUM DELETE ONLY
- VACUUM SORT ONLY
- VACUUM REINDEX

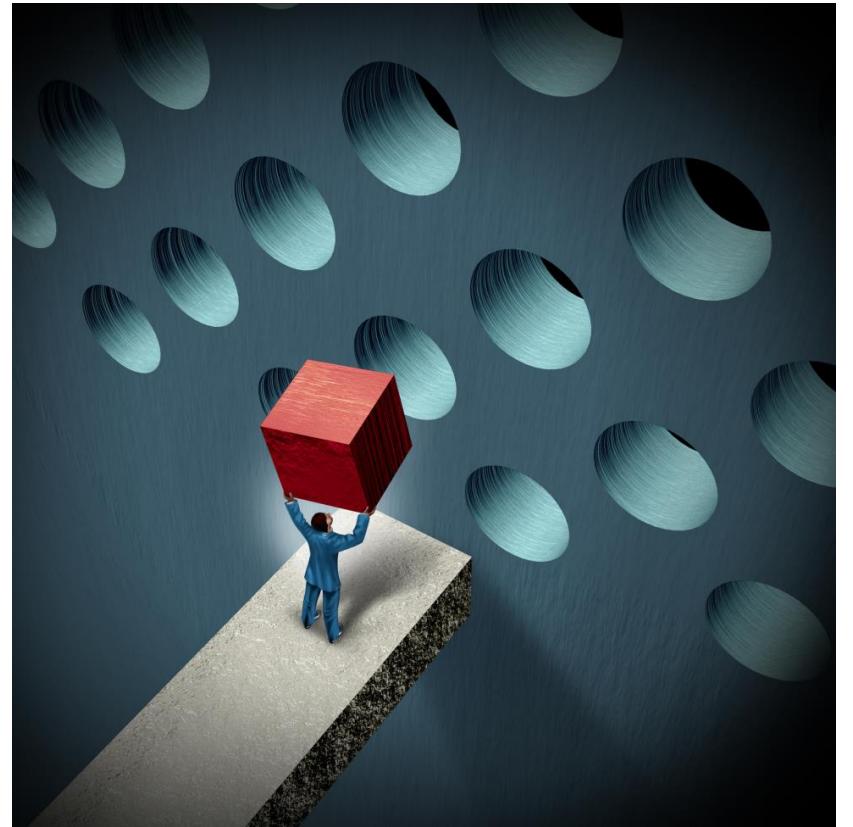


# New Redshift features for 2020

- RA3 nodes with managed storage
  - Enable independent scaling of compute and storage
- Redshift data lake export
  - Unload Redshift query to S3 in Apache Parquet format
  - Parquet is 2x faster to unload and consumes up to 6X less storage
  - Compatible with Redshift Spectrum, Athena, EMR, SageMaker
  - Automatically partitioned

# Redshift anti-patterns

- Small data sets
  - Use RDS instead
- OLTP
  - Use RDS or DynamoDB instead
- Unstructured data
  - ETL first with EMR etc.
- BLOB data
  - Store references to large binary files in S3, not the files themselves.



# Amazon RDS

## Relational Database Service

# What is RDS?

- Hosted relational database
  - Amazon Aurora
  - MySQL
  - PostgreSQL
  - MariaDB
  - Oracle
  - SQL Server
- Not for “big data”
  - Might appear on exam as an example of what not to use
  - Or in the context of migrating from RDS to Redshift etc.



**Amazon RDS**

# ACID

- RDS databases offer full ACID compliance
  - Atomicity
  - Consistency
  - Isolation
  - Durability



# Amazon Aurora

- MySQL and PostgreSQL – compatible
- Up to 5X faster than MySQL, 3X faster than PostgreSQL
- 1/10 the cost of commercial databases
- Up to 64TB per database instance
- Up to 15 read replicas
- Continuous backup to S3
- Replication across availability zones
- Automatic scaling with Aurora Serverless

# Aurora Security

- VPC network isolation
- At-rest with KMS
  - Data, backup, snapshots, and replicas can be encrypted
- In-transit with SSL



# Amazon QuickSight

Business analytics and visualizations in the cloud

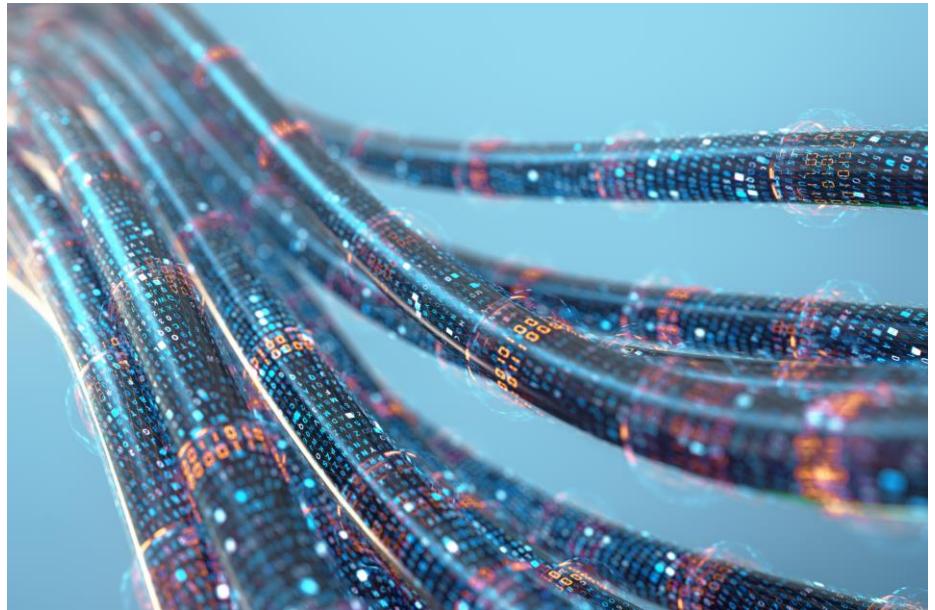
# What is QuickSight?

- Fast, easy, cloud-powered business analytics service
- Allows all employees in an organization to:
  - Build visualizations
  - Perform ad-hoc analysis
  - Quickly get business insights from data
  - Anytime, on any device (browsers, mobile)
- Serverless



# QuickSight Data Sources

- Redshift
- Aurora / RDS
- Athena
- EC2-hosted databases
- Files (S3 or on-premises)
  - Excel
  - CSV, TSV
  - Common or extended log format
- Data preparation allows limited ETL



# SPICE

- Data sets are imported into SPICE
  - Super-fast, Parallel, In-memory Calculation Engine
  - Uses columnar storage, in-memory, machine code generation
  - Accelerates interactive queries on large datasets
- Each user gets 10GB of SPICE
- Highly available / durable
- Scales to hundreds of thousands of users

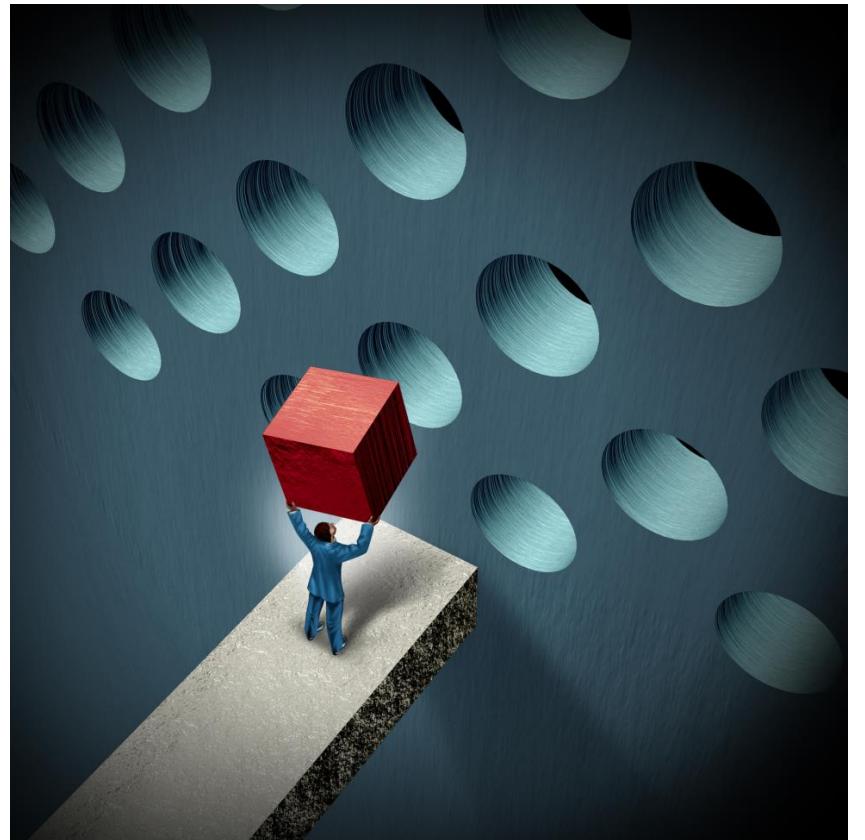


# QuickSight Use Cases

- Interactive ad-hoc exploration / visualization of data
- Dashboards and KPI's
- Stories
  - Guided tours through specific views of an analysis
  - Convey key points, thought process, evolution of an analysis
- Analyze / visualize data from:
  - Logs in S3
  - On-premise databases
  - AWS (RDS, Redshift, Athena, S3)
  - SaaS applications, such as Salesforce
  - Any JDBC/ODBC data source

# QuickSight Anti-Patterns

- Highly formatted canned reports
  - QuickSight is for ad-hoc queries, analysis, and visualization
- ETL
  - Use Glue instead, although QuickSight can do some transformations



# QuickSight Security

- Multi-factor authentication on your account
- VPC connectivity
  - Add QuickSight's IP address range to your database security groups
- Row-level security
- Private VPC access
  - Elastic Network Interface, AWS Direct Connect



# QuickSight User Management

- Users defined via IAM, or email signup
- Active Directory integration with QuickSight Enterprise Edition



# QuickSight Pricing

- Annual subscription
  - Standard: \$9 / user / month
  - Enterprise: \$18 / user / month
- Extra SPICE capacity (beyond 10GB)
  - \$0.25 (standard) \$0.38 (enterprise) / GB / user / month
- Month to month
  - Standard: \$12 / user / month
  - Enterprise: \$24 / user / month
- Enterprise edition
  - Encryption at rest
  - Microsoft Active Directory integration

# QuickSight Dashboards

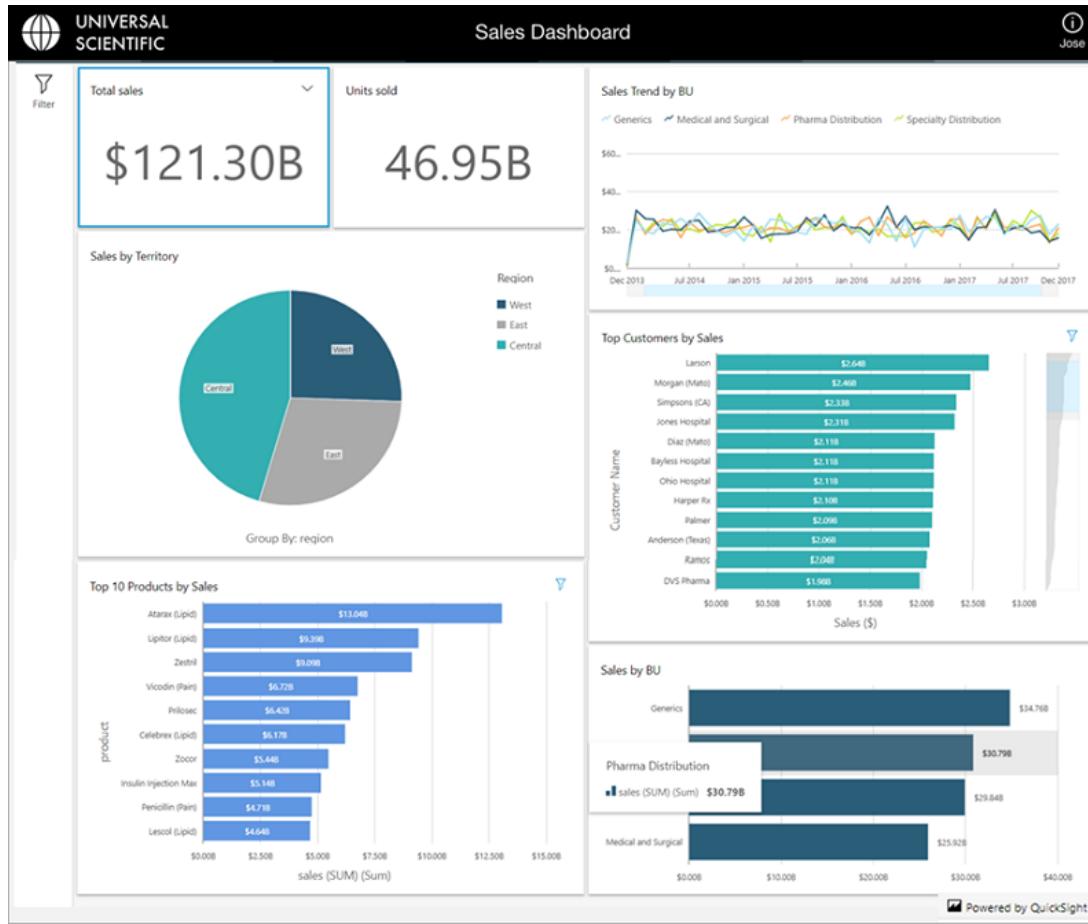
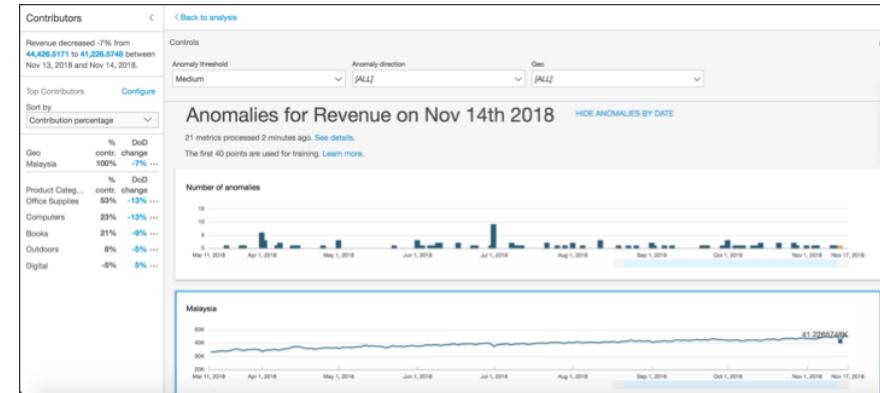


Image: AWS Big Data Blog

# Quicksight Machine Learning Insights

- ML-powered anomaly detection
  - Uses Random Cut Forest
  - Identify top contributors to significant changes in metrics
- ML-powered forecasting
  - Also uses Random Cut Forest
  - Detects seasonality and trends
  - Excludes outliers and imputes missing values
- Autonarratives
  - Adds “story of your data” to your dashboards
- Suggested Insights
  - “Insights” tab displays read-to-use suggested insights



Total Revenue for Nov 17, 2018 decreased by 2.15%  
**(-131,031.34720000066)** from 6,100,697.1616 to  
 5,969,665.8144. Compounded growth rate for the last 4 days is  
 -0.26% worse than expected.

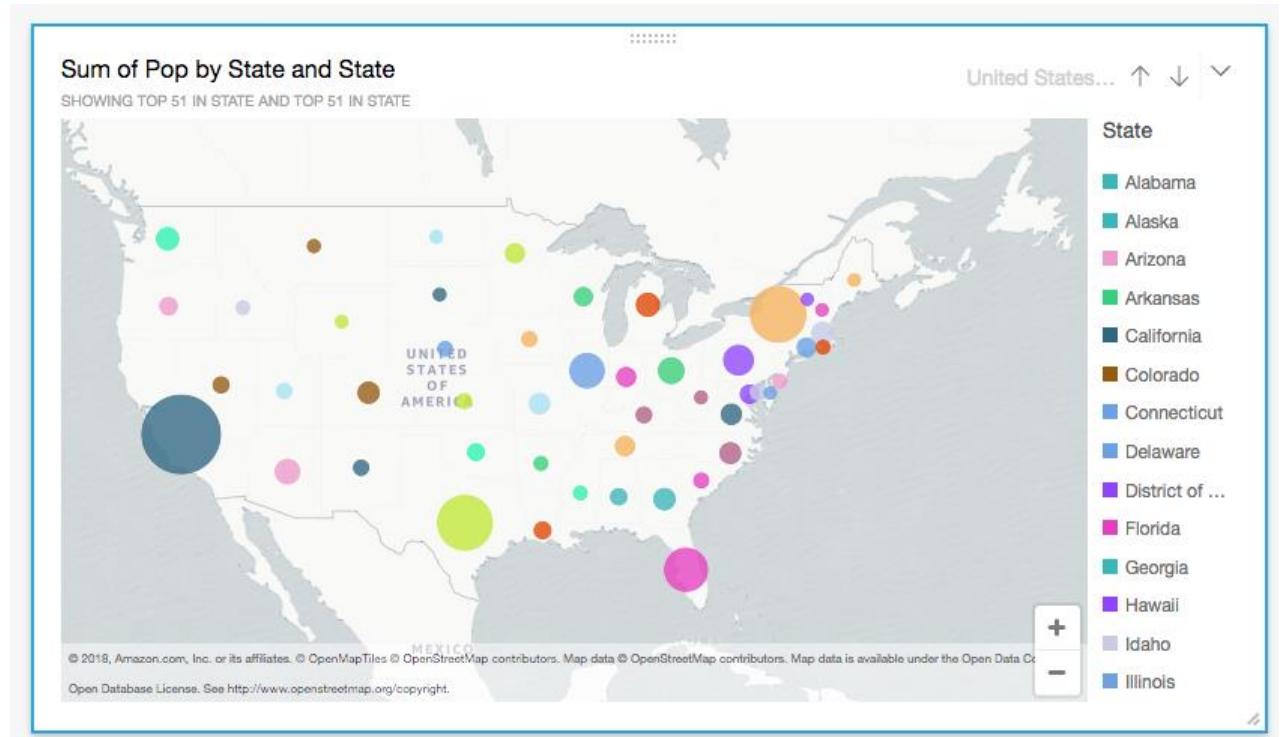
# QuickSight Visual Types

- AutoGraph
- Bar Charts
  - For comparison and distribution (histograms)
- Line graphs
  - For changes over time
- Scatter plots, heat maps
  - For correlation
- Pie graphs, tree maps
  - For aggregation
- Pivot tables
  - For tabular data
- Stories



