AWS Analytics Overview

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

amazon web services

Agenda

- Amazon Kinesis Platform Overview
- Glue – Upcoming managed ETL Service
- Quick and Powerful Analytics using Athena and QuickSight
- Database State of the Union
- Database Migration Service

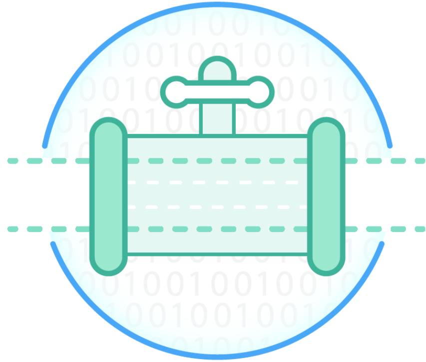
amazon web services

Amazon Kinesis Platform

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

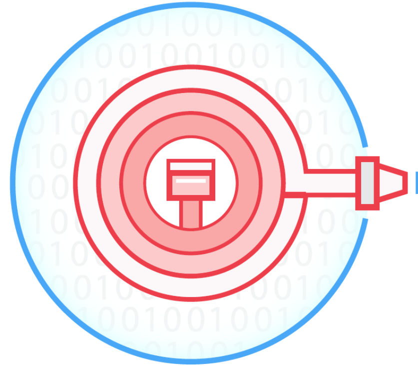
Amazon Kinesis: Streaming data made easy