# Additional Visual Types

- KPIs
- Geospatial Charts (maps)
- Donut Charts
- Gauge Charts
- Word Clouds



# Bar Charts: Comparison, Distribution

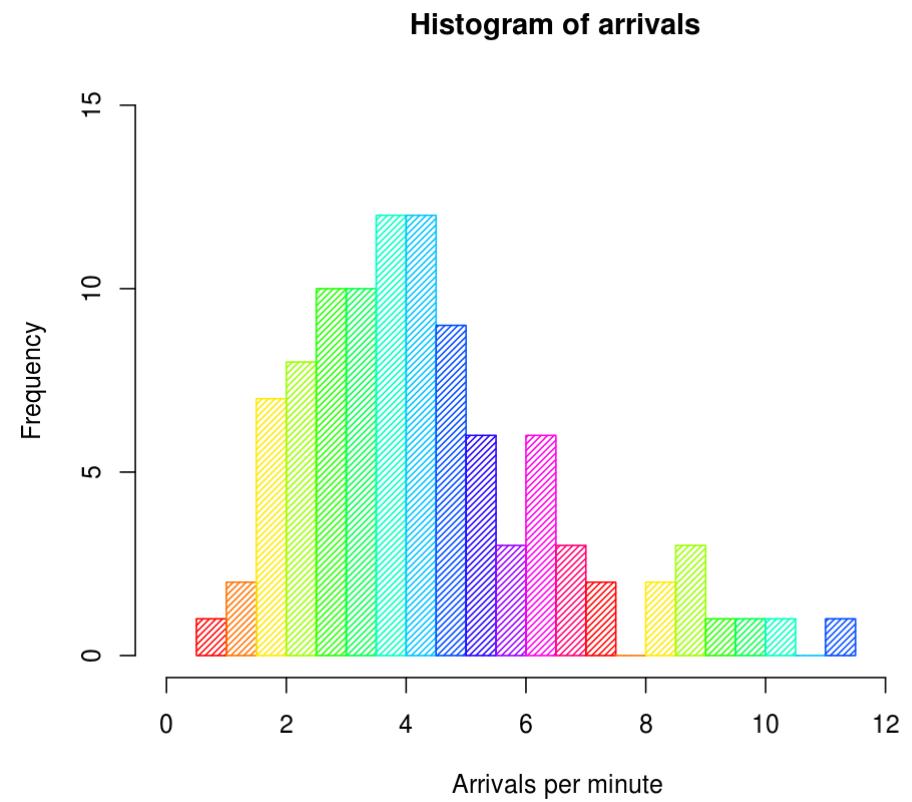
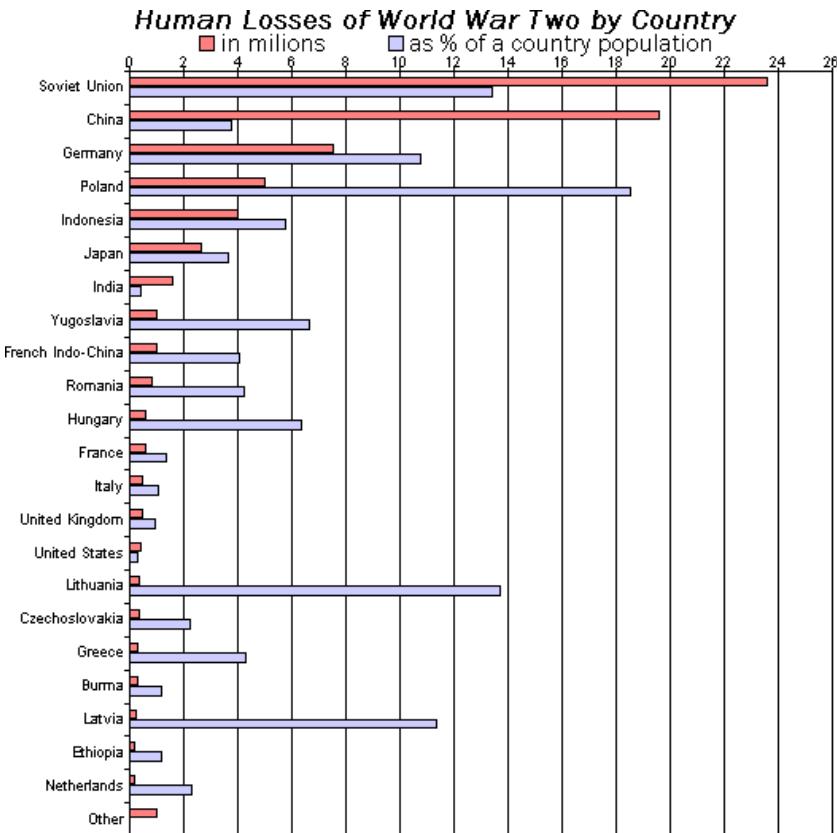
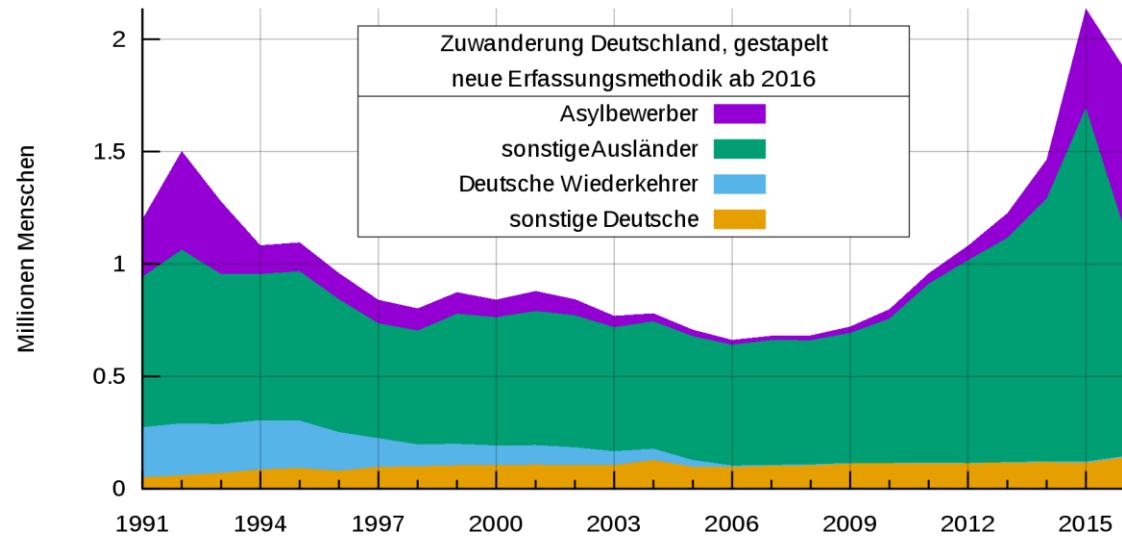
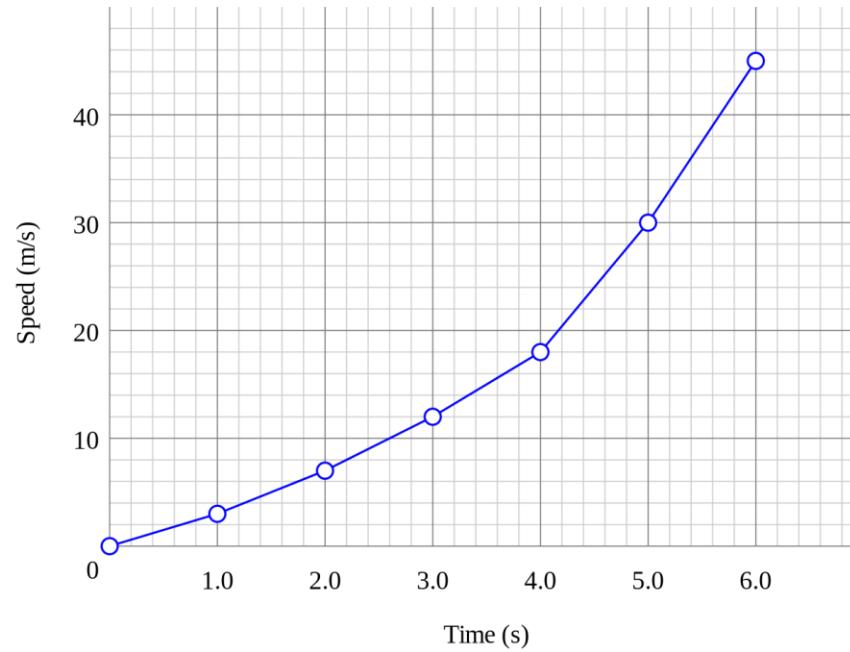
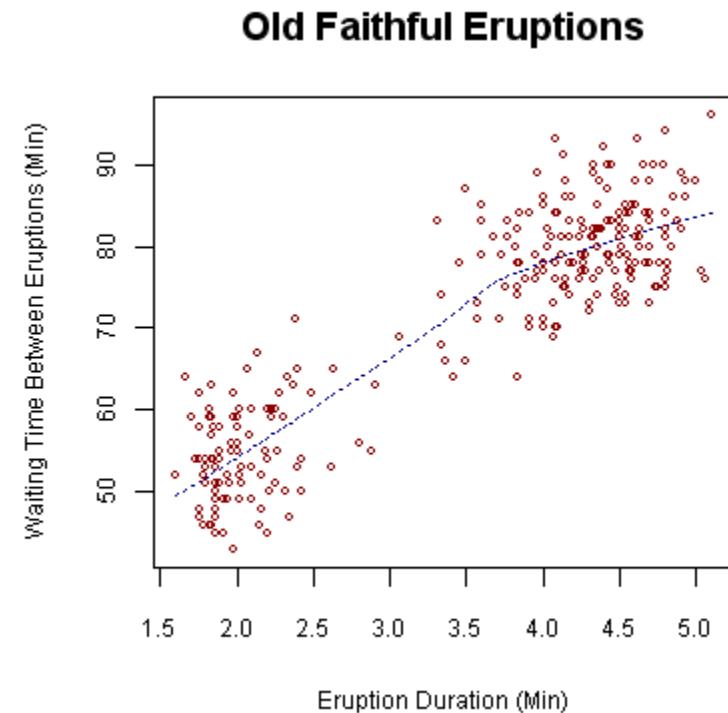


Image: DanielPenfield, Wikipedia CC BY-SA 3.0

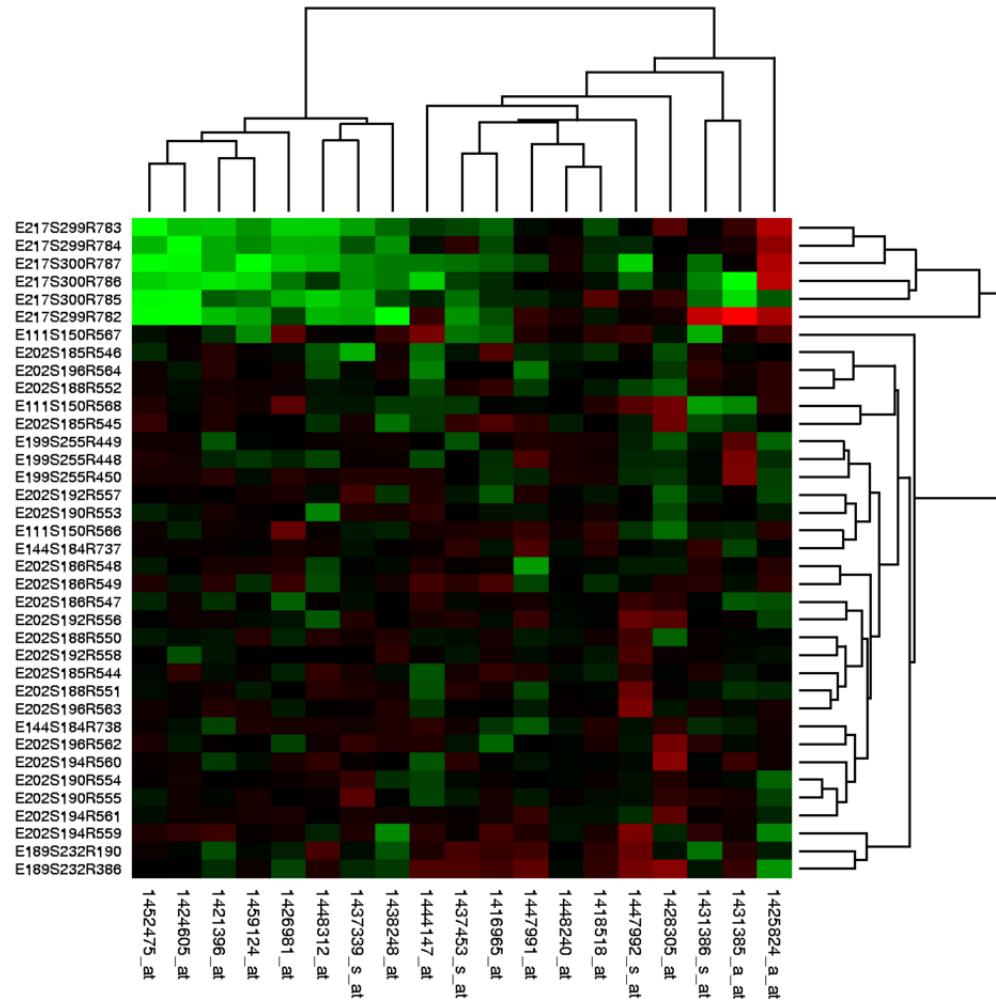
# Line Charts: Changes over Time



# Scatter Plots: Correlation



# Heat Maps: Correlation



# Pie Charts: Aggregation

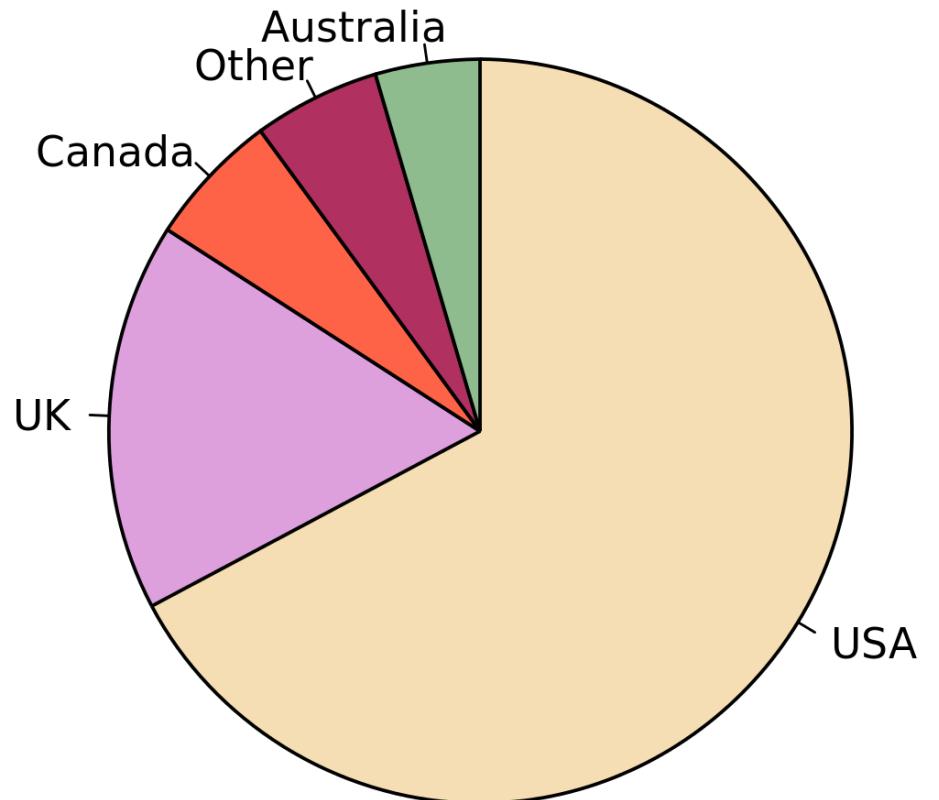
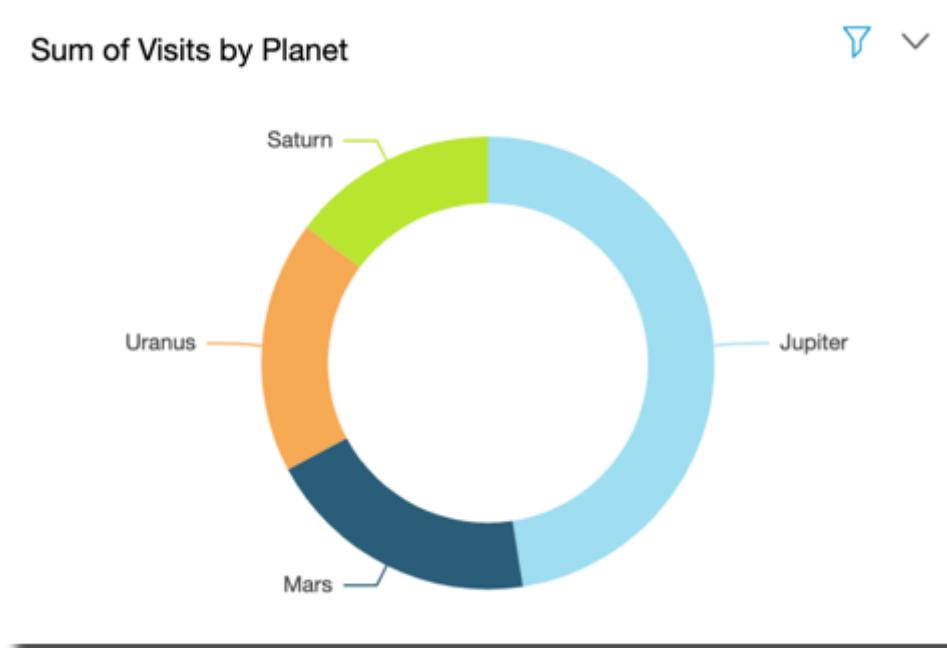
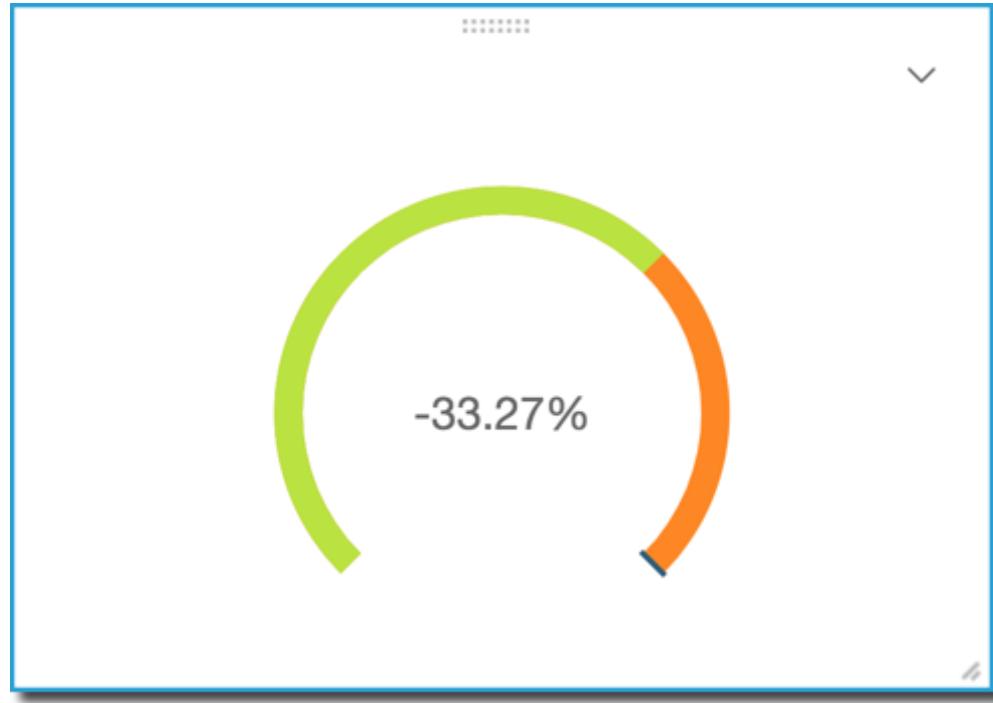


Image: M.W. Toews, Wikipedia, CC BY-SA 4.0

# Donut Charts: Percentage of Total Amount



# Gauge Charts: Compare values in a measure



# Tree Maps: Heirarchical Aggregation

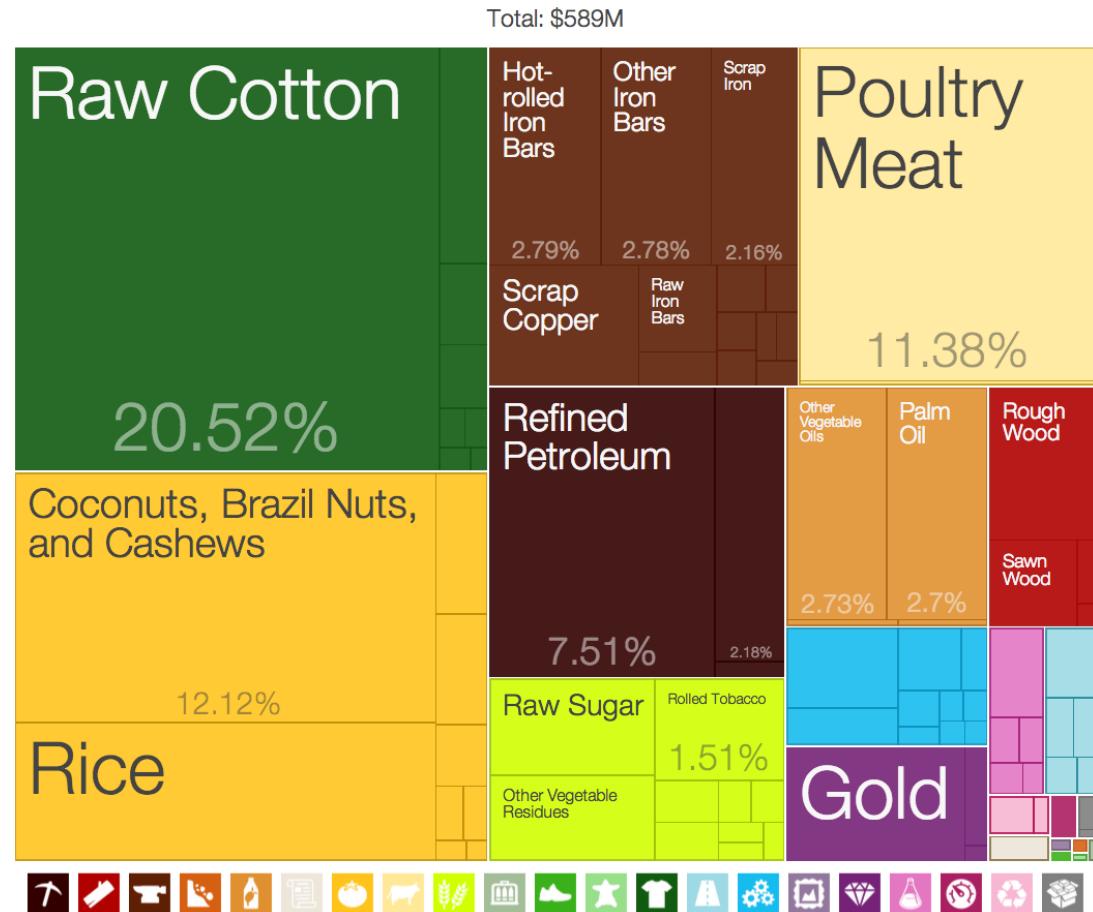


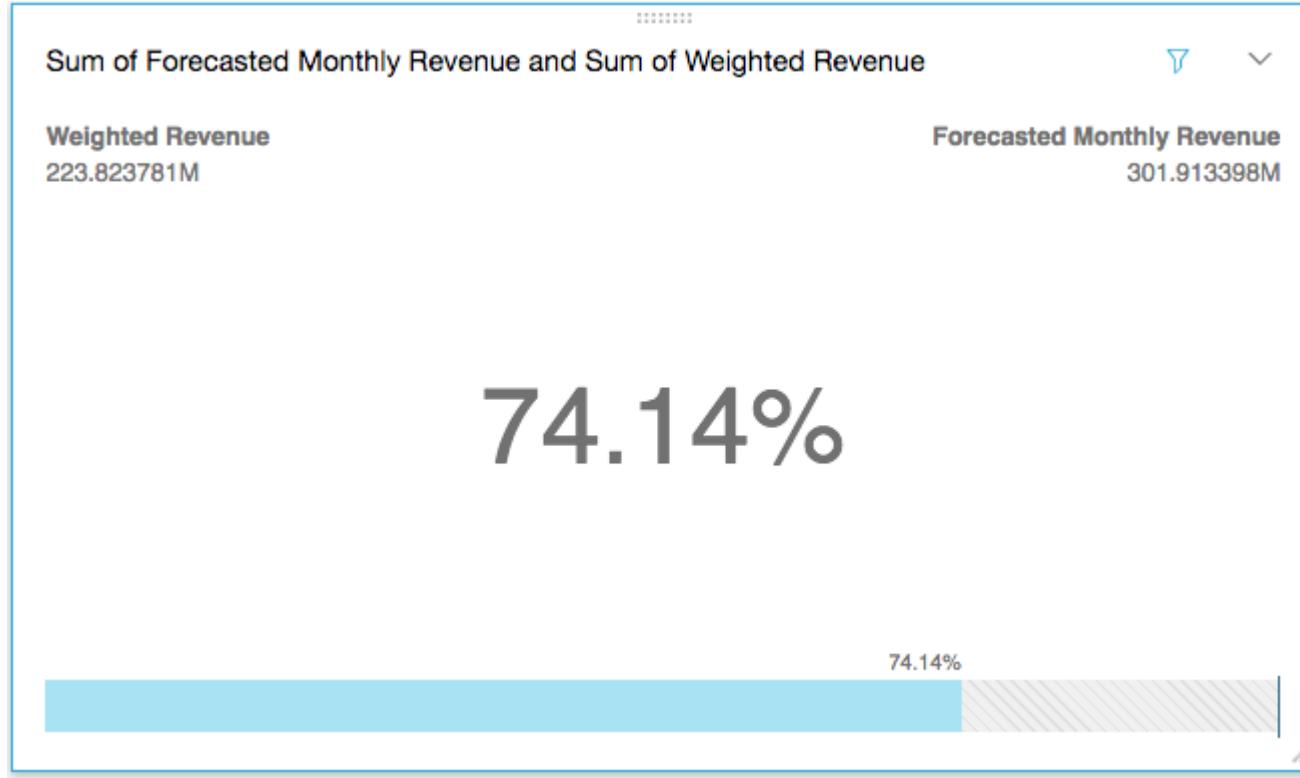
Image: Harvard-MIT Observatory of Economic Complexity, Wikipedia, CC-BY-SA 3.0

# Pivot Tables: Tabular Data

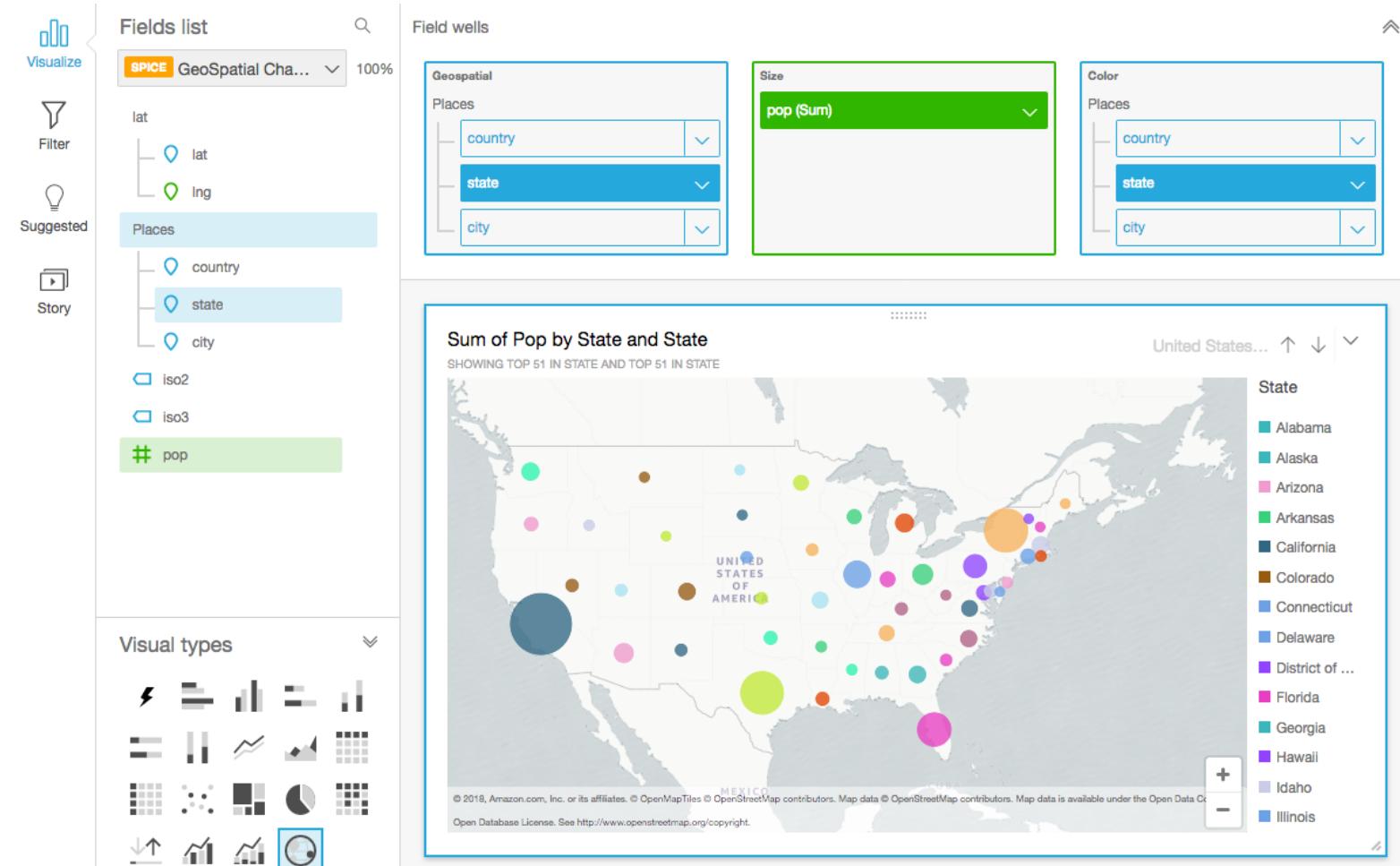
	A	B	C	D	E	F	G
1	Region	Gender	Style	Ship Date	Units	Price	Cost
2	East	Boy	Tee	1/31/2005	12	11.04	10.42
3	East	Boy	Golf	1/31/2005	12	13	12.6
4	East	Boy	Fancy	1/31/2005	12	11.96	11.74
5	East	Girl	Tee	1/31/2005	10	11.27	10.56
6	East	Girl	Golf	1/31/2005	10	12.12	11.95
7	East	Girl	Fancy	1/31/2005	10	13.74	13.33
8	West	Boy	Tee	1/31/2005	11	11.44	10.94
9	West	Boy	Golf	1/31/2005	11	12.63	11.73
10	West	Boy	Fancy	1/31/2005	11	12.06	11.51
11	West	Girl	Tee	1/31/2005	15	13.42	13.29
12	West	Girl	Golf	1/31/2005	15	11.48	10.67

Sum of Units	Ship Date ▼	1/31/2005	2/28/2005	3/31/2005	4/30/2005	5/31/2005	6/30/2005
Region	▼	66	80	102	116	127	125
East		96	117	138	151	154	156
North		123	141	157	178	191	202
South		78	97	117	136	150	157
(blank)							
Grand Total		363	435	514	581	622	640

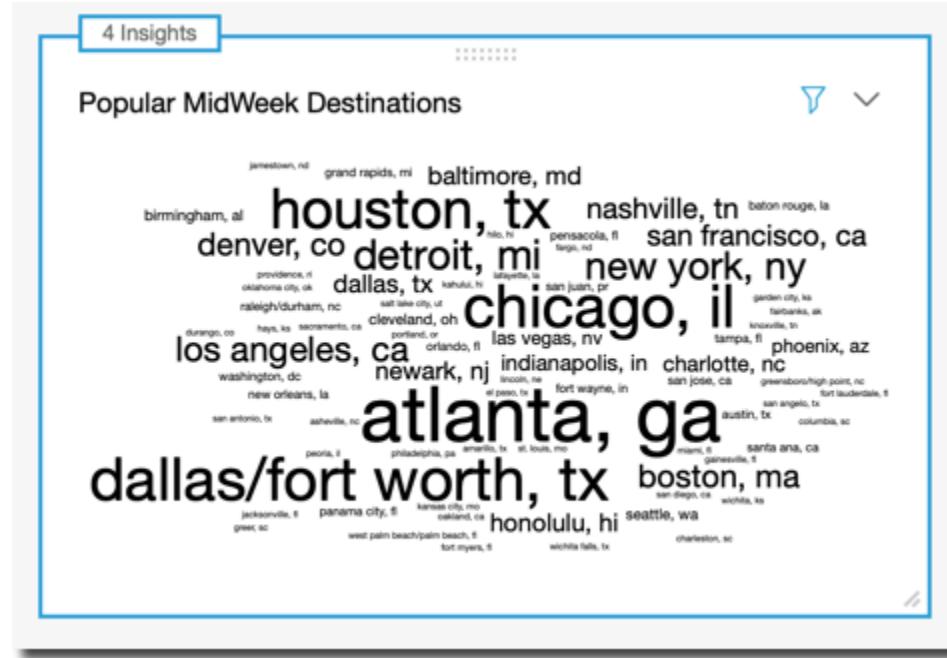
# KPI's: compare key value to its target value



# Geospatial Charts

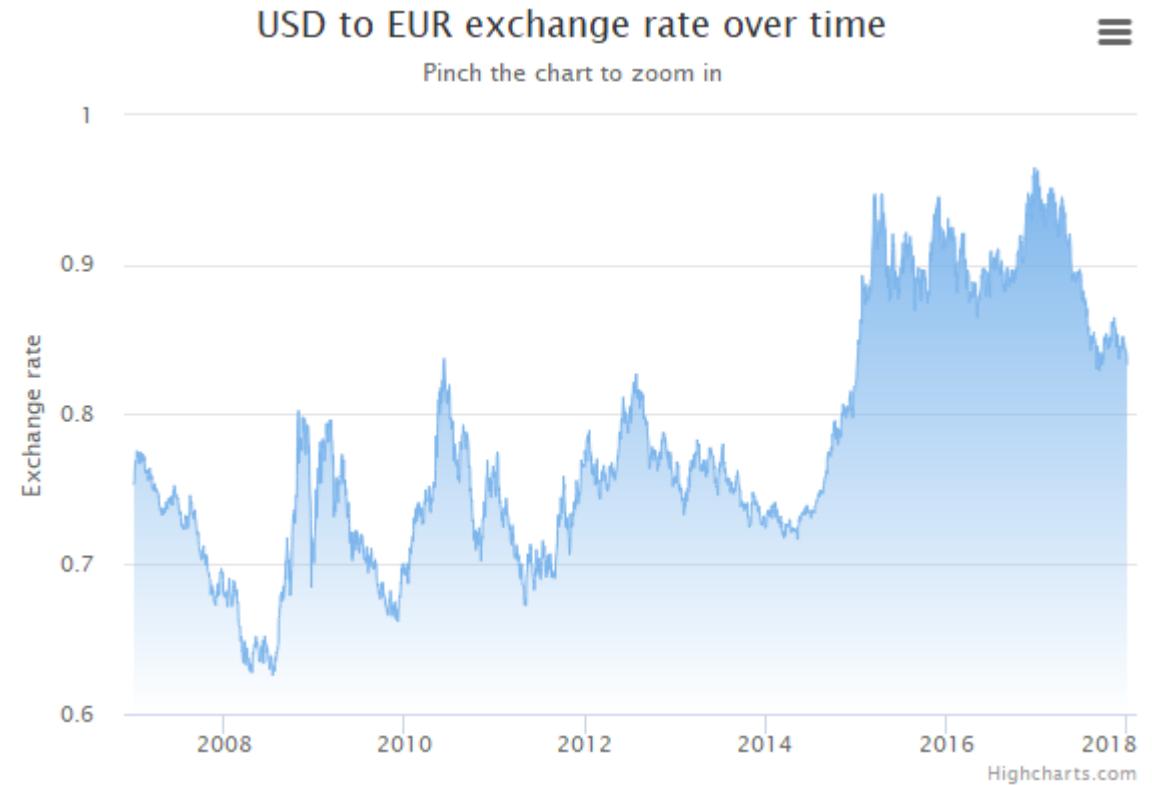


# Word Clouds: word or phrase frequency



# Alternative Visualization Tools

- Web-based visualizations tools (deployed to the public)
  - D3.js
  - Chart.js
  - Highchart.js
- Business Intelligence Tools
  - Tableau
  - MicroStrategy

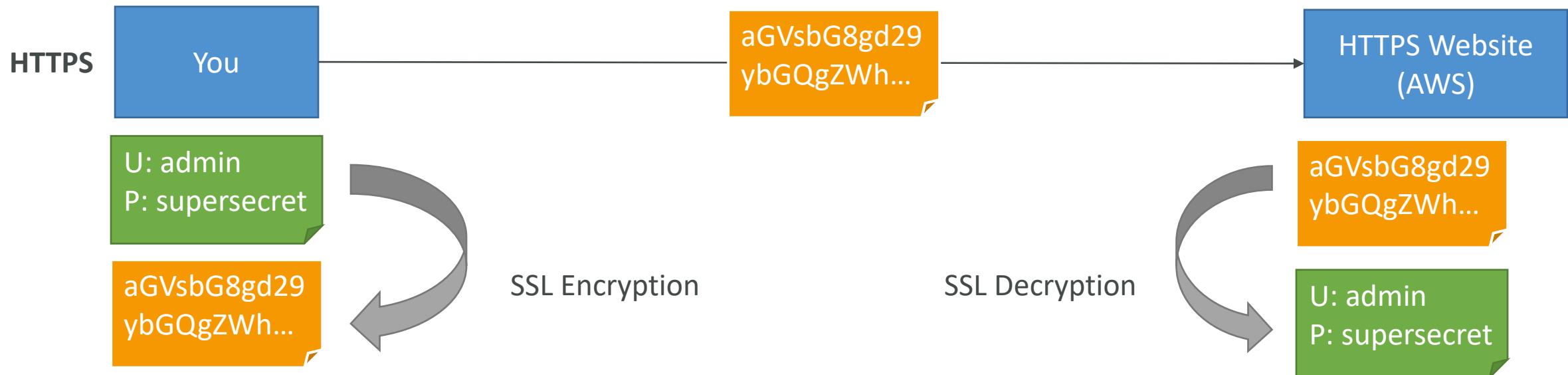


# Security

# Why encryption?

## Encryption in flight (SSL)

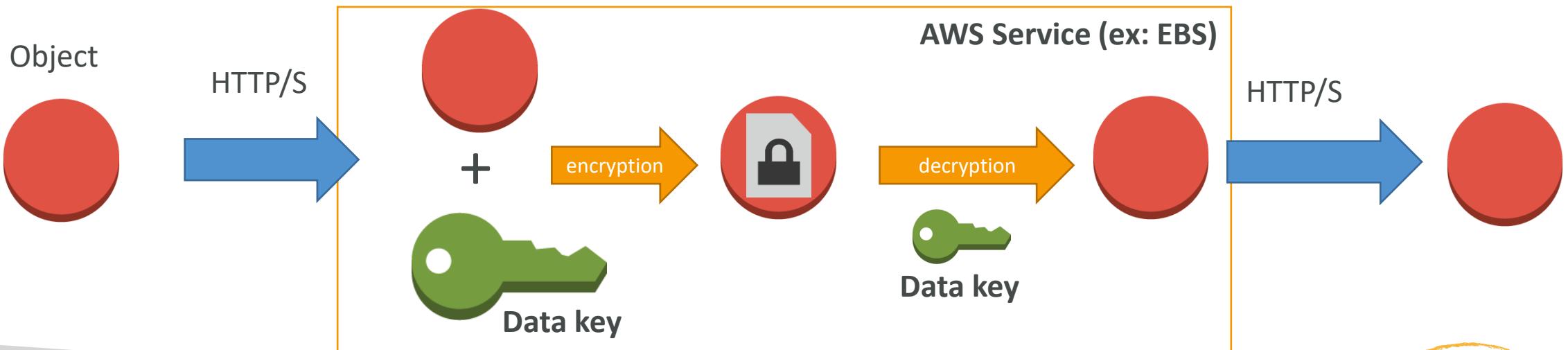
- Data is encrypted before sending and decrypted after receiving
- SSL certificates help with encryption (HTTPS)
- Encryption in flight ensures no MITM (man in the middle attack) can happen



# Why encryption?

## Server side encryption at rest

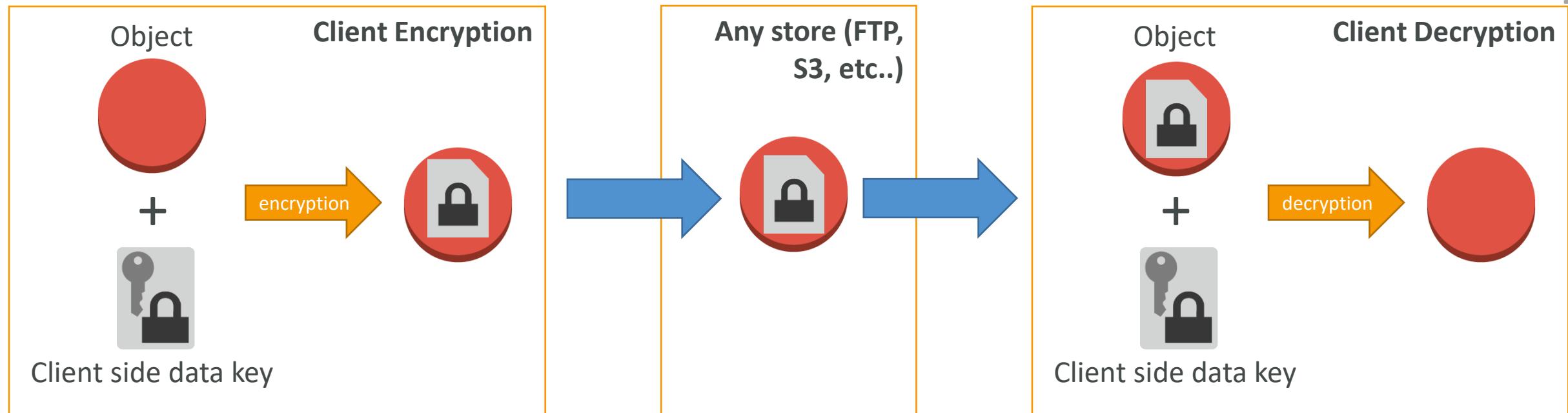
- Data is encrypted after being received by the server
- Data is decrypted before being sent
- It is stored in an encrypted form thanks to a key (usually a data key)
- The encryption / decryption keys must be managed somewhere and the server must have access to it