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



Amazon Kinesis Streams

Build your own custom applications that process or analyze streaming data



Amazon Kinesis Firehose

Easily load massive volumes of streaming data into Amazon S3 and Amazon Redshift

Amazon Kinesis Analytics

Easily analyze data streams using standard SQL queries

Amazon Confidential

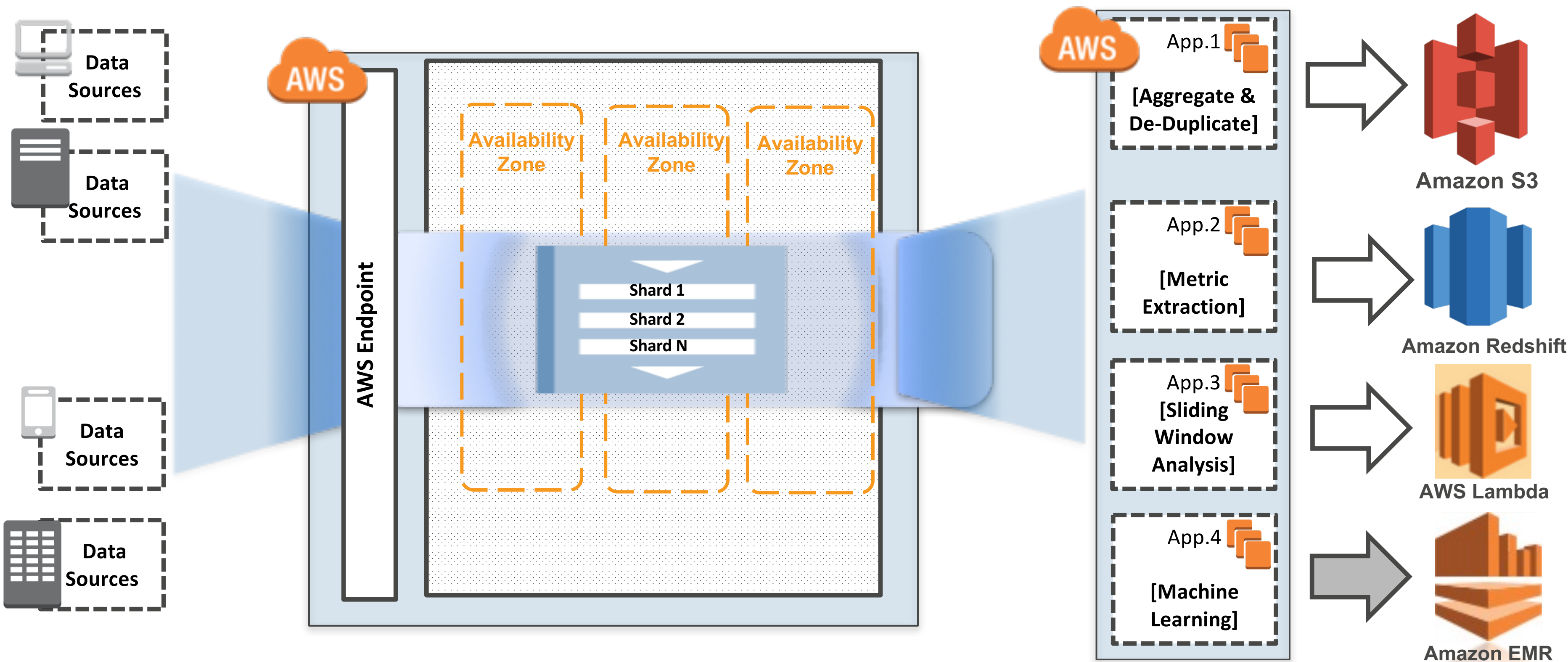
amazon web services

Amazon Kinesis Streams

amazon web services

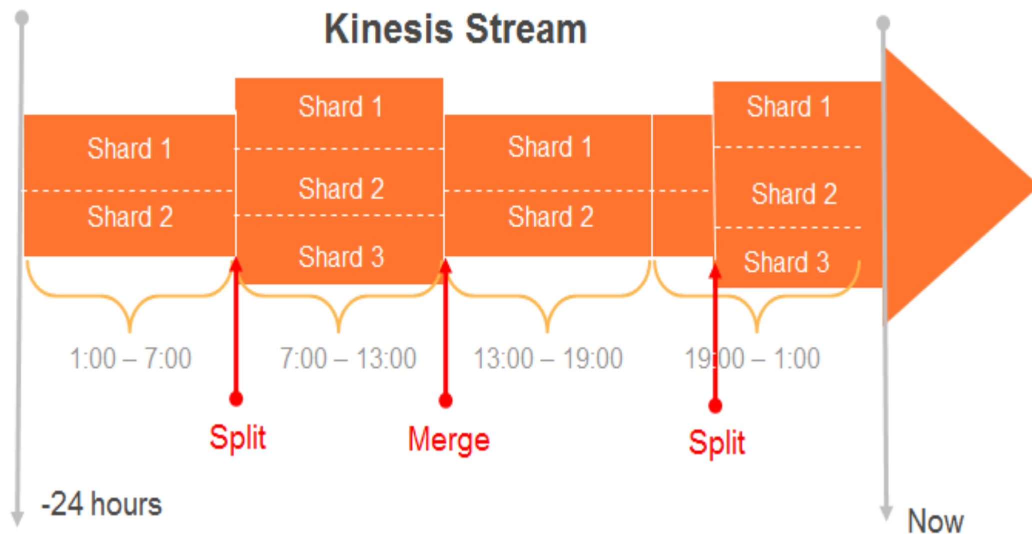
Amazon Kinesis Streams

Managed service for real-time streaming



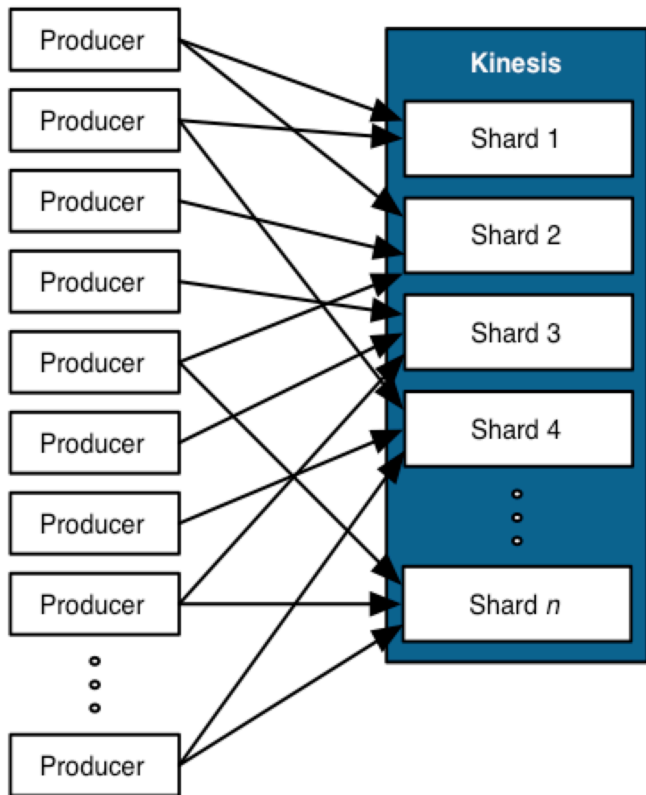
Amazon Kinesis Streams

Managed ability to capture and store data



- Streams are made of **shards**
- Each shard ingests up to 1 MB/sec, and 1000 records/sec
- Each shard emits up to 2 MB/sec
- All data is **stored for 24 hours by default**; storage can **be extended for up to 7 days**
- **Scale** Amazon Kinesis Streams using scaling util
- **Replay** data inside of 24-hour window

Putting Data in Amazon Kinesis



Simple Put Interface to put Data in Amazon

Kinesis Producers use a **PUT** call to store data in a Stream. Each record <= 1 MB

PutRecord {Data,StreamName,PartitionKey}

PutRecords {Records{Data,PartitionKey}, StreamName}

A **Partition Key** is supplied by producer and used to distribute (MD5 hash) the PUTs across (hash key range) of Shards

A unique **Sequence #** is returned to the Producer upon a successful PUT call

amazon web services

Putting Data in Amazon Kinesis

Determine your Partition Key strategy

- Managed buffer or a streaming MapReduce
- Ensure High Cardinality for your shards

Provision Adequate Shards

- For ingress needs
- Egress needs for all consuming applications: if more than 2 simultaneous applications
- Include head-room for catching up with data in stream

amazon web services

Record Order and Multiple Shards

Unordered processing

- Randomize partition key to distribute events over many shards and use multiple workers

Exact order processing

- Control the partition key to ensure events are grouped onto the same shard and read by the same worker.

Need both? Get global sequence number

Scaling Amazon Kinesis Shards

```
java -cp
KinesisScalingUtils.jar-complete.jar
-Dstream-name=MyStream
-Dscaling-action=scaleUp
-Dcount=10
-Dregion=eu-west-1 ScalingClient
```

Options:

- stream-name - The name of the Stream to be scaled
- scaling-action - The action to be taken to scale. Must be one of "scaleUp", "scaleDown" or "resize"
- count - Number of shards by which to absolutely scale up or down, or resize to or:
- pct - Percentage of the existing number of shards by which to scale up or down

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

Putting Data in Amazon Kinesis

Pre-batch before Puts for better efficiency

- Consider Flume,FLuntD as collectors/agents
  https://github.com/awslabs/aws-fluent-plugin-kinesis

Make a tweak to your existing logging

- log4j appender optionhttps://github.com/awslabs/kinesis-log4j-appender

amazon web services

Amazon Kinesis Producer Library

Writes to one or more Amazon Kinesis Streams with an automatic and configurable Retry Mechanism

Collect Records and uses PutRecords to write multiple records to multiple shards per Request

Aggregates user records to increase payload size and improve throughput

Integrates seamlessly with the Amazon KCL to de-aggregate Batched Records

Submits Amazon CloudWatch metrics on your behalf to provide visibility into producer performance

https://github.com/awslabs/amazon-kinesis-producer/tree/master/java

Amazon Kinesis Client Library

Build Kinesis Application with Client Library

Open source client library available for Java, Ruby, Python, Node.JS dev

Deploy on your set of EC2 instances

KCL Application includes 3 components:

- Record Processor Factory – Creates the record processor
- Record Processor – The processor unit that processes data from a shard from Amazon Kinesis Stream
- Worker – The processing unit that maps to each application instance