# Why encryption?

## Client side encryption

- Data is encrypted by the client and never decrypted by the server
- Data will be decrypted by a receiving client
- The server should not be able to decrypt the data
- Could leverage Envelope Encryption

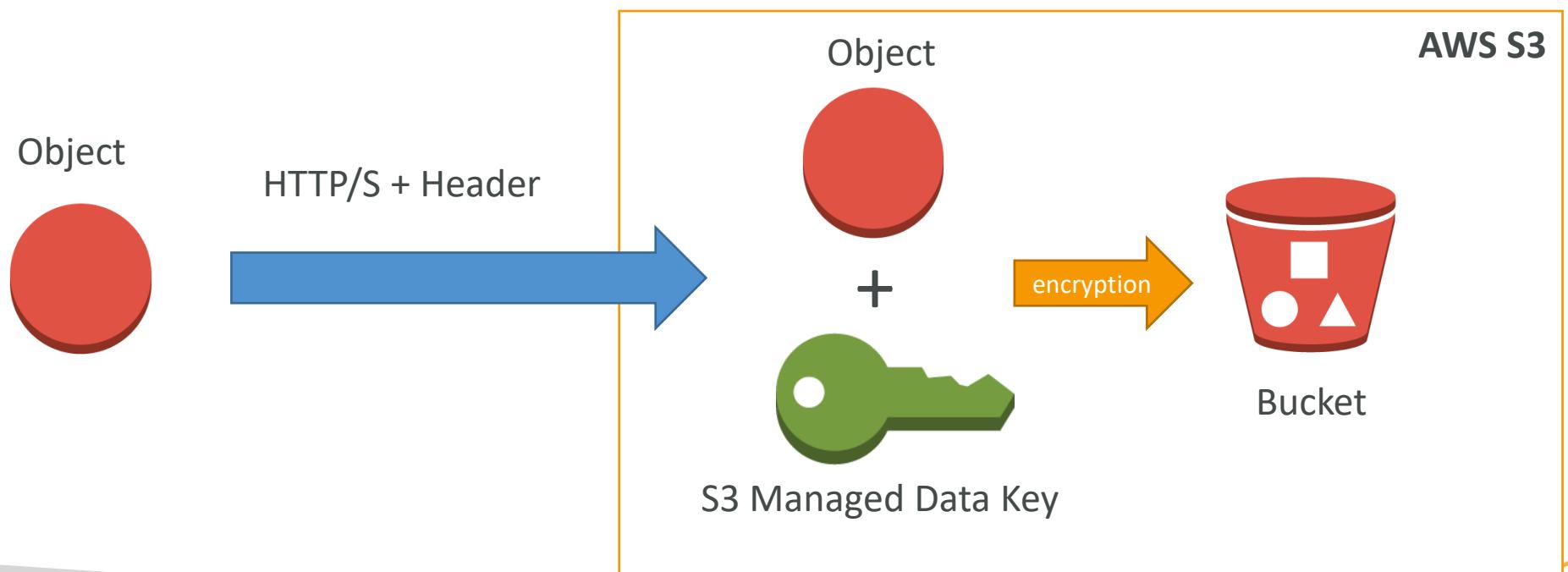


# S3 Encryption for Objects (Reminder)

- There are 4 methods of encrypting objects in S3
  - SSE-S3: encrypts S3 objects using keys handled & managed by AWS
  - SSE-KMS: leverage AWS Key Management Service to manage encryption keys
  - SSE-C: when you want to manage your own encryption keys
  - Client Side Encryption
- It's important to understand which ones are adapted to which situation for the exam

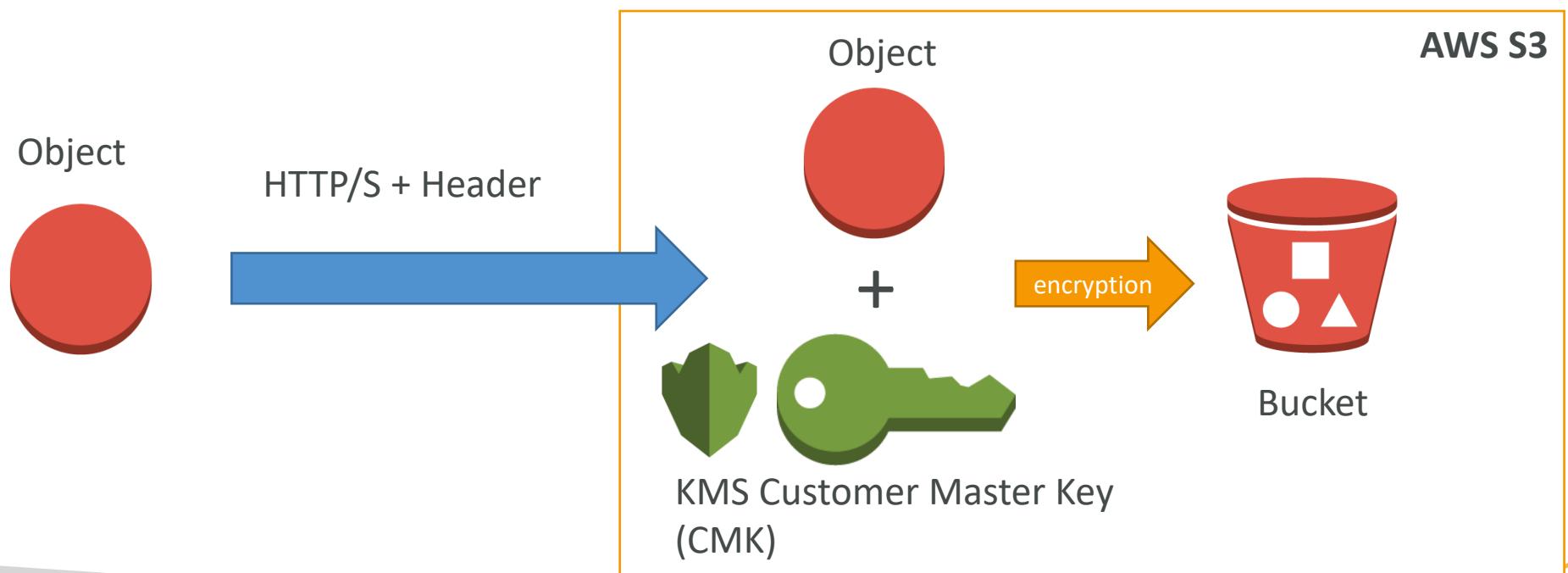
# SSE-S3

- SSE-S3: encryption using keys handled & managed by AWS S3
- Object is encrypted server side
- AES-256 encryption type
- Must set header: “**x-amz-server-side-encryption**”: "AES256"



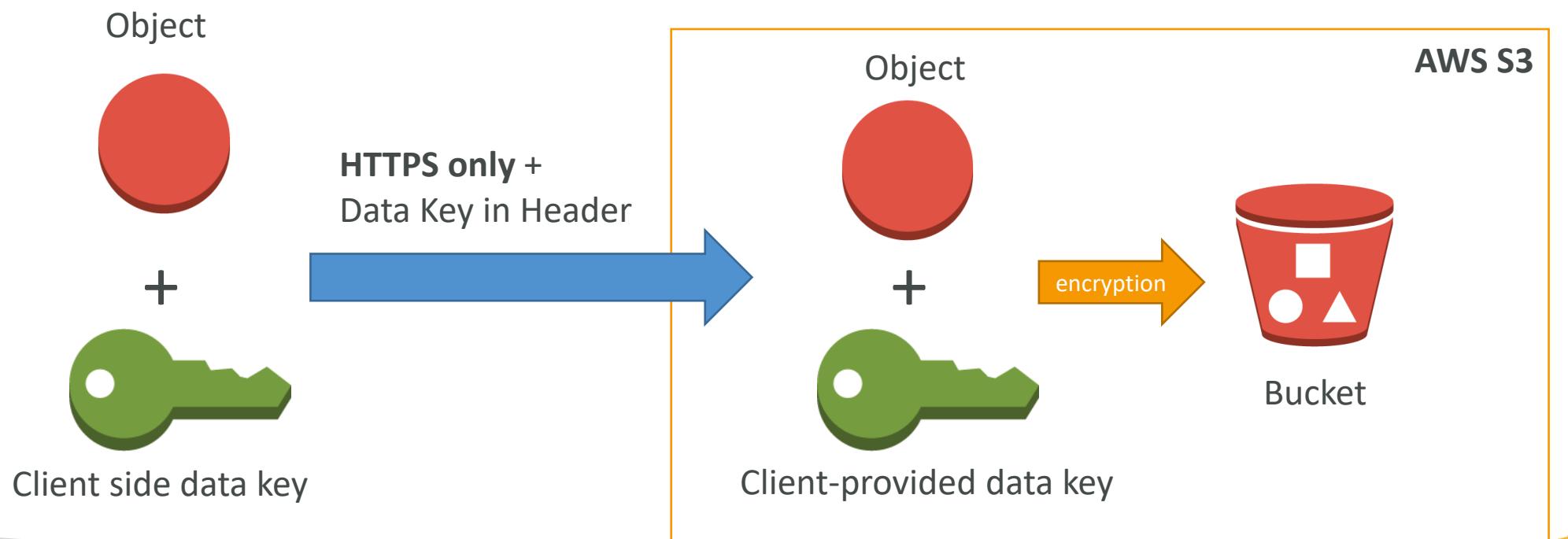
# SSE-KMS

- SSE-KMS: encryption using keys handled & managed by KMS
- KMS Advantages: user control + audit trail
- Object is encrypted server side
- Must set header: “**x-amz-server-side-encryption**”: “aws:kms”



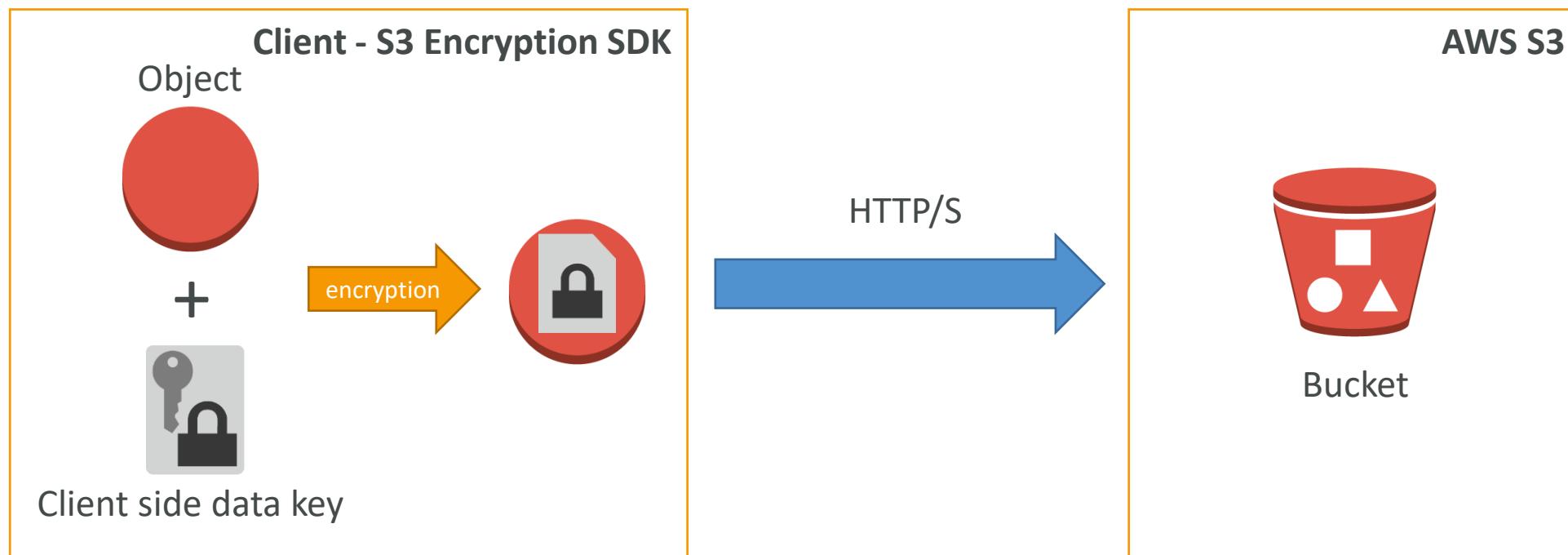
# SSE-C

- SSE-C: server-side encryption using data keys fully managed by the customer outside of AWS
- Amazon S3 does not store the encryption key you provide
- **HTTPS must be used**
- Encryption key must provided in HTTP headers, for every HTTP request made



# Client Side Encryption

- Client library such as the Amazon S3 Encryption Client
- Clients must encrypt data themselves before sending to S3
- Clients must decrypt data themselves when retrieving from S3
- Customer fully manages the keys and encryption cycle



# Encryption in transit (SSL)

- AWS S3 exposes:
  - HTTP endpoint: non encrypted
  - HTTPS endpoint: encryption in flight
- You're free to use the endpoint you want, but HTTPS is recommended
- HTTPS is mandatory for SSE-C
- Encryption in flight is also called SSL / TLS

# AWS KMS (Key Management Service)



- Anytime you hear “encryption” for an AWS service, it’s most likely KMS
- Easy way to control access to your data, AWS manages keys for us
- Fully integrated with IAM for authorization
- Seamlessly integrated into:
  - Amazon EBS: encrypt volumes
  - Amazon S3: Server side encryption of objects
  - Amazon Redshift: encryption of data
  - Amazon RDS: encryption of data
  - Amazon SSM: Parameter store
  - Etc...
- But you can also use the CLI / SDK

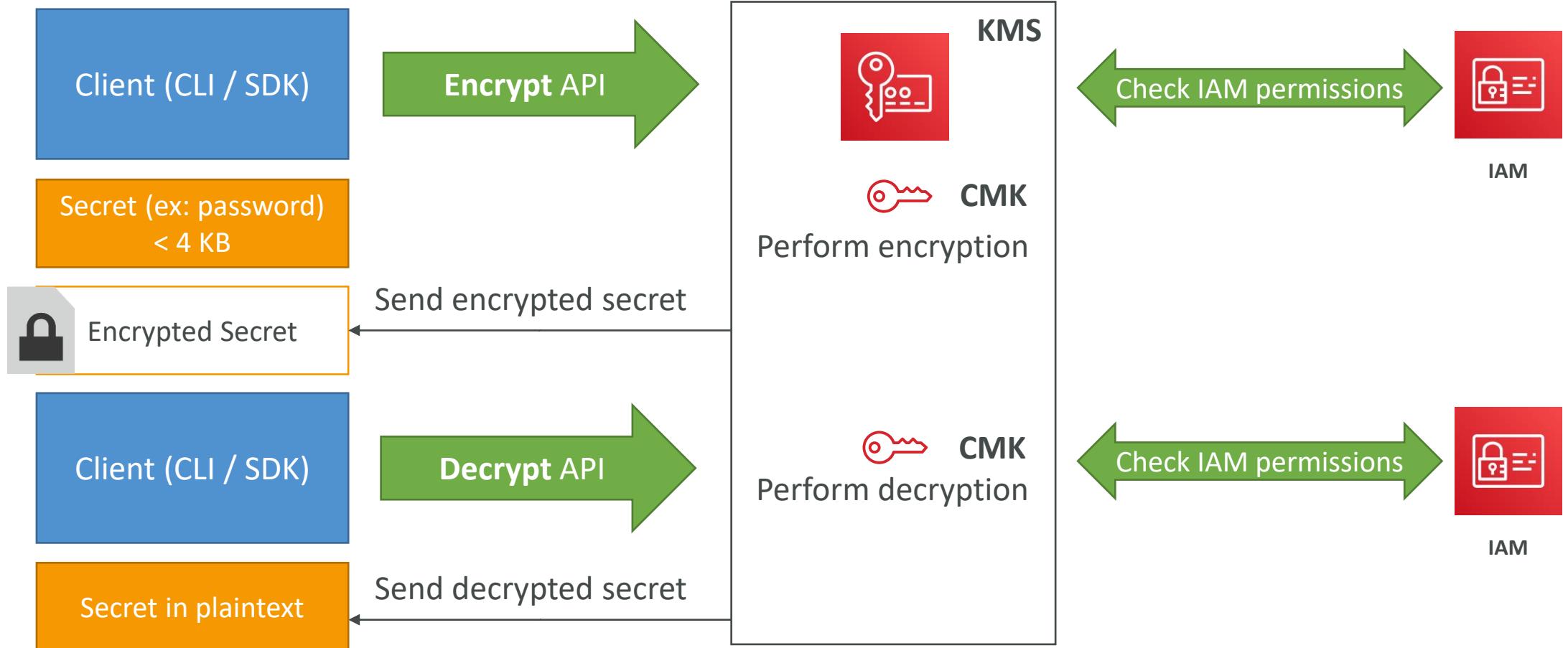
# AWS KMS 101

- Anytime you need to share sensitive information... use KMS
  - Database passwords
  - Credentials to external service
  - Private Key of SSL certificates
- The value in KMS is that the CMK used to encrypt data can never be retrieved by the user, and the CMK can be rotated for extra security
- **Never ever store your secrets in plaintext, especially in your code!**
- Encrypted secrets can be stored in the code / environment variables
- **KMS can only help in encrypting up to 4KB of data per call**
- If data > 4 KB, use envelope encryption
- To give access to KMS to someone:
  - Make sure the Key Policy allows the user
  - Make sure the IAM Policy allows the API calls

# AWS KMS (Key Management Service)

- Able to fully manage the keys & policies:
  - Create
  - Rotation policies
  - Disable
  - Enable
- Able to audit key usage (using CloudTrail)
- Three types of Customer Master Keys (CMK):
  - AWS Managed Service Default CMK: **free**
  - User Keys created in KMS: **\$1 / month**
  - User Keys imported (must be 256-bit symmetric key): **\$1 / month**
- + pay for API call to KMS (**\$0.03 / 10000 calls**)

# How does KMS work? API – Encrypt and Decrypt



# Encryption in AWS Services

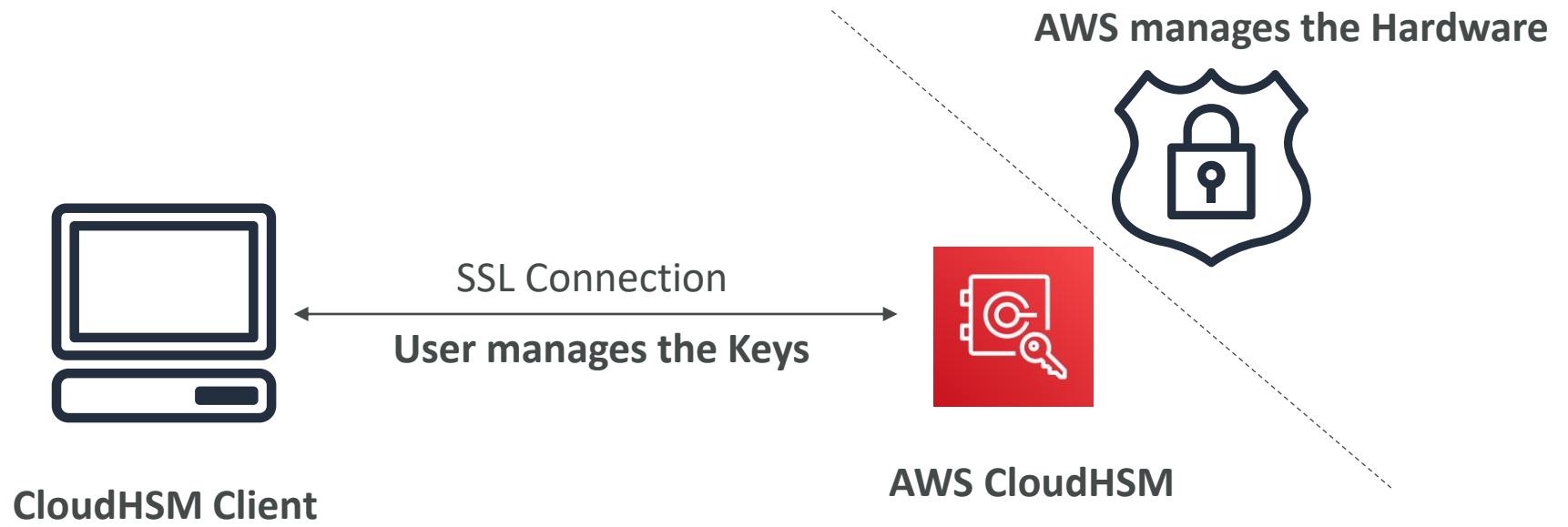
- Requires migration (through Snapshot / Backup):
  - EBS Volumes
  - RDS databases
  - ElastiCache
  - EFS network file system
- In-place encryption:
  - S3

# CloudHSM



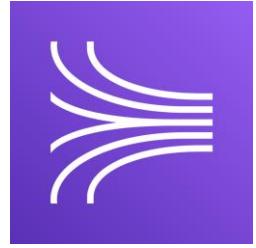
- KMS => AWS manages the software for encryption
- CloudHSM => AWS provisions encryption **hardware**
- Dedicated Hardware (HSM = Hardware Security Module)
- You manage your own encryption keys entirely (not AWS)
- HSM device is tamper resistant, FIPS 140-2 Level 3 compliance
- **CloudHSM clusters are spread across Multi AZ (HA) – must setup**
- Supports both symmetric and **asymmetric** encryption (SSL/TLS keys)
- No free tier available
- Must use the CloudHSM Client Software
- Redshift supports CloudHSM for database encryption and key management
- Good option to use with SSE-C encryption

# CloudHSM Diagram



# CloudHSM vs KMS

Feature	AWS KMS	AWS CloudHSM
<b>Tenancy</b>	Uses multi-tenant key storage	Single tenant key storage, dedicated to one customer
<b>Keys</b>	Keys owned and managed by AWS	Customer managed HSM
<b>Encryption</b>	Supports only symmetric key encryption	Supports both symmetric and asymmetric encryption
<b>Cryptographic Acceleration</b>	None	SSL/TLS Acceleration Oracle TDE Acceleration
<b>Key Storage and Management</b>	Accessible from multiple regions Centralized management from IAM	Deployed and managed from a customer VPC. Accessible and can be shared across VPCs using VPC peering
<b>Free Tier Availability</b>	Yes	No



# Security - Kinesis

- Kinesis Data Streams
  - SSL endpoints using the HTTPS protocol to do encryption in flight
  - AWS KMS provides server-side encryption [Encryption at rest]
  - For client side-encryption, you **must** use your own encryption libraries
  - Supported Interface VPC Endpoints / Private Link – access privately
  - KCL – must get read / write access to DynamoDB table
- Kinesis Data Firehose:
  - Attach IAM roles so it can deliver to S3 / ES / Redshift / Splunk
  - Can encrypt the delivery stream with KMS [Server side encryption]
  - Supported Interface VPC Endpoints / Private Link – access privately
- Kinesis Data Analytics
  - Attach IAM role so it can read from Kinesis Data Streams and reference sources and write to an output destination (example Kinesis Data Firehose)



# Security - SQS

- Encryption in flight using the HTTPS endpoint
- Server Side Encryption using KMS
- IAM policy must allow usage of SQS
- SQS queue access policy
  
- Client-side encryption must be implemented manually
- VPC Endpoint is provided through an Interface

# Security – AWS IoT



- AWS IoT policies:
  - Attached to X.509 certificates or Cognito Identities
  - Able to revoke any device at any time
  - IoT Policies are JSON documents
  - Can be attached to groups instead of individual Things.
- IAM Policies:
  - Attached to users, group or roles
  - Used for controlling IoT AWS APIs
- Attach roles to Rules Engine so they can perform their actions

# Security – Amazon S3



- IAM policies
- S3 bucket policies
- Access Control Lists (ACLs)
- Encryption in flight using HTTPS
- Encryption at rest
  - Server-side encryption: SSE-S3, SSE-KMS, SSE-C
  - Client-side encryption – such as Amazon S3 Encryption Client
- Versioning + MFA Delete
- CORS for protecting websites
- VPC Endpoint is provided through a Gateway
- Glacier – vault lock policies to prevent deletes (WORM)



# Security – DynamoDB

- Data is encrypted in transit using TLS (HTTPS)
- DynamoDB can be encrypted at rest
  - KMS encryption for base tables and secondary indexes
  - Only for new tables
  - To migrate un-encrypted table, create new table and copy the data
  - Encryption cannot be disabled once enabled
- Access to tables / API / DAX using IAM
- DynamoDB Streams do not support encryption
- VPC Endpoint is provided through a Gateway



# Security - RDS

- VPC provides network isolation
- Security Groups control network access to DB Instances
- KMS provides encryption at rest
- SSL provides encryption in-flight
- IAM policies provide protection for the RDS API
- IAM authentication is supported by PostgreSQL and MySQL
- Must manage user permissions within the database itself
- MSSQL Server and Oracle support TDE (Transparent Data Encryption)



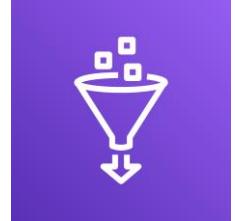
# Security - Aurora

- (very similar to RDS)
- VPC provides network isolation
- Security Groups control network access to DB Instances
- KMS provides encryption at rest
- SSL provides encryption in-flight
- IAM authentication is supported by PostgreSQL and MySQL
- Must manage user permissions within the database itself



# Security - Lambda

- IAM roles attached to each Lambda function
- Sources
- Targets
- KMS encryption for secrets
- SSM parameter store for configurations
- CloudWatch Logs
- Deploy in VPC to access private resources



# Security - Glue

- IAM policies for the Glue service
- Configure Glue to only access JDBC through SSL
- Data Catalog:
  - Encrypted by KMS
  - **Resource Policies to protect Data Catalog resources** (similar to S3 bucket policy)
- Connection passwords: Encrypted by KMS
- Data written by AWS Glue – Security Configurations:
  - S3 encryption mode: SSE-S3 or SSE-KMS
  - CloudWatch encryption mode
  - Job bookmark encryption mode

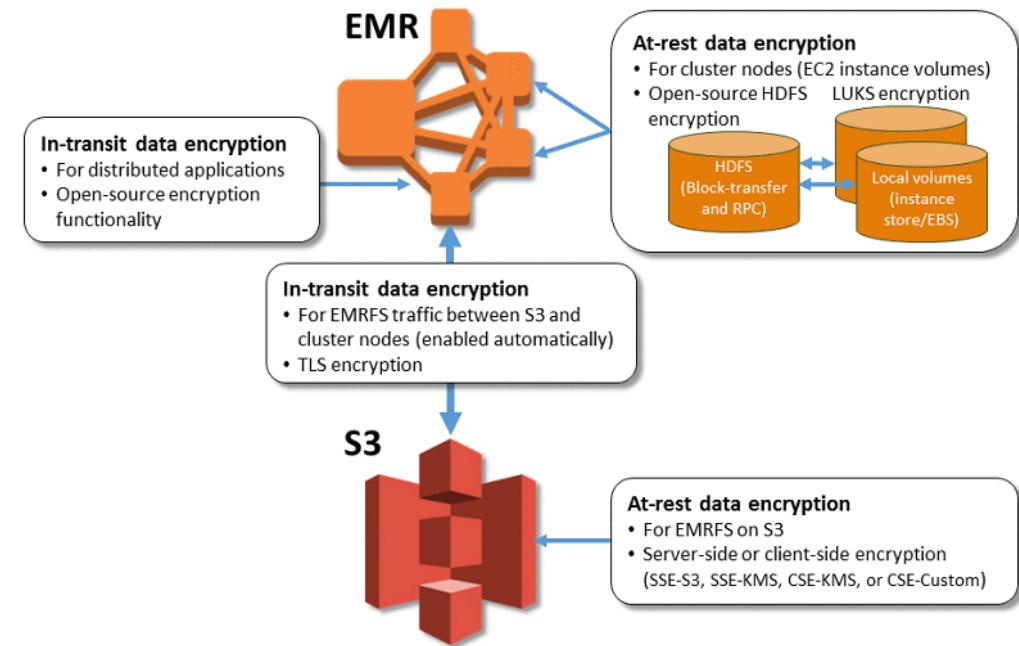


# Security - EMR

- Using Amazon EC2 key pair for SSH credentials
- Attach IAM roles to EC2 instances for:
  - proper S3 access
  - for EMRFS requests to S3
  - DynamoDB scans through Hive
- EC2 Security Groups
  - One for master node
  - Another one for cluster node (core node or task node)
- Encrypts data at-rest: EBS encryption, Open Source HDFS Encryption, LUKS + EMRFS for S3
- In-transit encryption: node to node communication, EMRFS, TLS
- Data is encrypted before uploading to S3
- Kerberos authentication (provide authentication from Active Directory)
- Apache Ranger: Centralized Authorization (RBAC – Role Based Access) – setup on external EC2
- <https://aws.amazon.com/blogs/big-data/best-practices-for-securin>

# Security – EMR Encryption (security config)

- **At-rest data encryption for EMRFS:**
  - Encryption in Amazon S3 (SSE-S3, SSE-KMS, Client-Side encryption)
  - Encryption in Local Disks
- **At-rest data encryption for local disks:**
  - Open-source HDFS encryption
  - EC2 Instance Store encryption: NVMe encryption, or LUKS encryption
  - EBS volumes: EBS encryption (KMS) – **works with root volume**  
LUKS encryption – does not work with root
- **In-transit encryption:**
  - Node to node communication
  - For EMRFS traffic between S3 and cluster nodes
  - TLS encryption



<https://docs.aws.amazon.com/emr/latest/ManagementGuide/emr-data-encryption-options.html>



# Security – ElasticSearch Service

- Amazon VPC provides network isolation
- ElasticSearch policy to manage security further
- Data security by encrypting data at-rest using KMS
- Encryption in-transit using SSL
  
- IAM or Cognito based authentication
- Amazon Cognito allow end-users to log-in to Kibana through enterprise identity providers such as Microsoft Active Directory using SAML

# Security - Redshift



- VPC provides network isolation
- Cluster security groups
- Encryption in flight using the JDBC driver enabled with SSL
- Encryption at rest using KMS or an HSM device (establish a connection)
- Supports S3 SSE using default managed key
- Use IAM Roles for Redshift
- To access other AWS Resources (example S3 or KMS)
- Must be referenced in the COPY or UNLOAD command (alternatively paste access key and secret key creds)



# Security - Athena

- IAM policies to control access to the service
- Data is in S3: IAM policies, bucket policies & ACLs
- Encryption of data according to S3 standards: SSE-S3, SSE-KMS, CSE-KMS
- Encryption in transit using TLS between Athena and S3 and JDBC
- Fine grained access using the AWS Glue Catalog



# Security - Quicksight

- Standard edition:
  - IAM users
  - Email based accounts
- Enterprise edition:
  - Active Directory
  - Federated Login
  - Supports MFA (Multi Factor Authentication)
  - Encryption at rest and in SPICE
- Row Level Security to control which users can see which rows

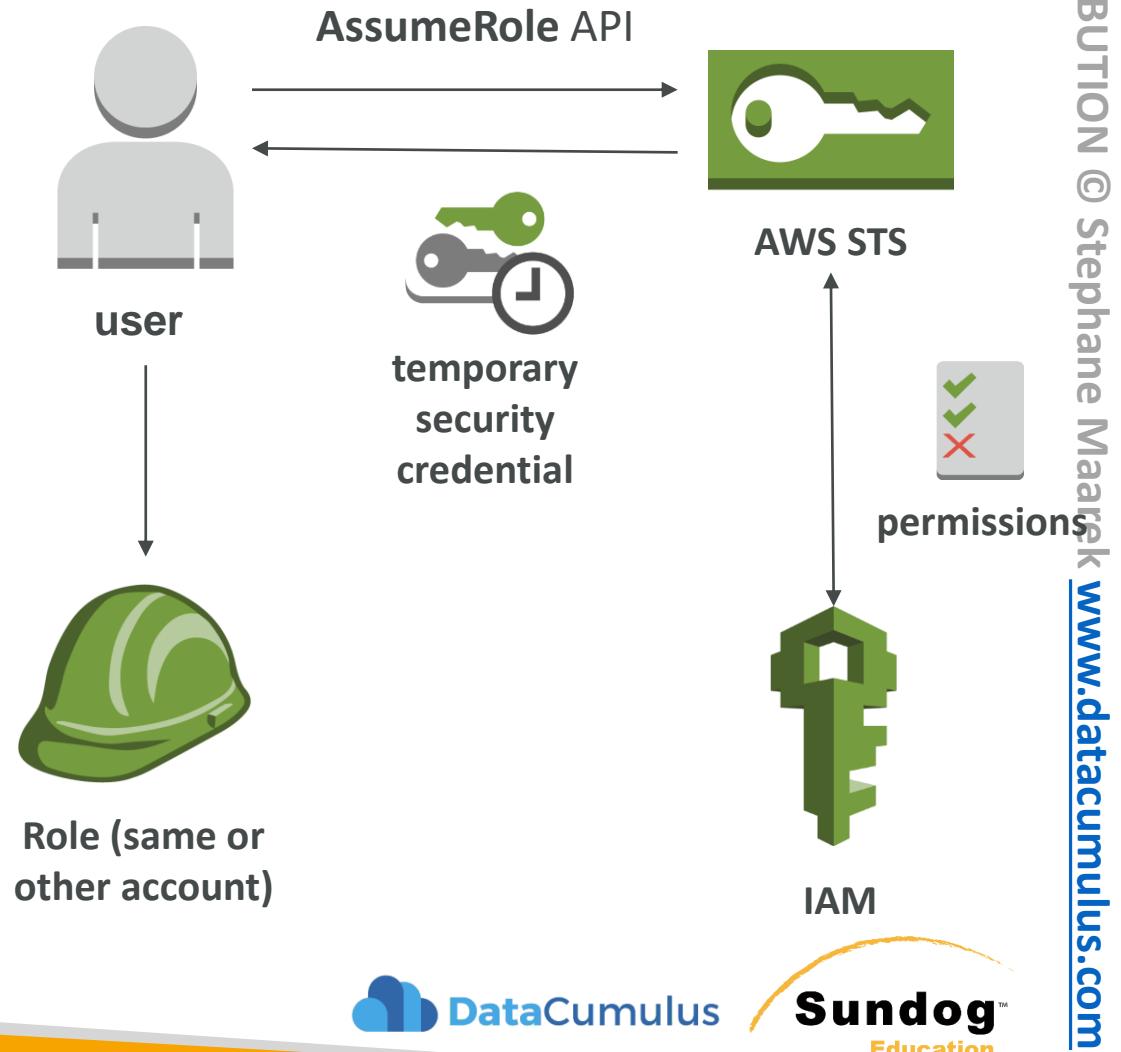
# AWS STS – Security Token Service



- Allows to grant limited and temporary access to AWS resources.
- Token is valid for up to one hour (must be refreshed)
- **Cross Account Access**
  - Allows users from one AWS account access resources in another
- **Federation (Active Directory)**
  - Provides a non-AWS user with temporary AWS access by linking users Active Directory credentials
  - Uses SAML (Security Assertion markup language)
  - Allows Single Sign On (SSO) which enables users to log in to AWS console without assigning IAM credentials
- **Federation with third party providers / Cognito**
  - Used mainly in web and mobile applications
  - Makes use of Facebook/Google/Amazon etc to federate them

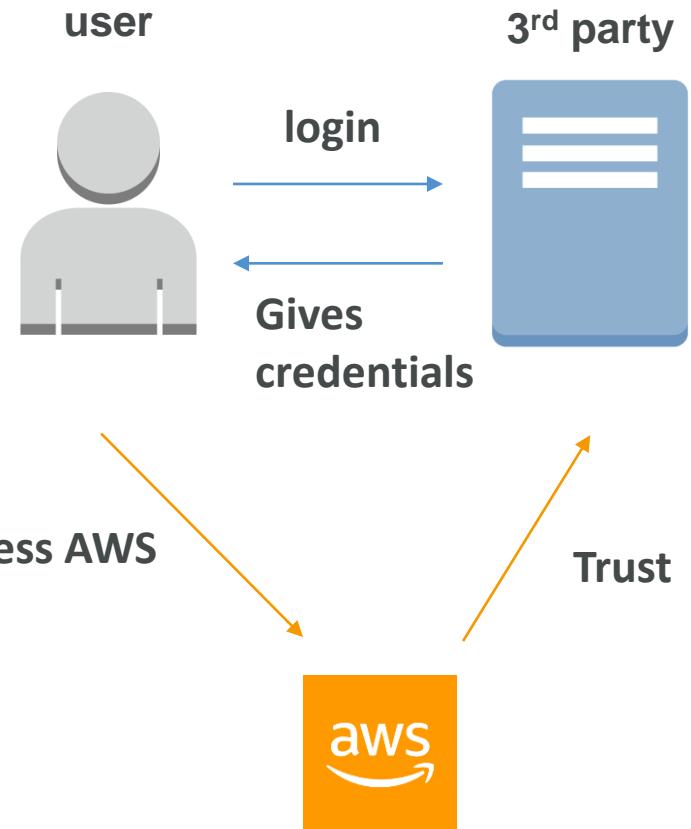
# Cross Account Access

- Define an IAM Role for another account to access
- Define which accounts can access this IAM Role
- Use AWS STS (Security Token Service) to retrieve credentials and impersonate the IAM Role you have access to (**AssumeRole API**)
- Temporary credentials can be valid between 15 minutes to 1 hour



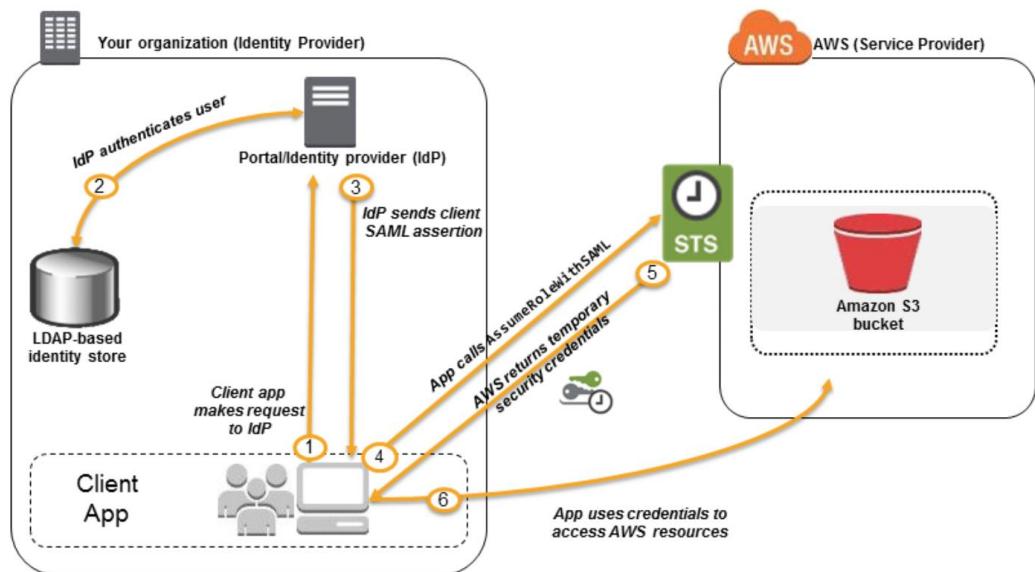
# What's Identity Federation?