IRecordProcessor: The heart of Kinesis Client Library

KCL uses IRcordProcessor interface to communicate with your application

A Kinesis application must implement KCL's IRecordProcessor interface

Contains the business logic for processing the data retrieved from the Kinesis stream

```
Class TwitterTrendsShardProcessor implements IRecordProcessor {

   public TwitterTrendsShardProcessor() { … }

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

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

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

amazon web services

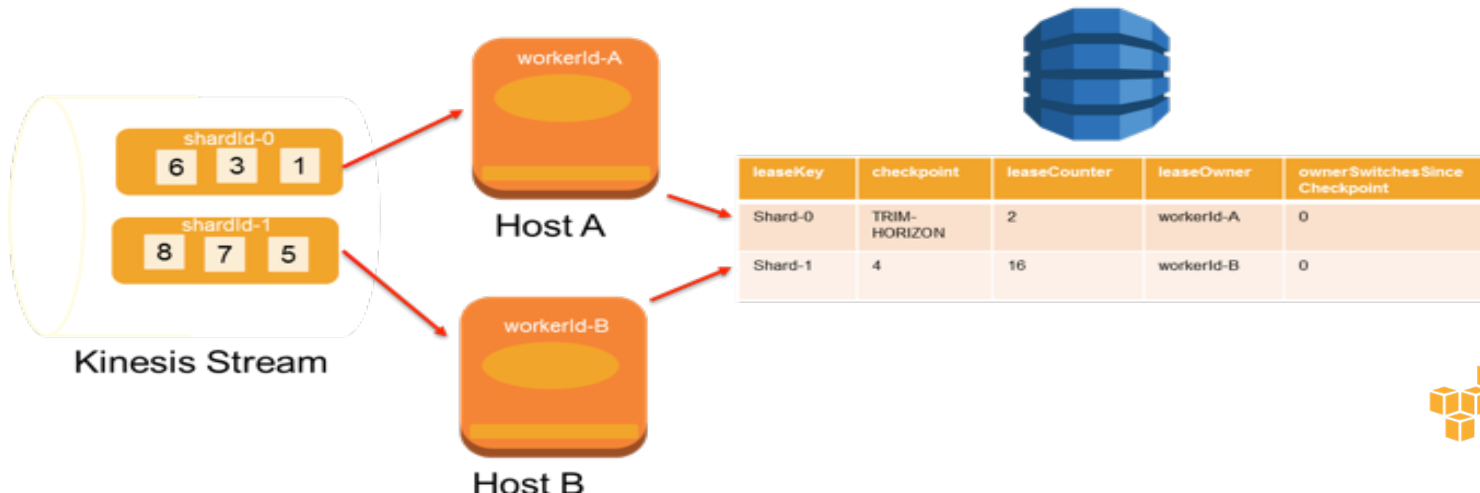
State Management with Kinesis Client Library

One record processor maps to one shard and processes data records from that shard

One worker maps to one or more record processors

Balances shard-worker associations when worker / instance counts change

Balances shard-worker associations when shards split or merge



leaseKey	checkpoint	leaseCounter	leaseOwner	ownerSwitchesSince Checkpoint
Shard-0	TRIM-HORIZON	2	workerId-A	0
Shard-1	4	16	workerId-B	0

Sending & Reading Data from Kinesis Streams

Sending

AWS SDK

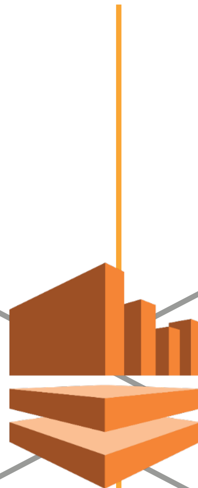
Kinesis Producer Library

AWS Mobile SDK

LOG4J

Flume

Fluentd



Consuming

Get* APIs

Kinesis Client Library
+
Connector Library

AWS Lambda

Amazon Elastic MapReduce

Apache Storm

Apache Spark

Amazon Kinesis Firehose

Load streaming data into Amazon S3, Amazon Redshift, and Amazon ES

Zero administration: Deliver streaming data into Amazon S3, Amazon Redshift, Amazon ES, and other destinations **without writing an application or managing infrastructure**.

Direct-to-data store integration: Batch, compress, and **encrypt** streaming data for delivery into data destinations **in as little as 60 seconds** using simple configurations.

Seamless elasticity: Seamlessly scale to match data throughput without intervention.

Capture IT and app logs, device and sensor data, and more

Enable near-real time analytics using existing tools

- Send data from IT infra, mobile devices, sensors
- Integrated with AWS SDK, agents, and AWS IoT

- Fully managed service to capture streaming data
- Elastic w/o resource provisioning
- Pay-as-you-go: 3.5 cents / GB transferred

- Batch, compress, and encrypt data before loads
- Loads data into Amazon Redshift tables by using the COPY command

Amazon Kinesis Firehose Features

- Destinations: S3, Redshift, Amazon ElasticSearch Service
- Knobs:
  - Buffer Size – minimum 1MB
  - Buffer Interval – minimum 1 minute
- Source Record Transformation (using Lambda)
- Compression
- Encryption

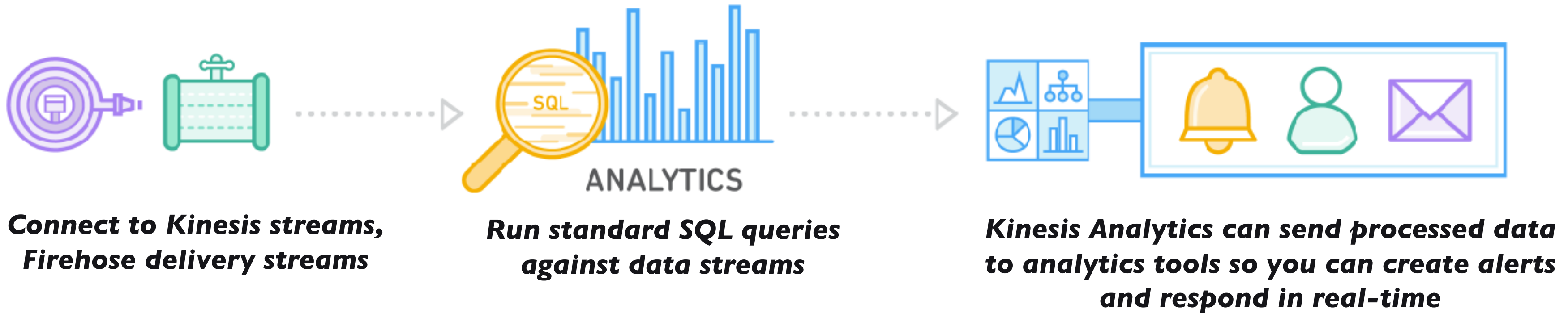
When to use Firehose or Amazon Kinesis Streams

Depends on your use case and nature of data

Amazon Kinesis Streams is for use cases that require **custom processing**, per incoming record, with sub-1 second processing latency, and a choice of stream processing frameworks.

Amazon Kinesis Firehose is for use cases that require zero administration, ability to **use existing analytics tools based on Amazon S3, Amazon Redshift and Amazon ES**, and a data latency of 60 seconds or higher.

Amazon Kinesis Analytics



Apply SQL on streams: Easily connect to a Kinesis Stream or Firehose Delivery Stream and apply SQL skills.

Build real-time applications: Perform continual processing on streaming big data with sub-second processing latencies.

Easy Scalability : Elastically scales to match data throughput.

Use SQL to build real-time applications

Connect to streaming source

Easily write SQL code to process streaming data

Continuously deliver SQL results

Connect to streaming source

- Streaming data sources include Amazon Kinesis Firehose or Streams
- Input formats include JSON, CSV, variable column, unstructured text
- Each input has a schema; schema is inferred but you can edit
- Reference data sources (Amazon S3) for data enrichment

Write SQL code

- Build streaming applications with one to many SQL statements
- Robust SQL support and advanced analytic functions
- Extensions to the SQL standard to work seamlessly with streaming data
- Support for at-least-once processing semantics

Continuously deliver SQL results

- Send processed data to multiple destinations
  - S3, Redshift, ElasticSearch Service (through Kinesis Firehose)
  - Kinesis Streams (with AWS Lambda integration for custom destinations)
- End to end processing speed as low as sub-second
- Separation of processing and data delivery

A simple streaming query

- Tweets about the AWS NYC Summit
- Selecting from a STREAM of tweets, *an in-application stream*
- Each row has a corresponding ROWTIME

```
SELECT STREAM ROWTIME, author, text
FROM Tweets
WHERE text LIKE '%#AWSNYCSummit%'
```

A streaming table is a STREAM

- In relational databases, you work with SQL tables
- With Kinesis Analytics, you work with STREAMs
- SELECT, INSERT, and CREATE can be used with STREAMs

```
INSERT INTO Tweets
SELECT …
```

```
CREATE STREAM Tweets
      (author VARCHAR(20),
       text VARCHAR(140));
```

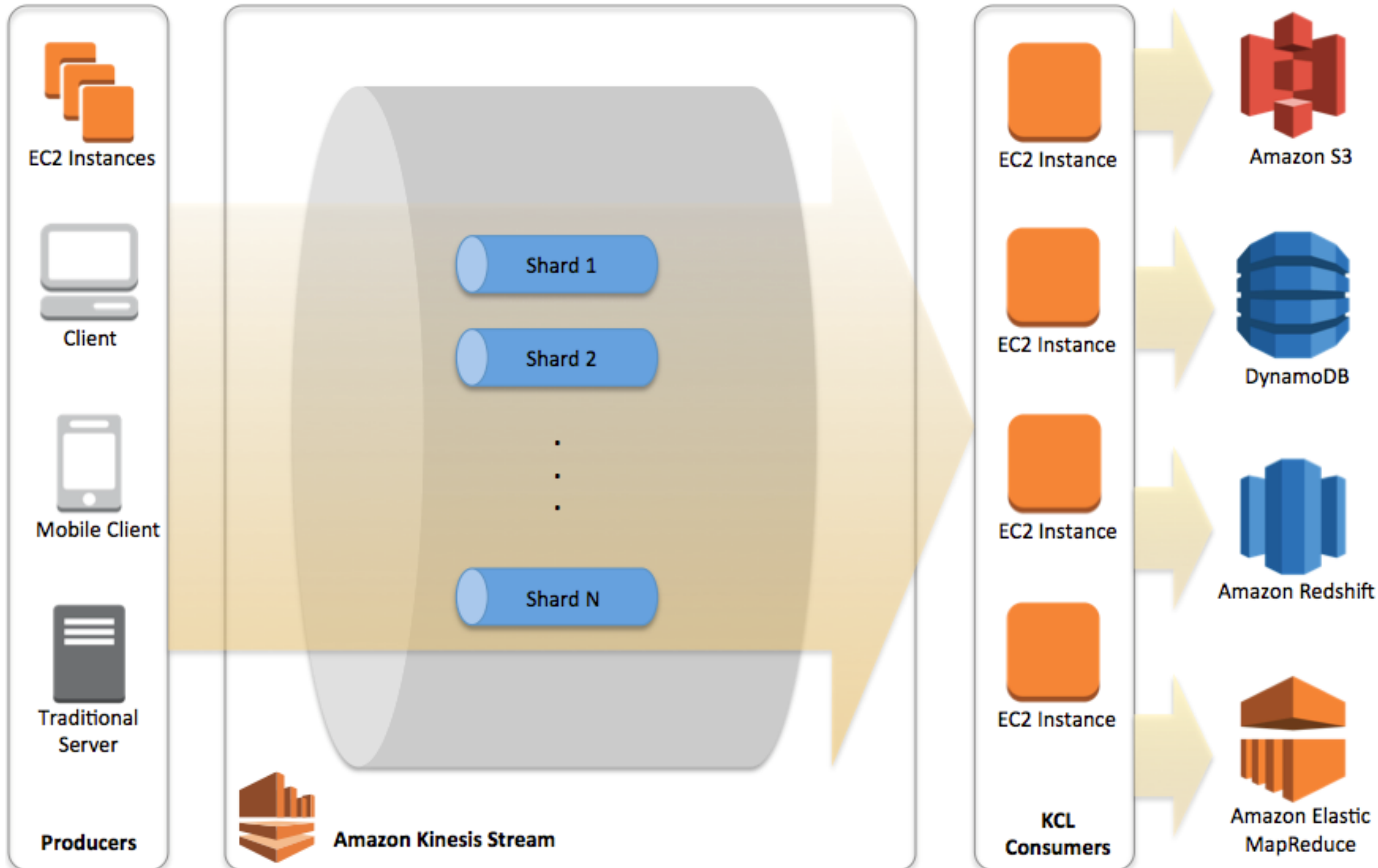
Writing queries on an unbounded data sets

- Streams are unbounded data sets
- Need continuous queries, *row by row* or *across rows*
- WINDOWS define a start and end to the query

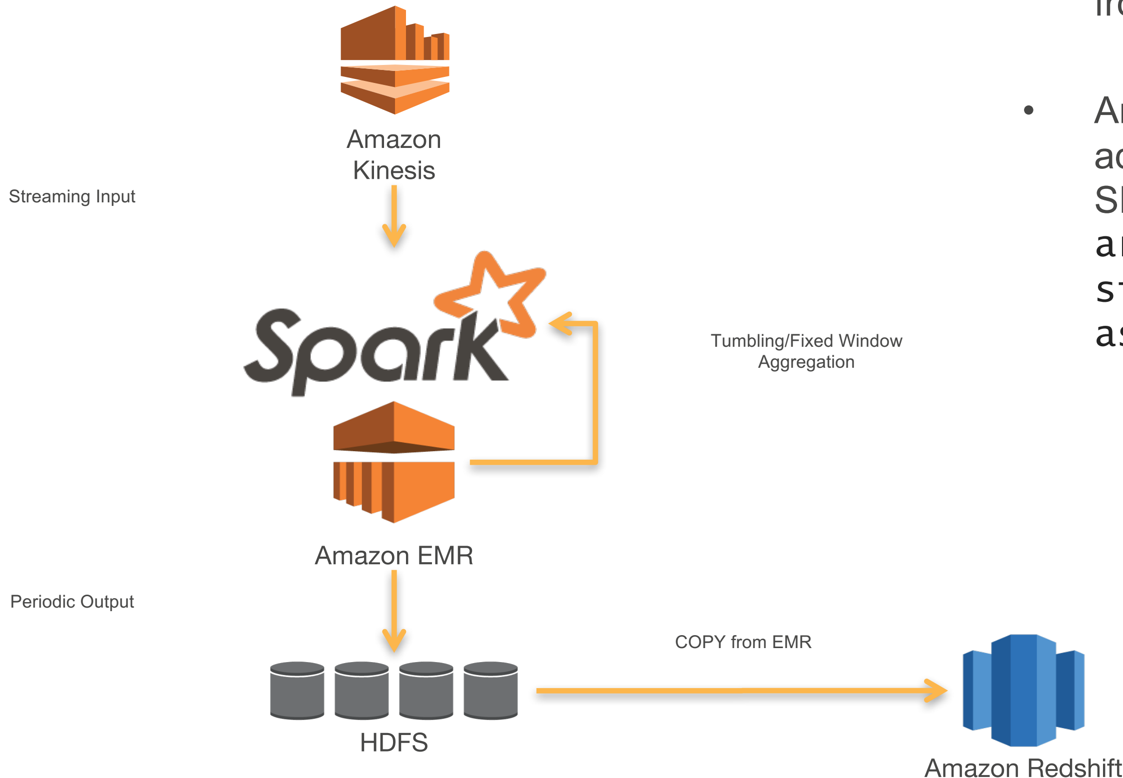
```
SELECT STREAM author,
    count(author) OVER ONE_MINUTE
FROM Tweets
WINDOW ONE_MINUTE AS
(PARTITION BY author
RANGE INTERVAL '1' MINUTE PRECEDING);
```

Streams processing

Build KCL Apps



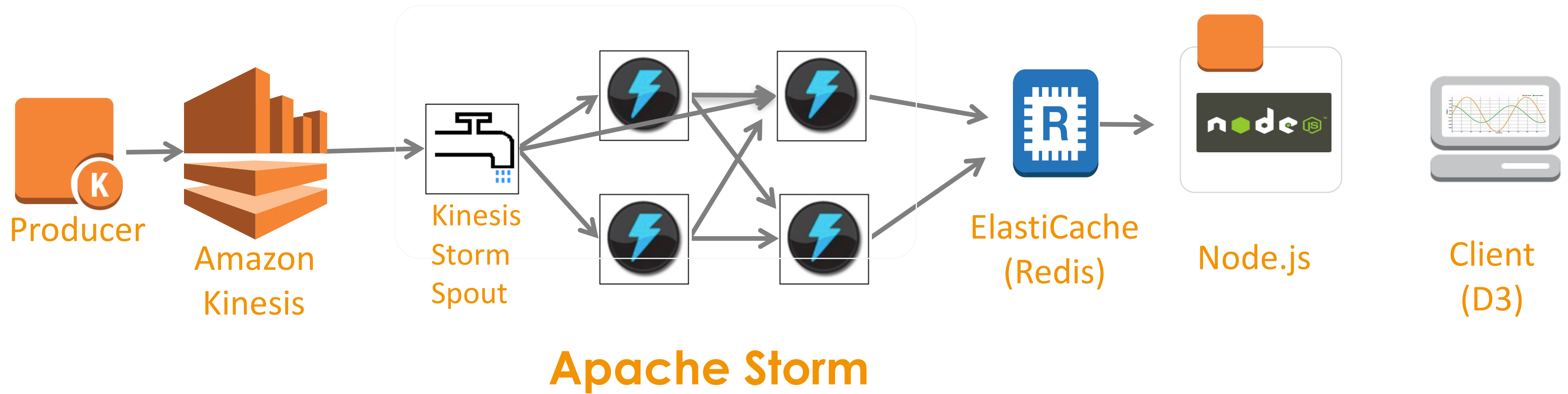
Spark + Kinesis



- Spark Streaming can read directly from an Amazon Kinesis stream