- Federation lets users outside of AWS to assume temporary role for accessing AWS resources.
- These users assume identity provided access role.
- **Federation assumes a form of 3rd party authentication**
  - LDAP
  - Microsoft Active Directory (~= SAML)
  - Single Sign On
  - Open ID
  - Cognito
- **Using federation, you don't need to create IAM users (user management is outside of AWS)**

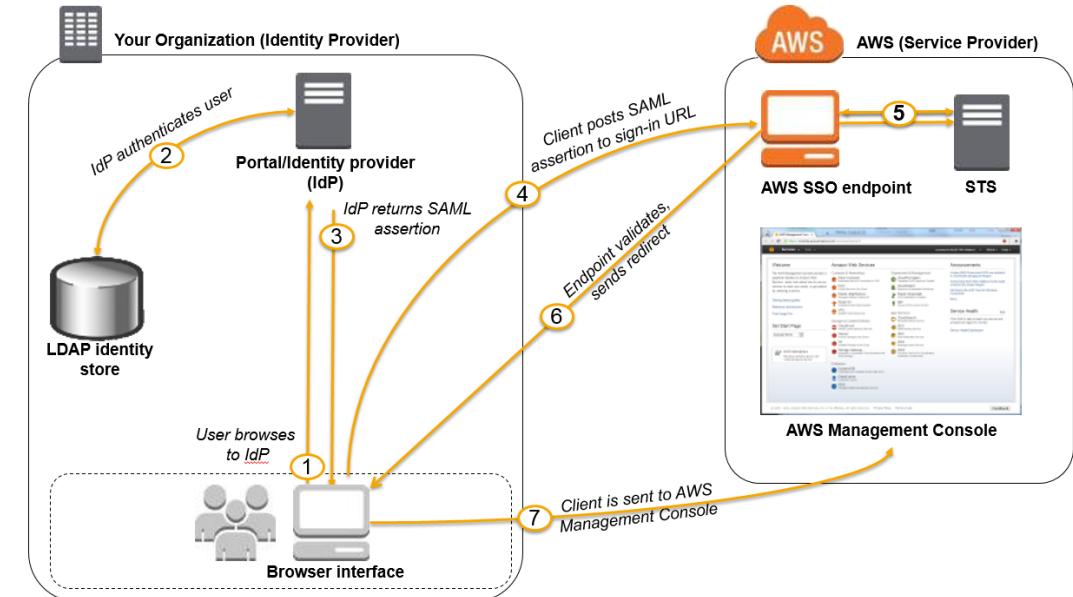


# SAML Federation For Enterprises

- To integrate Active Directory / ADFS with AWS (or any SAML 2.0)
- Provides access to AWS Console or CLI (through temporary creds)
- No need to create an IAM user for each of your employees



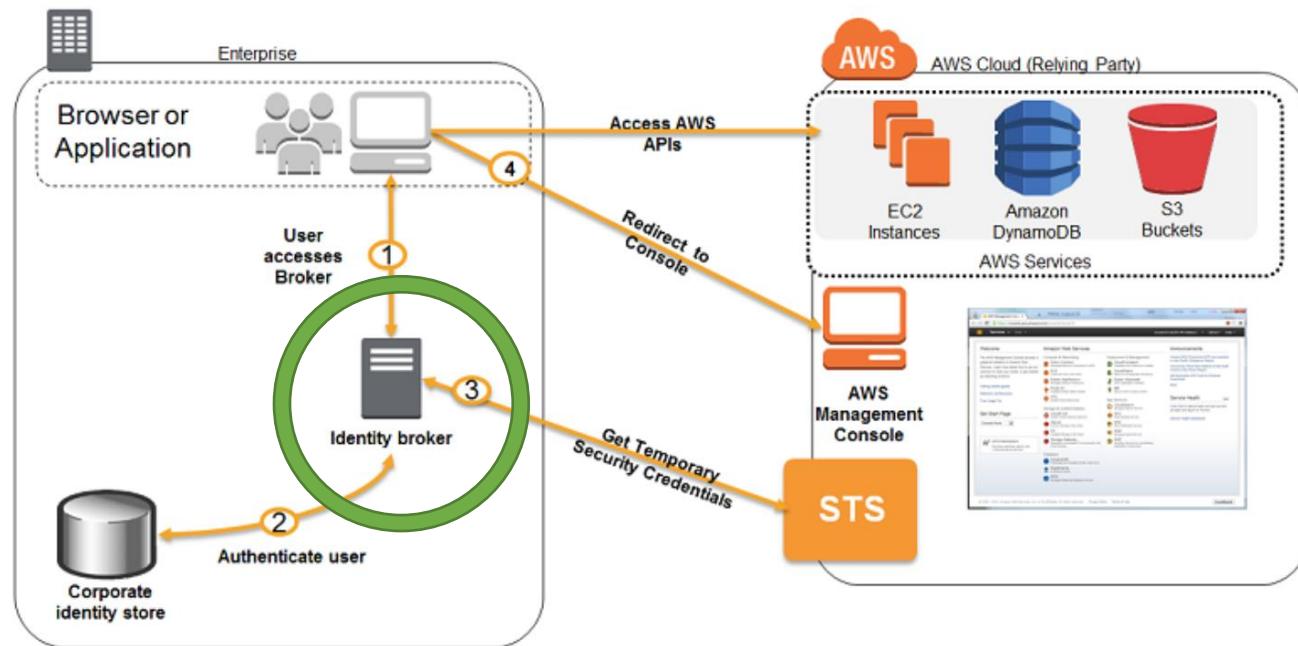
[https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_roles\\_providers\\_saml.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_saml.html)



[https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_roles\\_providers\\_enable-console-saml.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_providers_enable-console-saml.html)

# Custom Identity Broker Application For Enterprises

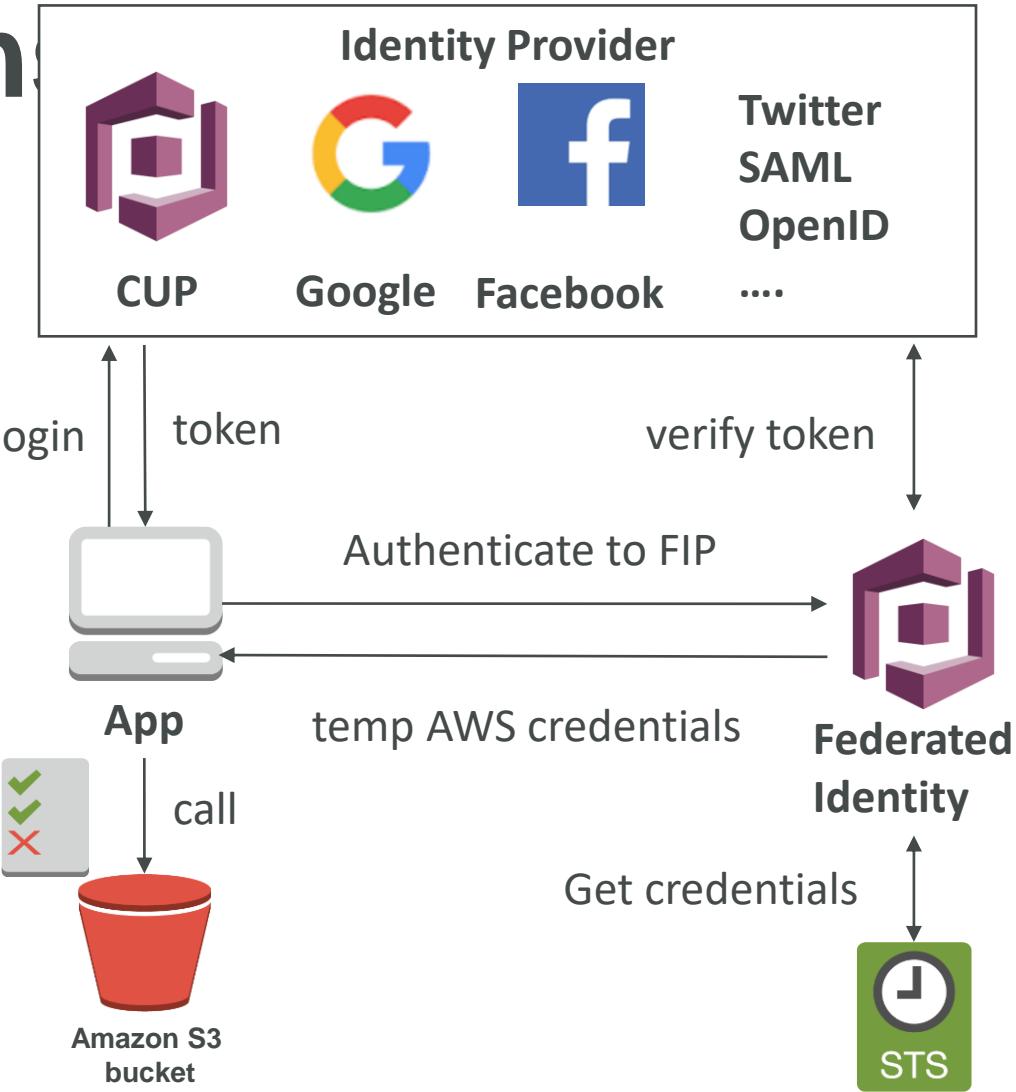
- Use only if identity provider is not compatible with SAML 2.0
- **The identity broker must determine the appropriate IAM policy**



[https://docs.aws.amazon.com/IAM/latest/UserGuide/id\\_roles\\_common-scenarios\\_federated-users.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_common-scenarios_federated-users.html)

# AWS Cognito - Federated Identity Pools For Public Application

- **Goal:**
  - Provide direct access to AWS Resources from the Client Side
- **How:**
  - Log in to federated identity provider – or remain anonymous
  - Get temporary AWS credentials back from the Federated Identity Pool
  - These credentials come with a pre-defined IAM policy stating their permissions
- **Example:**
  - provide (temporary) access to write to S3 bucket using Facebook Login
- **Note:**
  - Web Identity Federation is an alternative to using Cognito but AWS recommends against it



# Policies – leveraging AWS variables

- [https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_variables.html](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_variables.html)
  - \${aws:username} : to restrict users to tables / buckets
  - \${aws:principaltype} : account, user, federated, or assumed role
  - \${aws:PrincipalTag/department} : to restrict using Tags
- [https://docs.aws.amazon.com/IAM/latest/UserGuide/reference\\_policies\\_iam-condition-keys.html#condition-keys-wif](https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_iam-condition-keys.html#condition-keys-wif)
  - \${aws:FederatedProvider} : which IdP was used for the user (Cognito, Amazon..)
  - \${www.amazon.com:user\_id} , \${cognito-identity.amazonaws.com:sub} ...
  - \${saml:sub}, \${sts:ExternalId}

# Policies - Advanced

- For S3 - let's analyze the policies at:  
<https://docs.aws.amazon.com/AmazonS3/latest/dev/example-bucket-policies.html>
- For DynamoDB – let's analyze the policies at:  
<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/specifying-conditions.html>
- Note for RDS – IAM policies don't help with **in-database** security, as it's a proprietary technology and we are responsible for users & authorization

# AWS CloudTrail



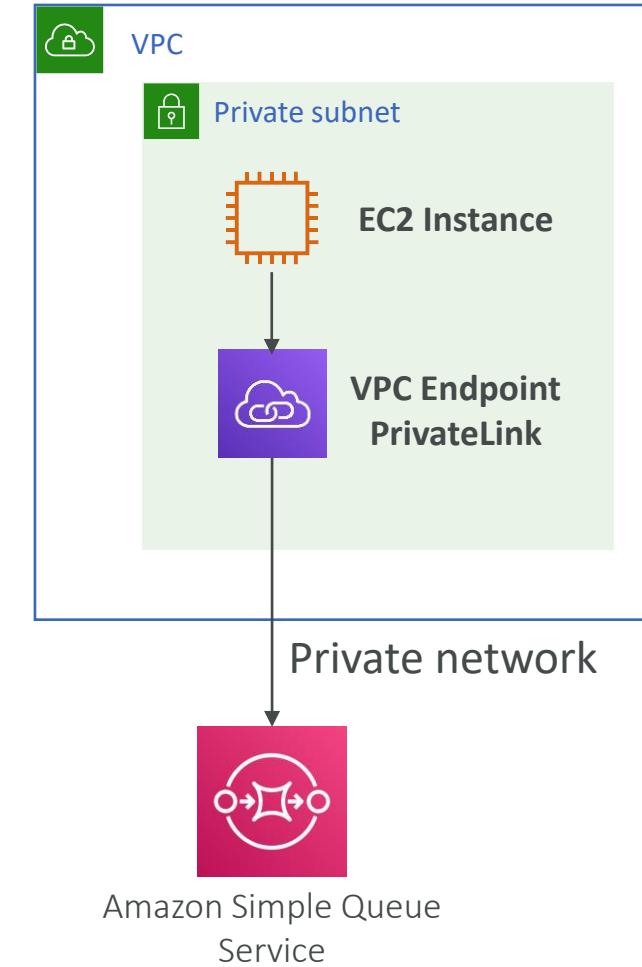
- Provides governance, compliance and audit for your AWS Account
- CloudTrail is enabled by default!
- Get an history of events / API calls made within your AWS Account by:
  - Console
  - SDK
  - CLI
  - AWS Services
- Can put logs from CloudTrail into CloudWatch Logs
- If a resource is deleted in AWS, look into CloudTrail first!

# CloudTrail continued...

- CloudTrail shows the past 90 days of activity
- The default UI only shows “Create”, “Modify” or “Delete” events
- **CloudTrail Trail:**
  - Get a detailed list of all the events you choose
  - Ability to store these events in S3 for further analysis
  - Can be region specific or global
- CloudTrail Logs have SSE-S3 encryption when placed into S3
- Control access to S3 using IAM, Bucket Policy, etc...

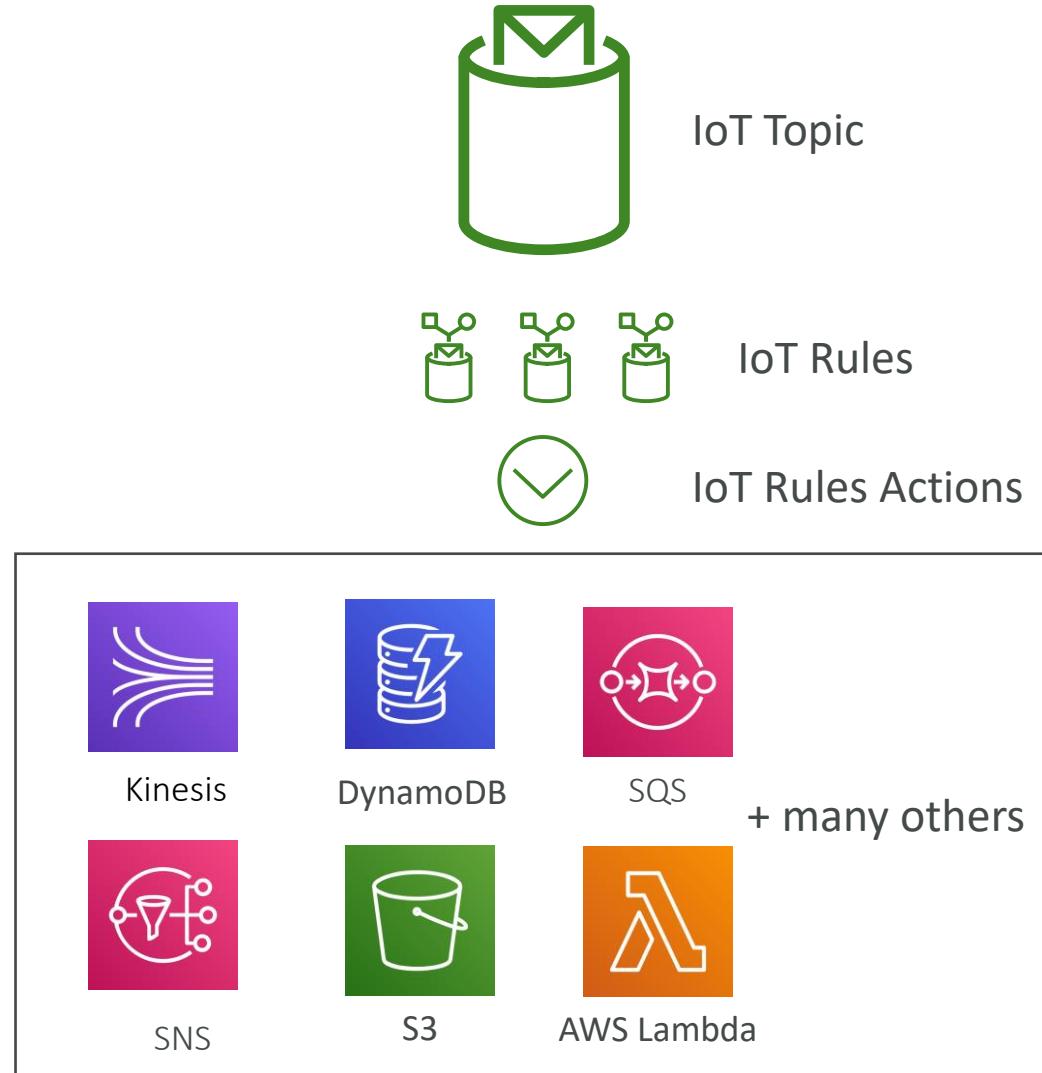
# VPC Endpoints

- Endpoints allow you to connect to AWS Services using a private network instead of the public www network
- They scale horizontally and are redundant
- They remove the need of IGW, NAT, etc... to access AWS Services
- **Gateway:** provisions a target and must be used in a route table  
ONLY S3 and DynamoDB
- **Interface:** provisions an ENI (private IP address) as an entry point (must attach security group) – most AWS services  
Also called VPC PrivateLink

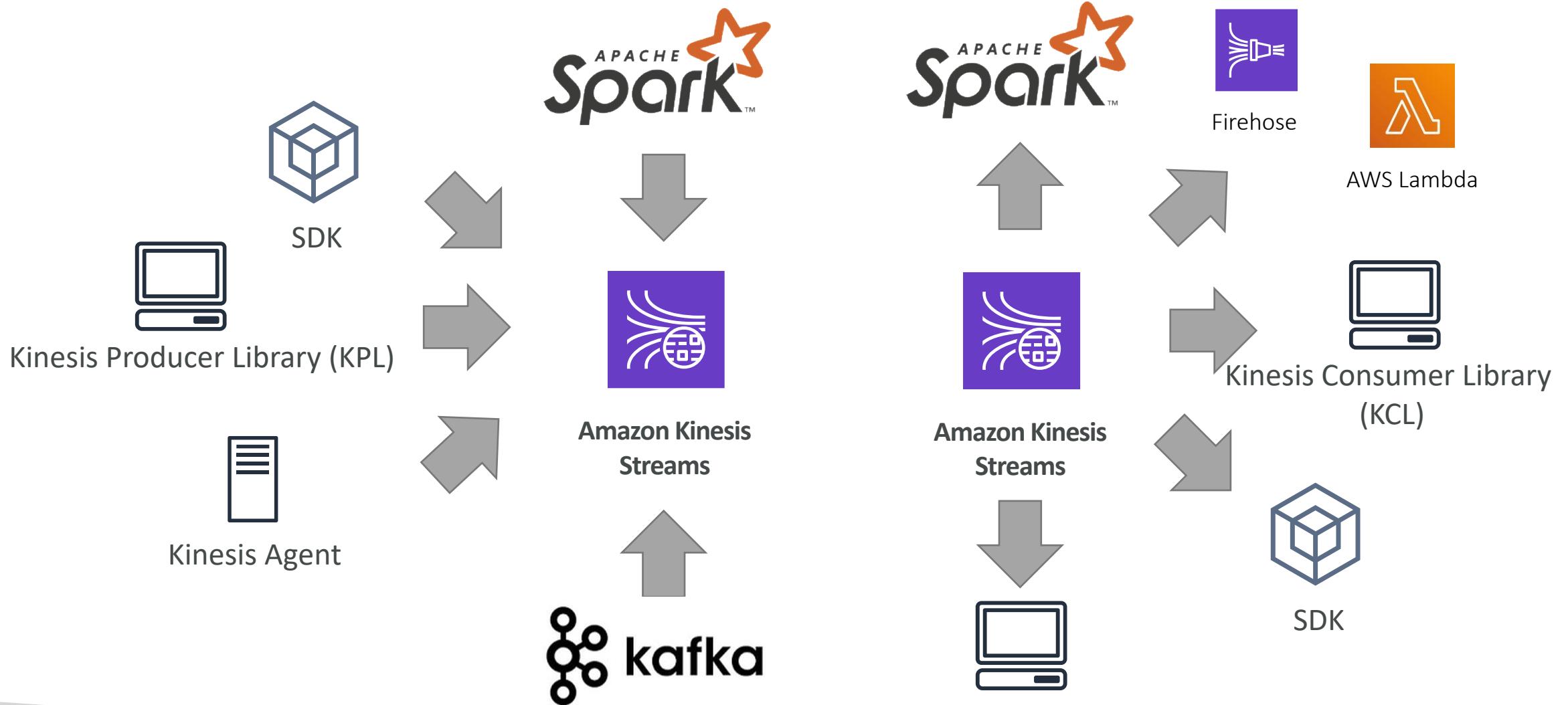


# Everything Else

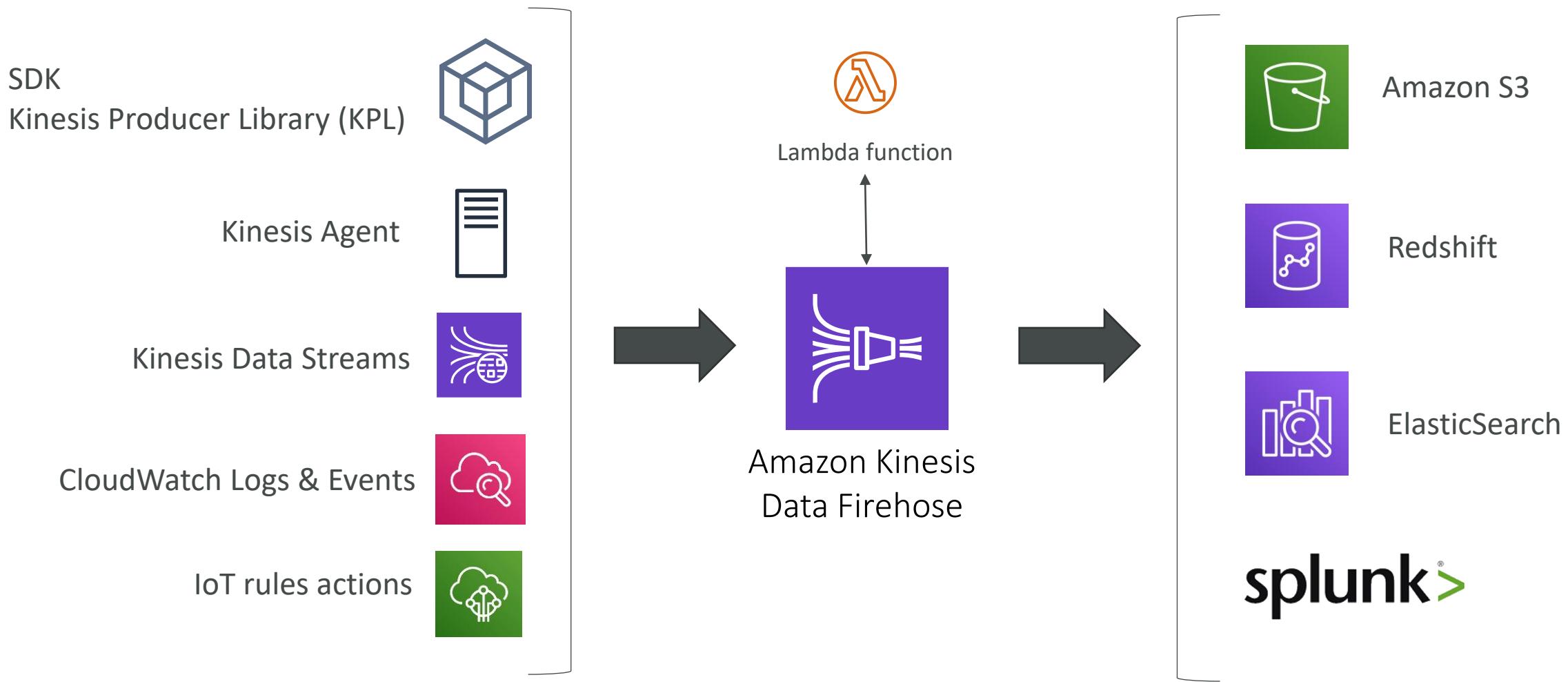
# IoT



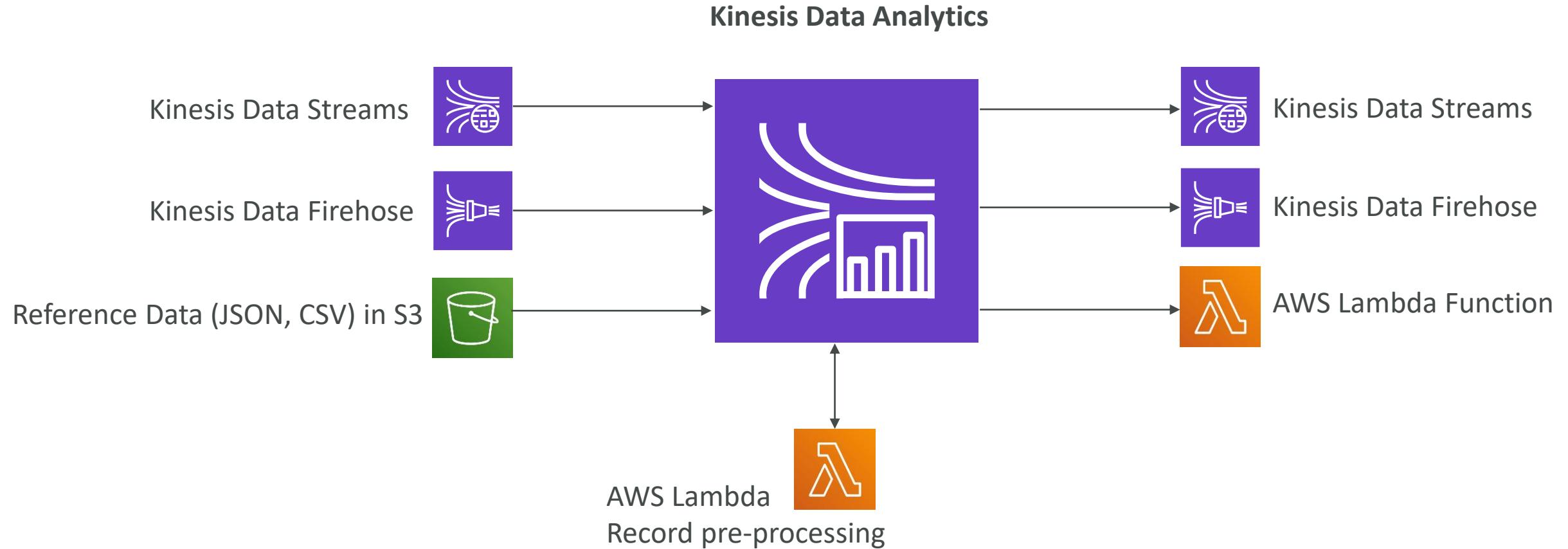
# Kinesis Data Streams



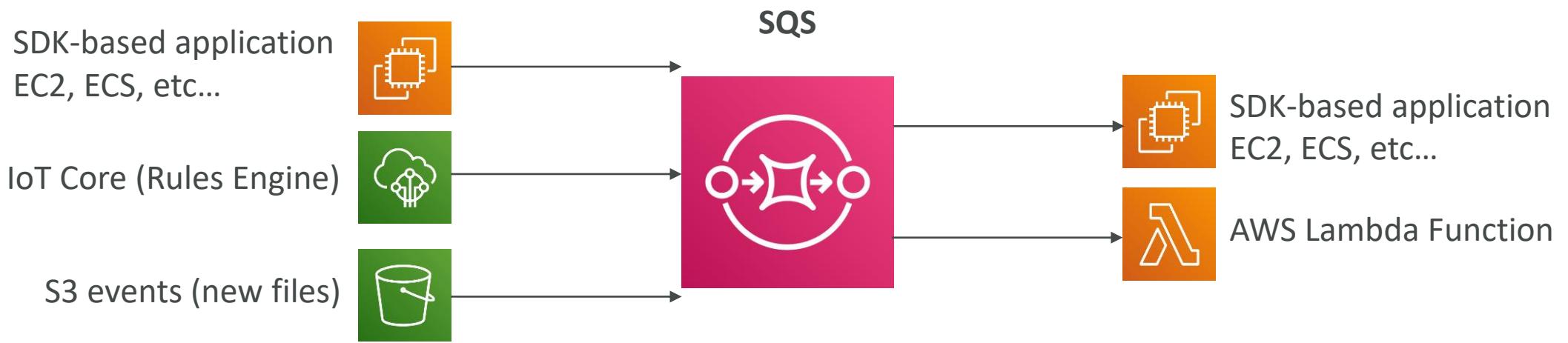
# Kinesis Data Firehose



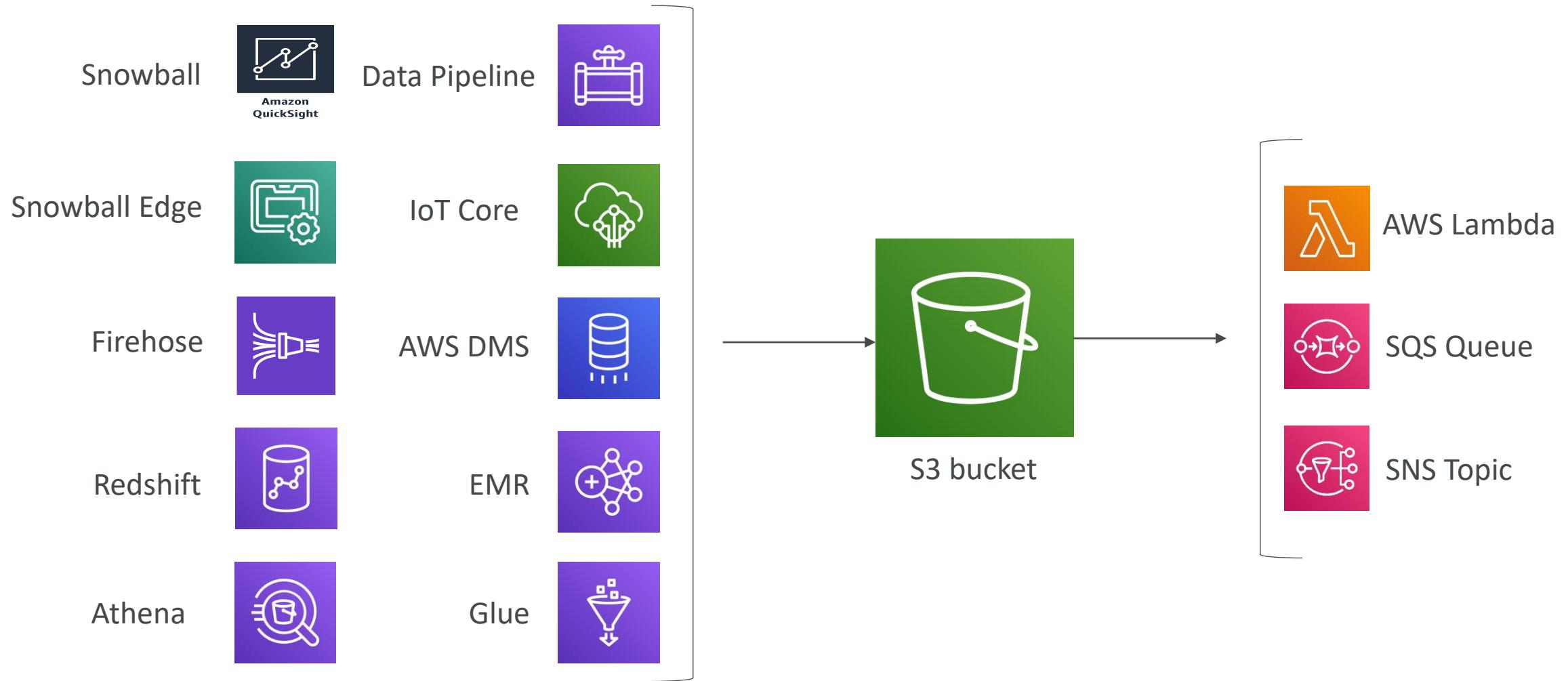
# Kinesis Data Analytics



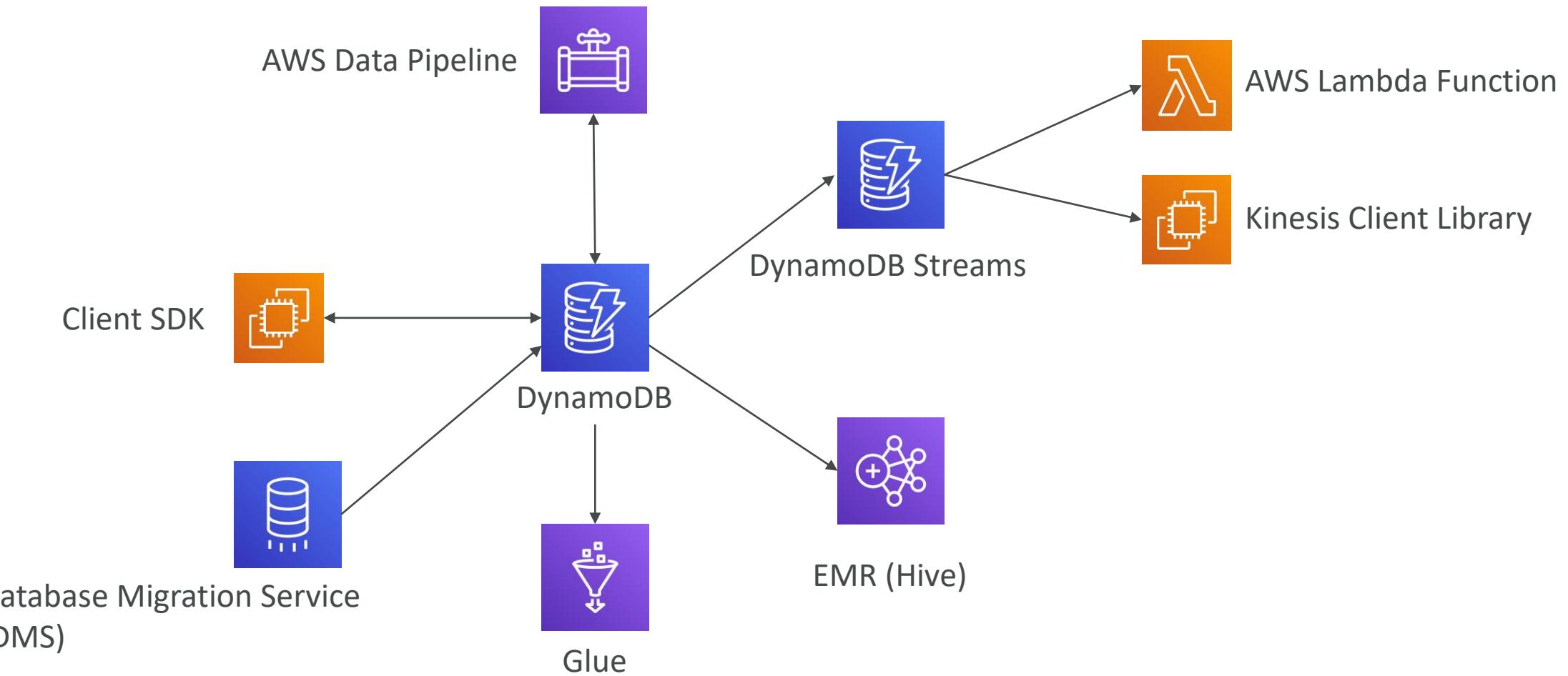
# SQS



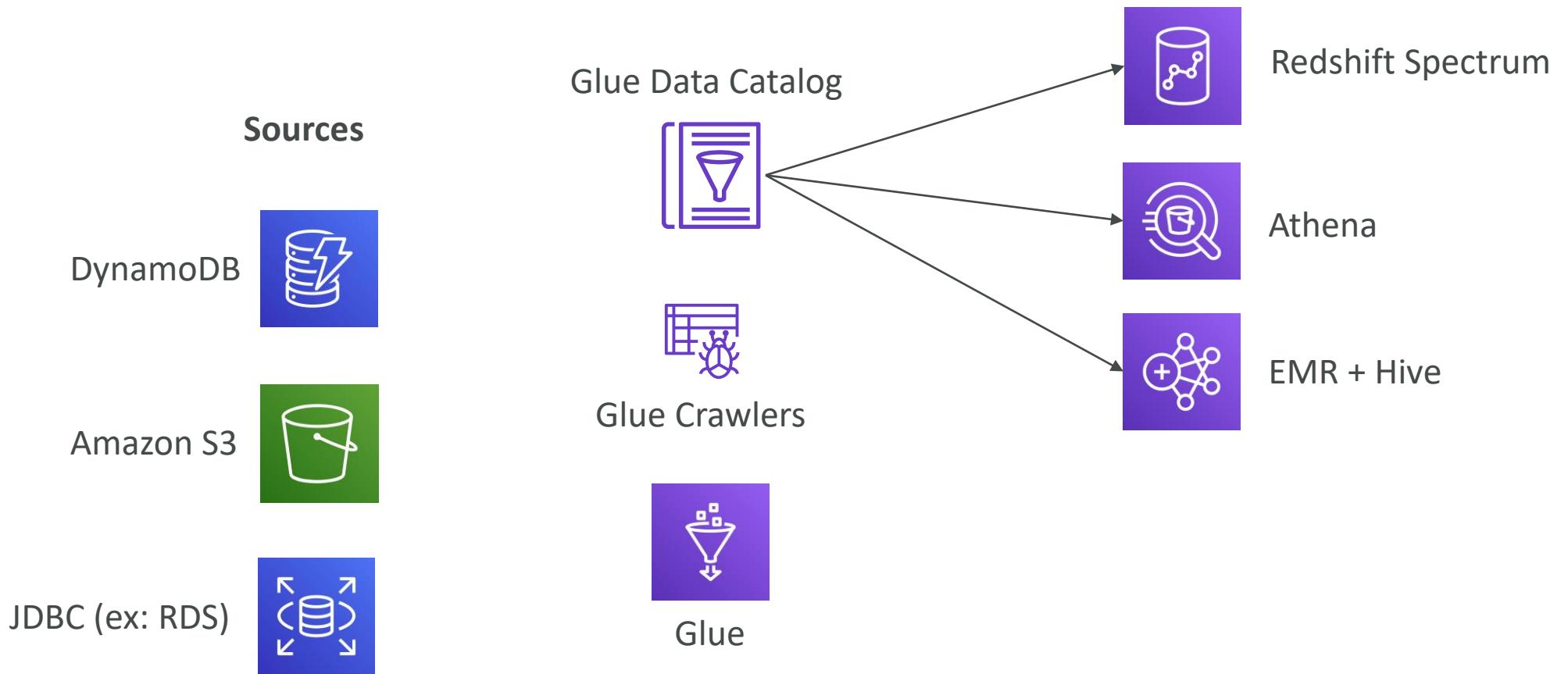
# S3



# DynamoDB



# Glue



# EMR

Glue Data Catalog



Amazon S3 / EMRFS



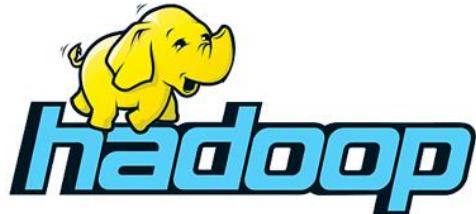
DynamoDB



Apache Ranger on EC2



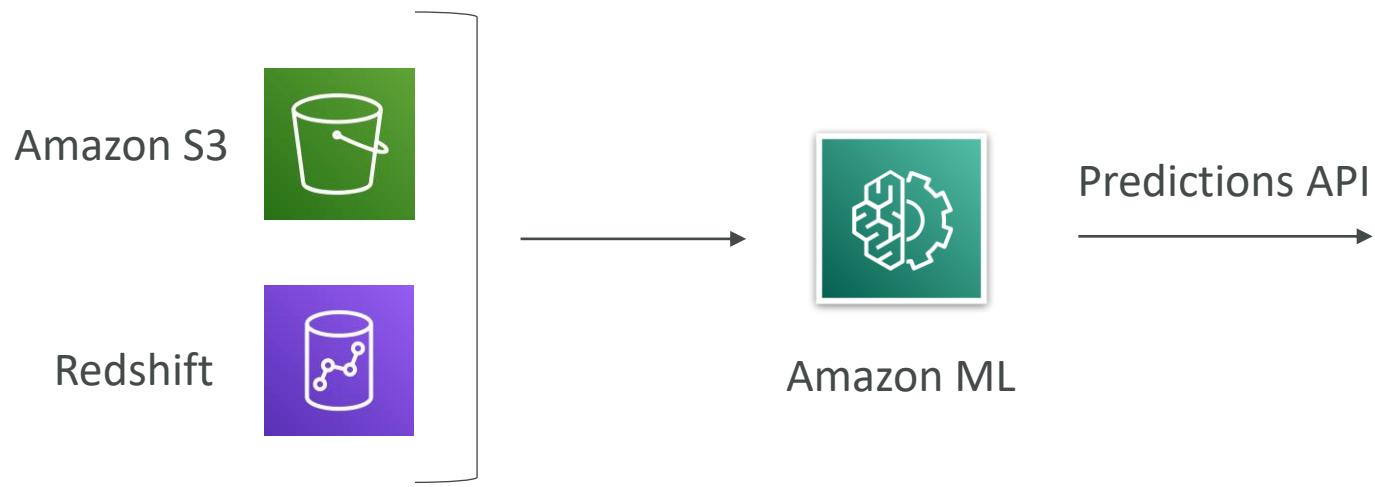
EMR



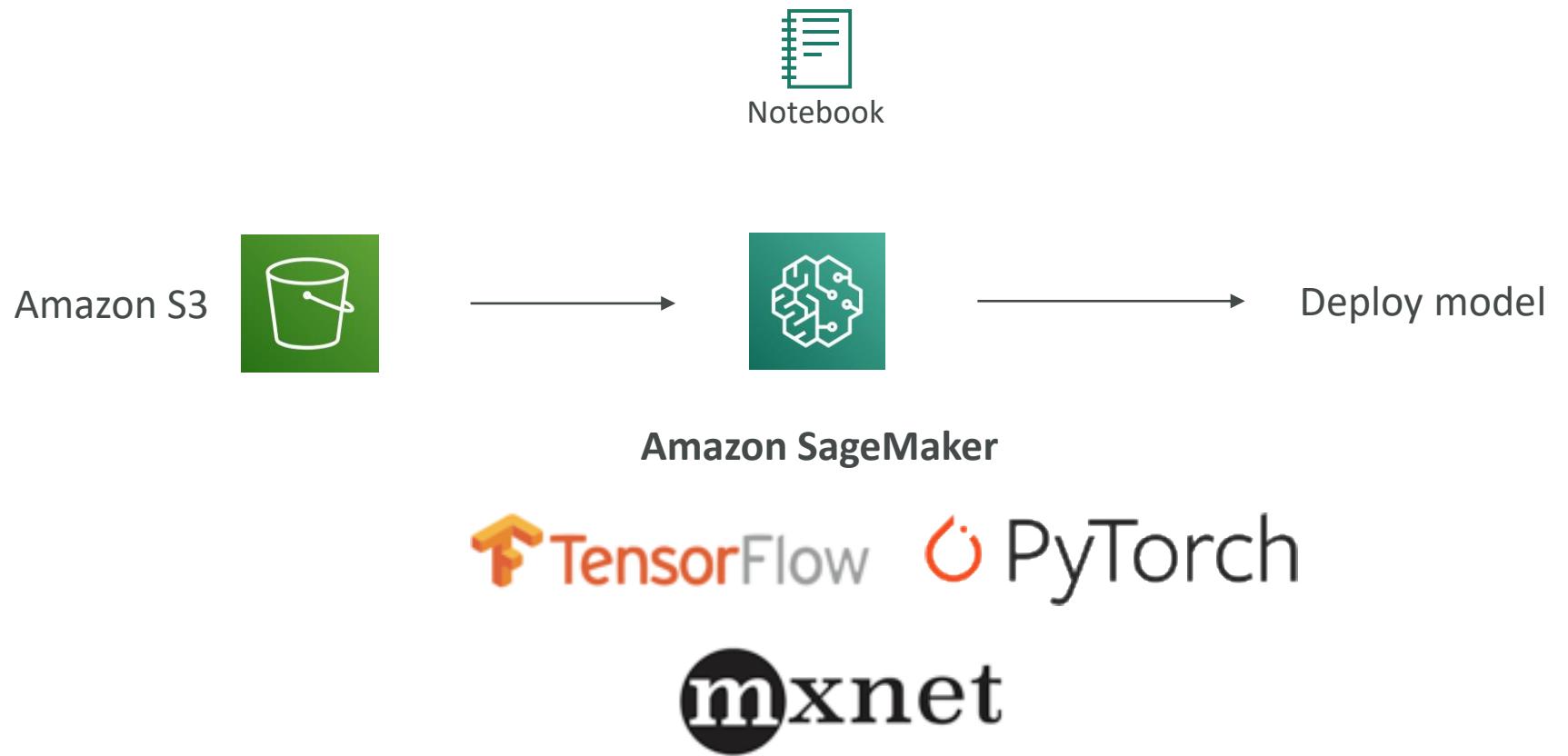
Flink



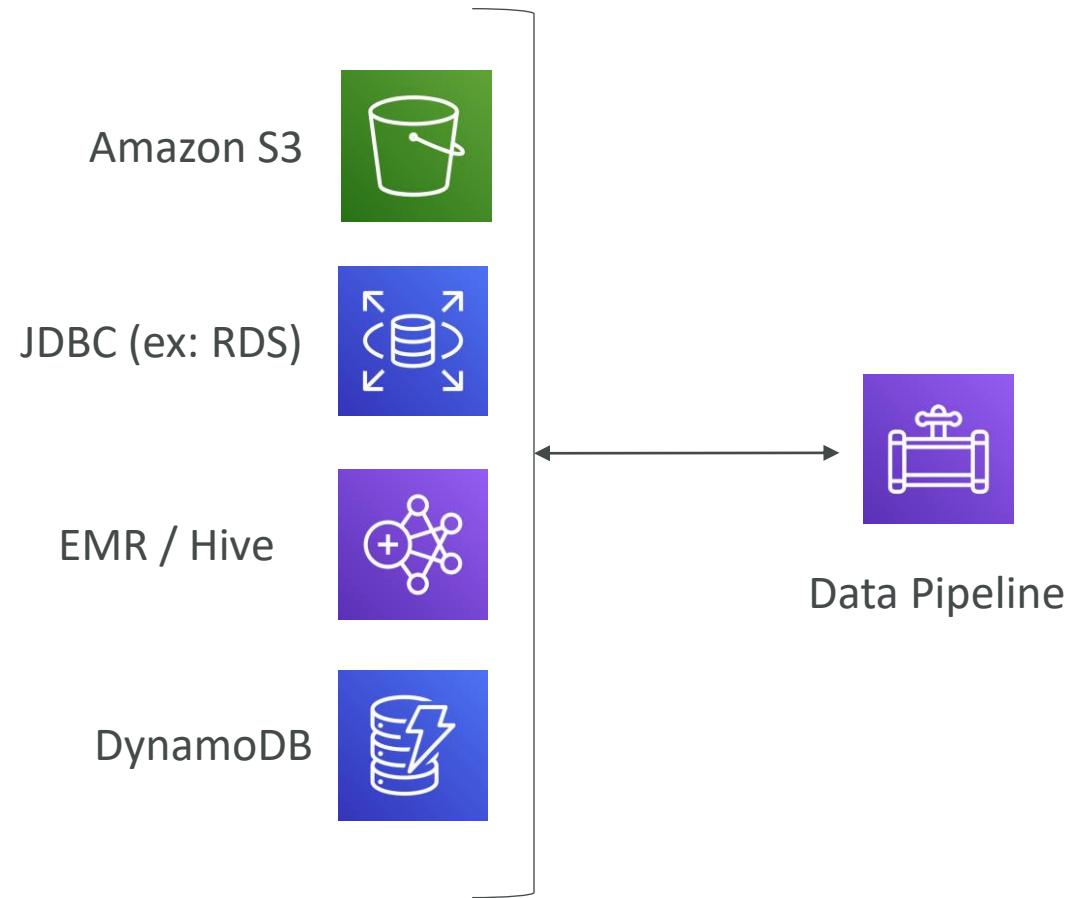
# Amazon Machine Learning (ML) (deprecated)



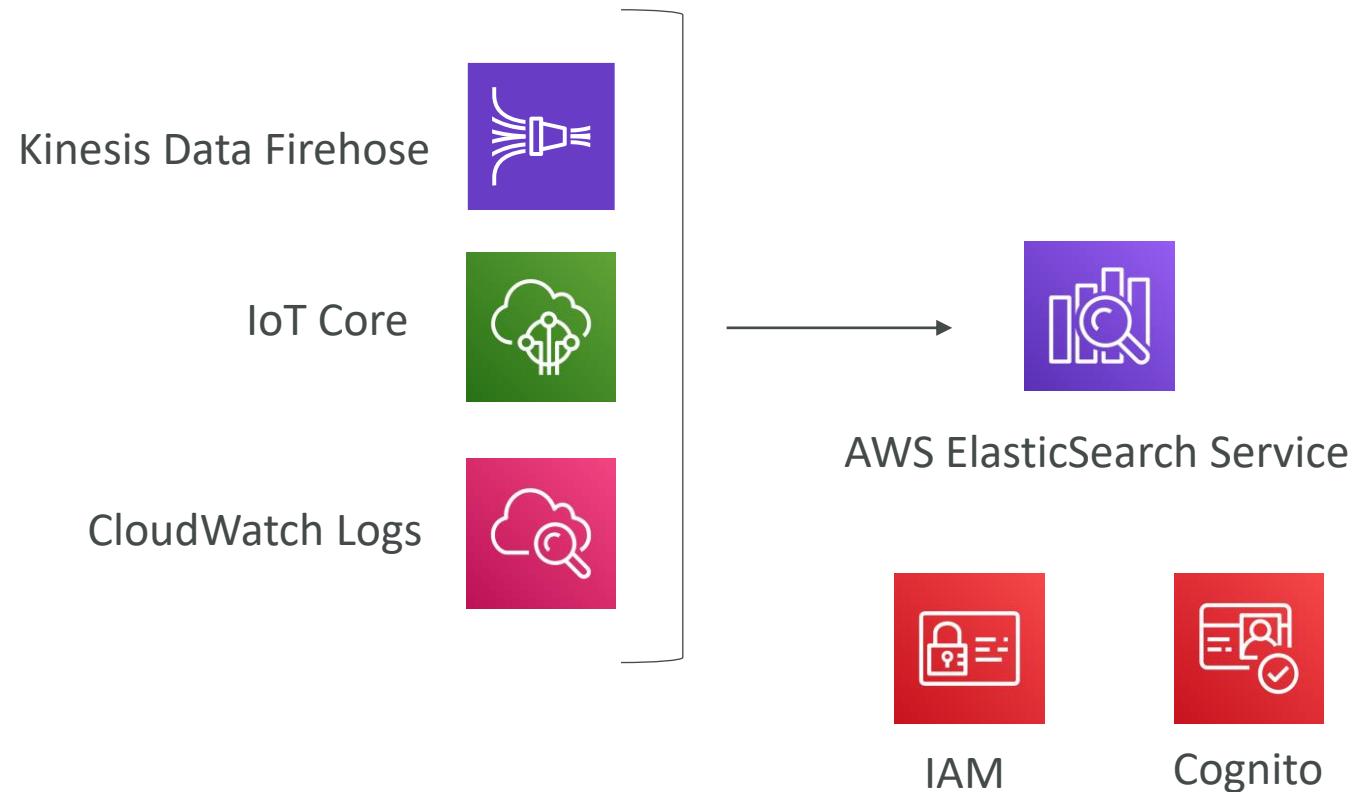
# Amazon SageMaker



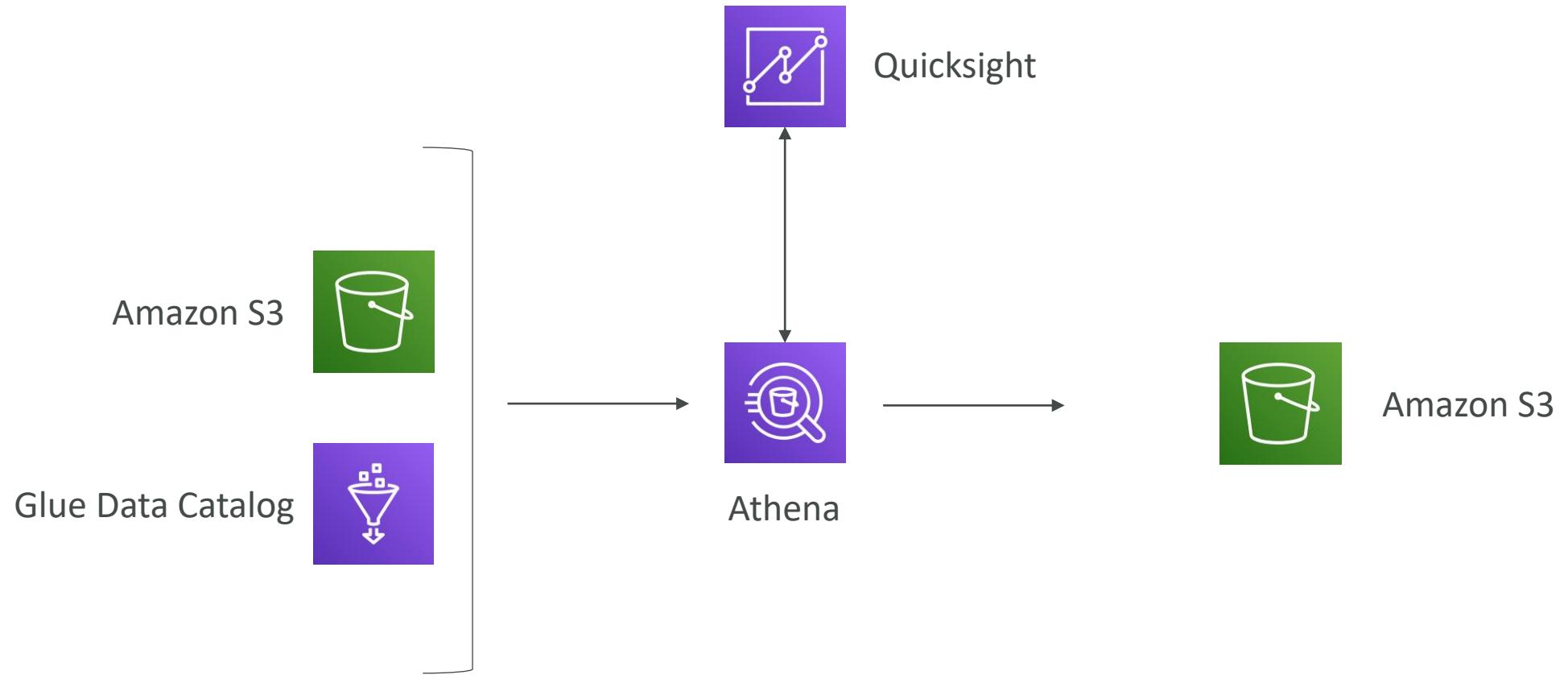
# AWS Data Pipeline



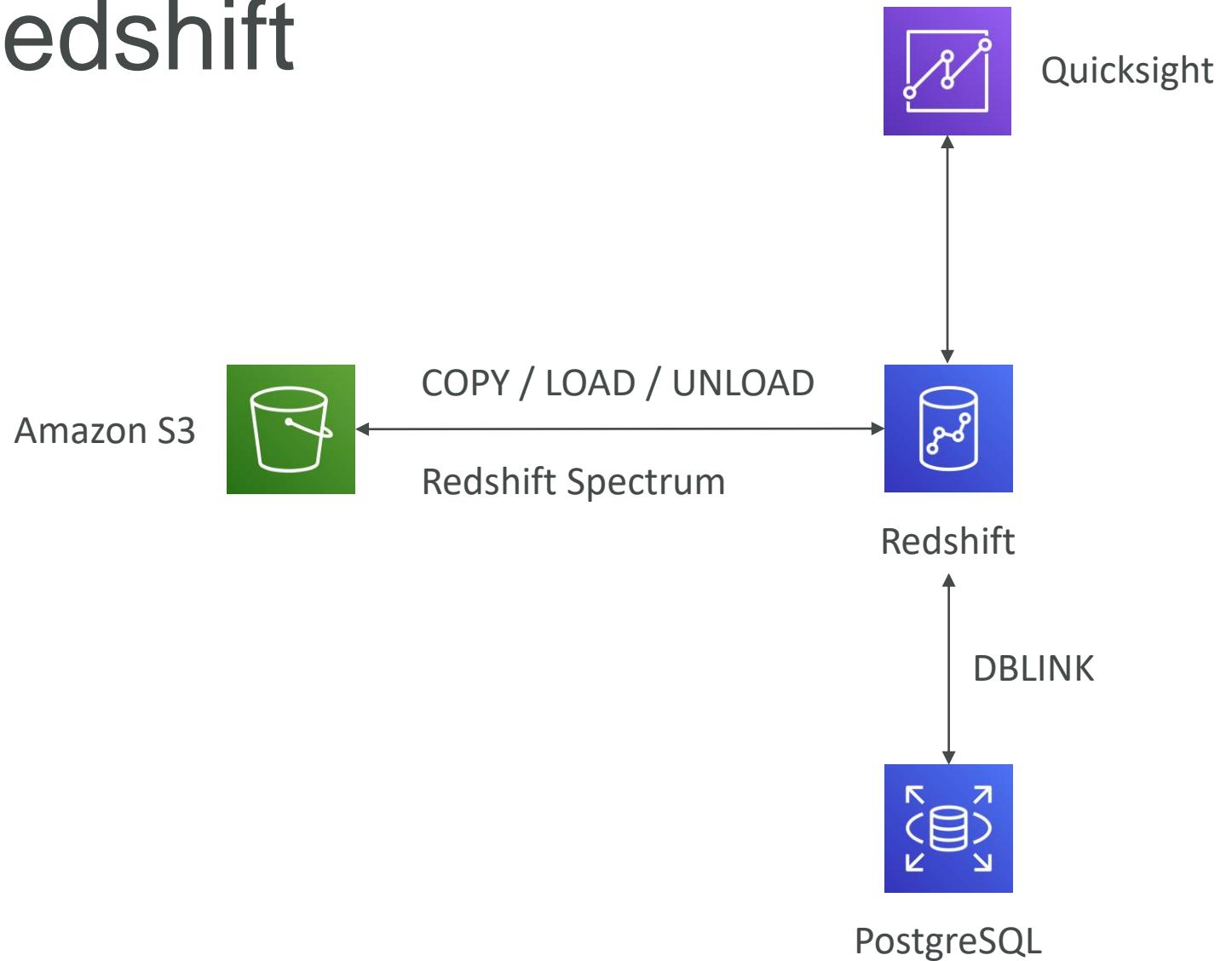
# ElasticSearch Service



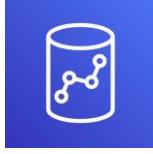
# Athena



# Redshift



# Quicksight

RDS / JDBC	
Redshift	
Athena	
Amazon S3	



Quicksight

# AWS Instance Types

- General Purpose: T2, T3, M4, M5
- Compute Optimized: C4, C5
  - Batch processing, Distributed analytics, Machine / Deep Learning Inference
- Memory Optimized: R4, R5, X1, Z1d
  - High performance database, In memory database, Real time big data analytics
- Accelerated Computing: P2, P3, G3, F1
  - GPU instances, Machine or Deep Learning, High Performance Computing
- Storage Optimized: H1, I3, D2
  - Distributed File System (HDFS), NFS, Map Reduce, Apache Kafka, Redshift

# EC2 in Big Data

- On demand, Spot & Reserved instances:
  - Spot: can tolerate loss, low cost => checkpointing feature (ML, etc)
  - Reserved: long running clusters, databases (over a year)
  - On demand: remaining workloads
- Auto Scaling:
  - Leverage for EMR, etc
  - Automated for DynamoDB, Auto Scaling Groups, etc...
- EC2 is behind EMR
  - Master Nodes
  - Compute Nodes (contain data) + Tasks Nodes (do not contain data)

# Preparing for the exam

# Exam Tips

The strategic aspect...

# Take your time reading the question

- Look for key words about requirements