- Amazon software license linking – add ASL dependency to SBT/MAVEN project, `artifactId = spark-streaming-kinesis-asl_2.10`

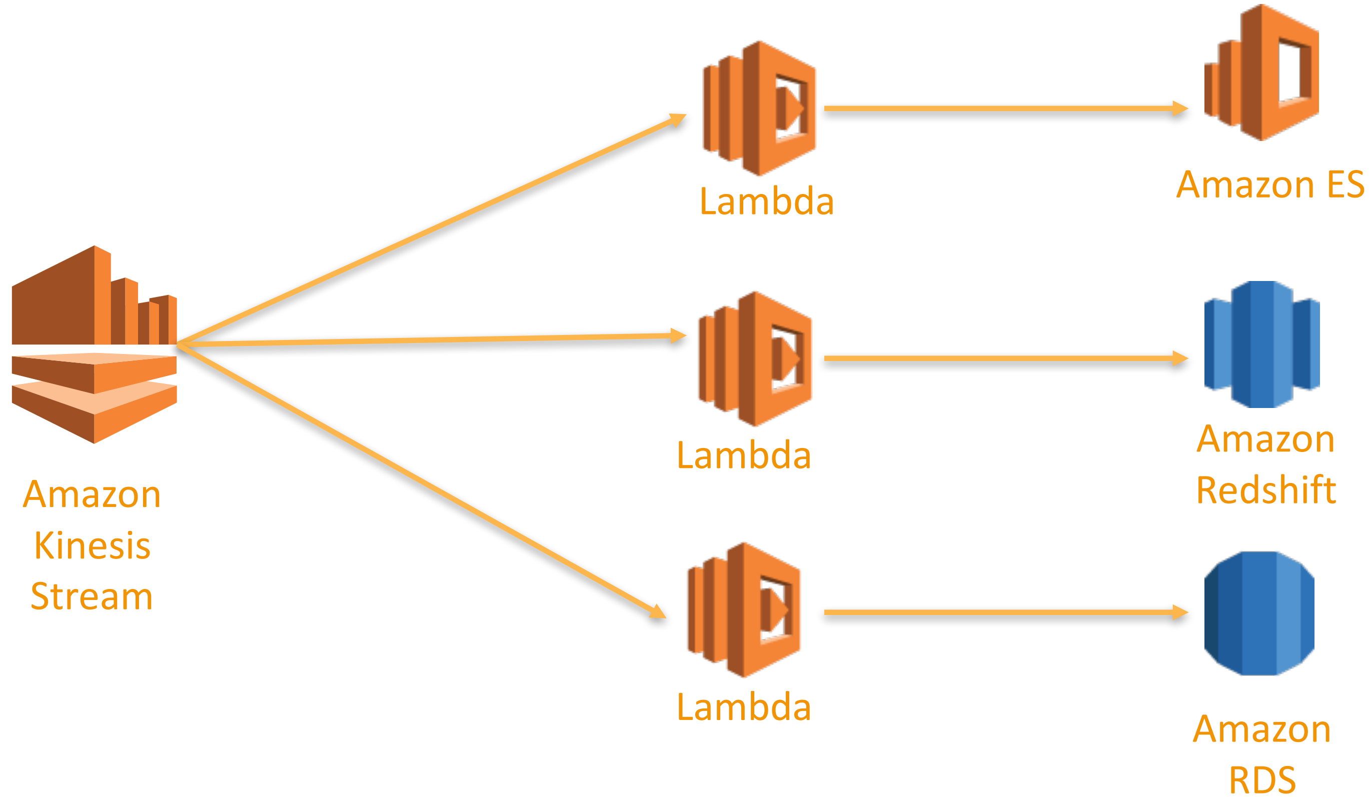
http://spark.apache.org/docs/latest/streaming-kinesis-integration.html

Real-Time Event Processing using Storm

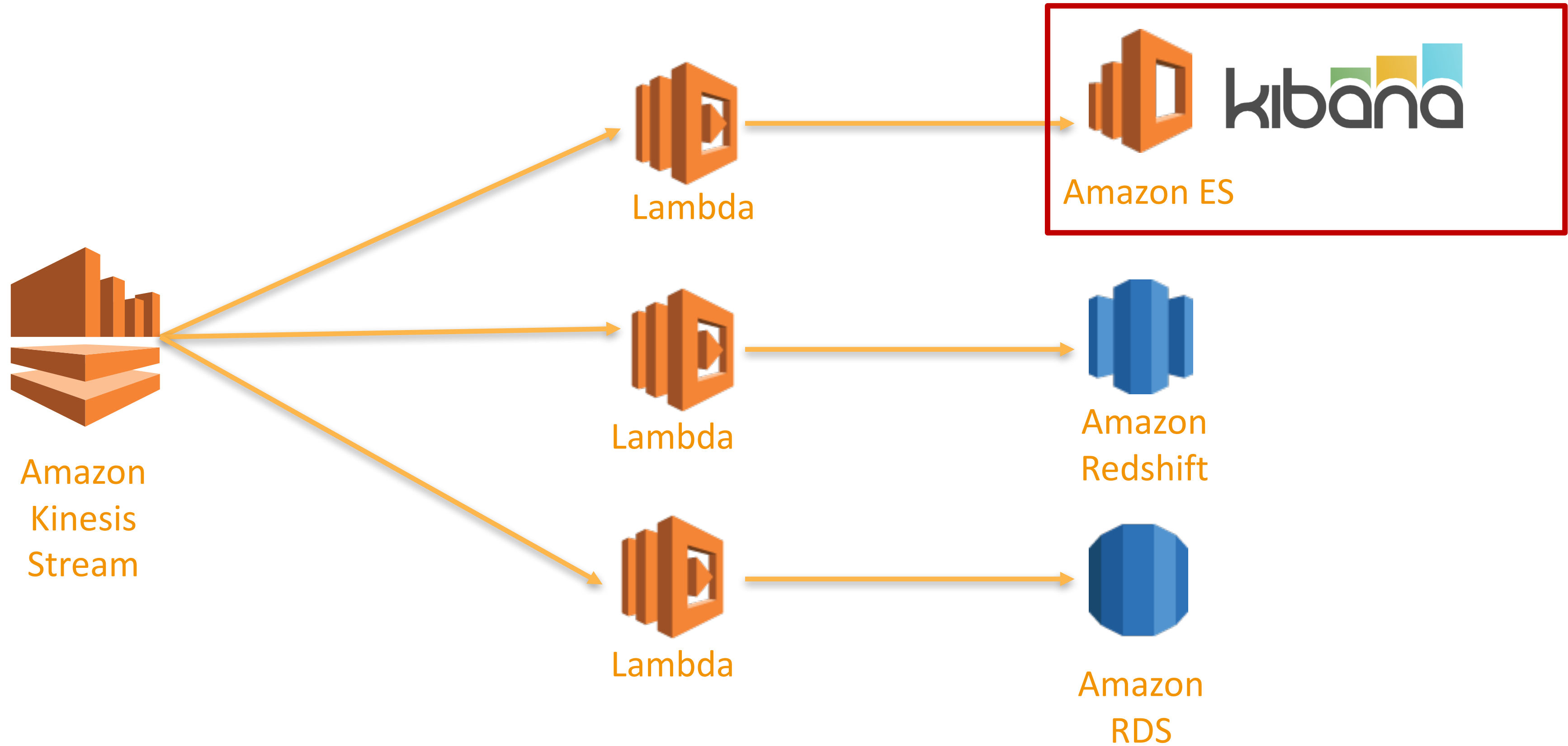


http://blogs.aws.amazon.com/bigdata/post/Tx36LYSCY2R0A9B/Implement-a-Real-time-Sliding-Window-Application-Using-Amazon-Kinesis-and-Apache

Lambda Application



Lambda Application



kibana | Discover | Visualize | Dashboard | Settings | 30 seconds | Last

6 US Presidential Race

realDonaldTrump,realBenCarson,JebBush,tedcruz,ChrisChristie,JohnKasich,GovMikeHuckabee,RandPaul,MarcoRubio,CarlyFiorina,JebBush,Hillary(

Sentiment

Legend
- Negative
- Neutral
- Positive
- Confused

US Map

Leaflet | Tiles by MapQuest — Map data © OpenStreetMap contributors,

r - top mentions

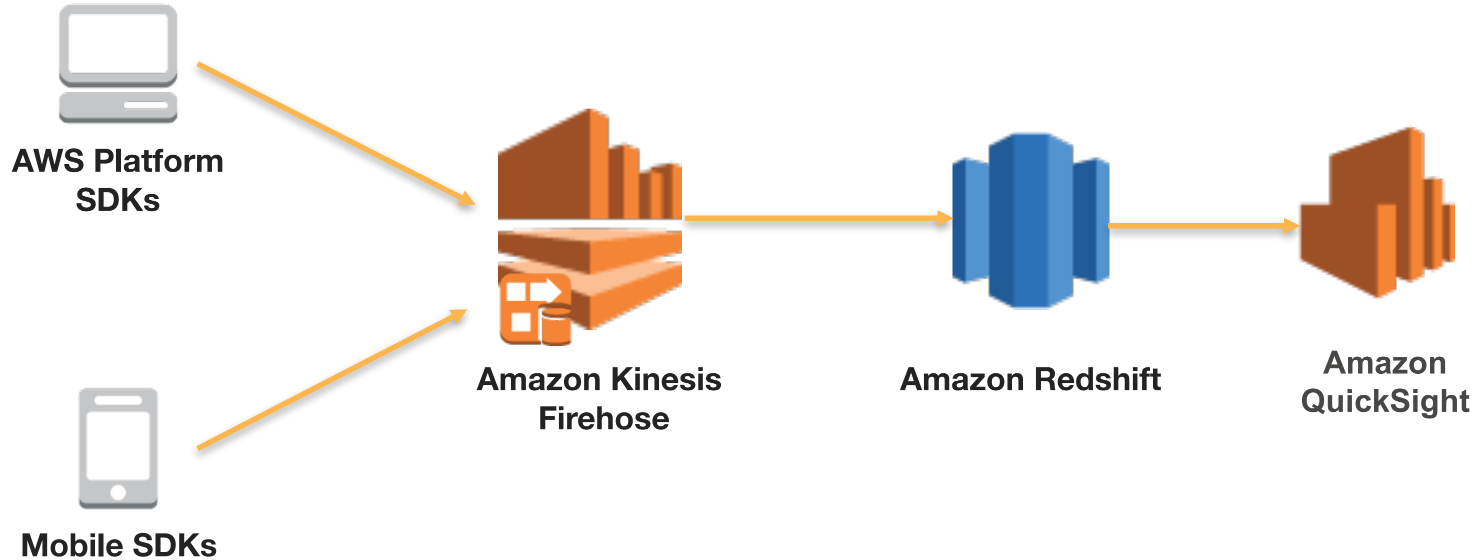
Legend
- Count

0 | 10 | 20 | 30 | 40

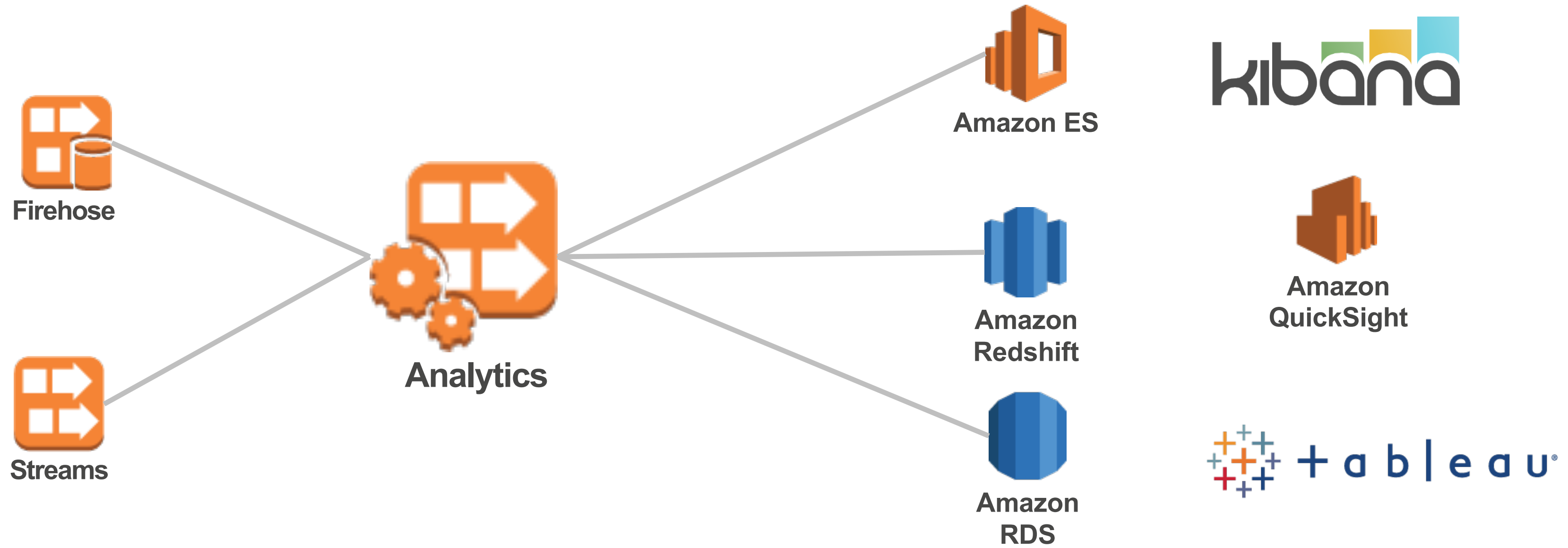
realdonaldtrump, realbencarson, berniesanders, marcorubio, jebbush, cnn, cnnpolitics, johnkasich, foxbusiness, hillaryclinton