A large news website needs to produce personalized recommendations for articles **to its readers**, by training a machine learning model **on a daily basis** using historical click data. The influx of this data is fairly constant, except during major elections when **traffic to the site spikes considerably**.

Which system would provide the most **cost-effective** and **reliable** solution?

# Pace yourself

- You have 170 minutes and about 65 questions
- That's about 2 ½ minutes per question!
- Try not to get stressed out... that's enough time to read and understand each question



# Flag questions for later review

- If you're stumped on something, don't spend too much time on it
- Select your best guess, and mark it for review
- Then use any time you have at the end to go back and reconsider
- Flag questions you're not totally sure about, too.



# Arrive prepared

- Get a good night's sleep
- Do whatever you need to do to stay alert – this test requires stamina
- Go to the bathroom before arriving
- Arrive early – the exam location may be hard to find



# Additional prep resources

- AWS Big Data White Paper (“Big Data Analytics Options on AWS”)
- AWS’s free online prep course
- White papers on Kinesis, Database Migration Service, Migrating Applications to AWS
- Exam overview from AWS
- This shouldn’t be your first certification exam
- Take our practice exam

# State of learning checkpoint

- Let's look how far we've gone on our learning journey
- <https://aws.amazon.com/certification/certified-big-data-specialty/>

# How will the exam work?

- You'll have to register online at <https://www.aws.training/>
- Fee for the exam is 300 USD
- Provide two identity documents (ID, Credit Card, details are in emails sent to you)
- No notes are allowed, no pen is allowed, no speaking
- ~65 questions will be asked in 170 minutes
- At the end you can optionally review all the questions / answers
  
- You will know right away if you passed / failed the exams
- You will not know which answers were right / wrong
- You will know the overall score a few days later (email notification)
- The pass score is not provided, but some people passed with 60%
- If you fail, you can retake the exam again 14 days later