Top 10 mentions

Kinesis Firehose + Redshift



Kinesis Analytics



Sushiro: Kaiten Sushi Restaurants

スシロー SUSHIRO

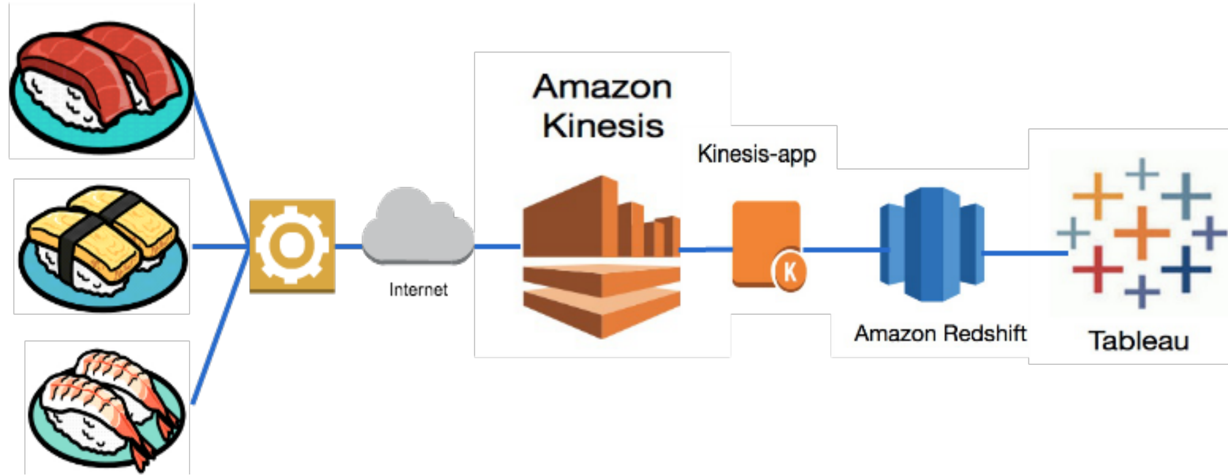
380 stores stream data from sushi plate sensors and stream to Kinesis

調理スペース
350m移動した
皿はごみ箱へ
ごみ箱
区切り皿
スシ
レーン
キッチン
客席
センサー
センサー
センサー
センサー
カウンター
カウンター
テーブル
オーダー
受付端末

Sushiro: Kaiten Sushi Restaurants

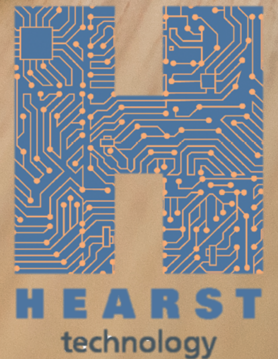
SUSHIRO

380 stores stream data from sushi plate sensors and stream to Kinesis



The Life of a Click

How Hearst Publishing Manages Clickstream Analytics



Final Hearst Data Pipeline

Users to Hearst Properties

Clickstream

Node.JS App- Proxy

Amazon Kinesis Streams

Spark

ETL on EMR

Firehose

S3

Agg Data

Models

Amazon Redshift

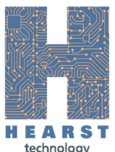
Data Science Application

API Ready Data

elastic

Buzzing API

LATENCY	Milliseconds	30 Seconds	100 Seconds	5 Seconds
THROUGHPUT	100 GB/Day	5 GB/Day	1 GB/Day	1 GB/Day



Demo

AWS Glue – Managed ETL Service

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

amazon web services

Why would AWS get into the ETL space?

We have lots of ETL partners

Amazon Redshift Partner Page for Data Integration

Fivetran

The problem is

70% of ETL jobs are hand-coded

With **no** use of ETL tools.

Why would anyone hand-code?

Brittle | Error-prone | Laborious

Code is flexible | Code is powerful

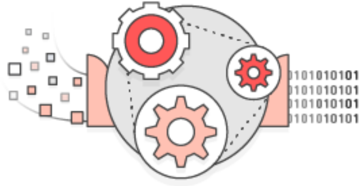
You can unit test | You can deploy with other code | You know your dev tools

You also need to maintain this code

- As data formats change
- As target schemas change
- As you add sources
- As data volume grows

ETL is the most time-consuming part of analytics

ETL



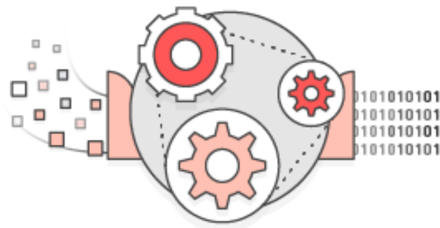
70% of time spent here

Data Warehousing

Amazon Redshift

Business Intelligence

Amazon QuickSight



Glue automates the undifferentiated heavy-lifting of ETL

- ✓ Cataloging data sources
- ✓ Identifying data formats and data types
- ✓ Handling errors
- ✓ Managing and scaling resources
- ✓ Generating Extract, Transform, Load code
- ✓ Executing ETL jobs; managing dependencies

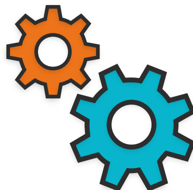
AWS Glue: components

Data Catalog

- Hive metastore compatible metadata repository of data sources.
- Crawls data source to infer table, data type, partition format.

Job Authoring

- Generates Python code to move data from source to destination.
- Edit with your favorite IDE; share code snippets using Git.



Job Execution

- Runs jobs in Spark containers – automatic scaling based on SLA.
- Glue is serverless - only pay for the resources you consume.

Glue data catalog

Discover and organize your data sets

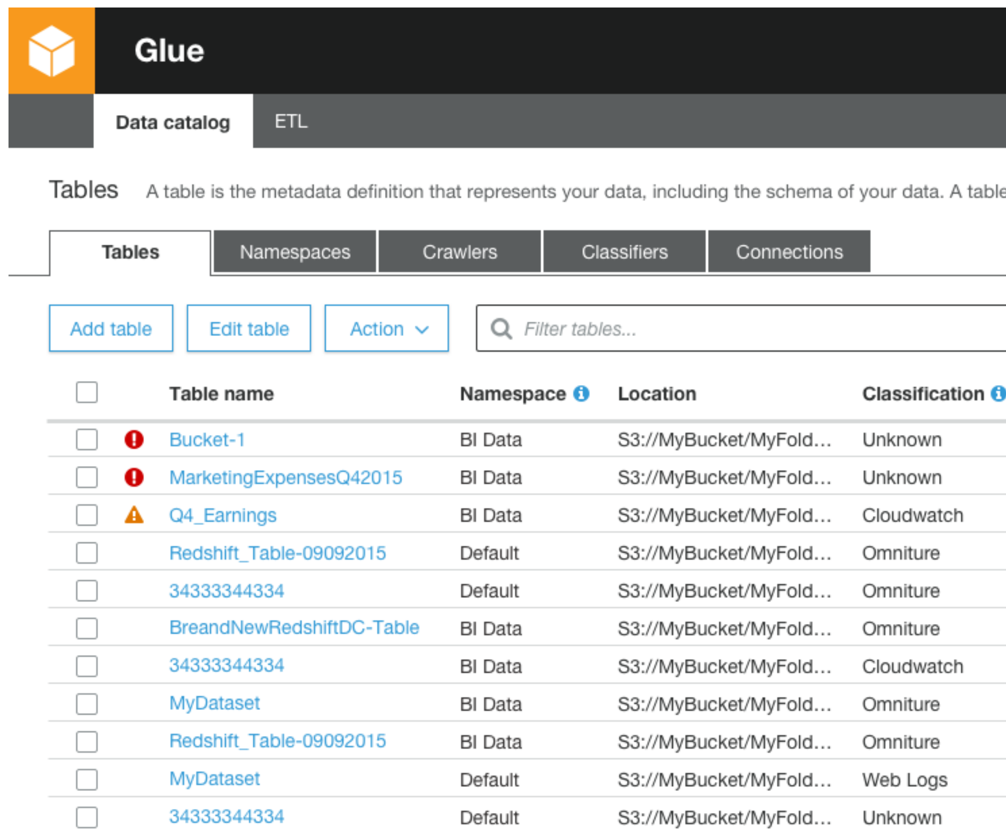
Glue data catalog

Manage table metadata through a Hive metastore API or Hive SQL. Supported by tools such as Hive, Presto, Spark, etc.

We added a few extensions:

- **Search** metadata for data discovery
- **Connection info** – JDBC URLs, credentials
- **Classification** for identifying and parsing files
- **Versioning** of table metadata as schemas evolve and other metadata are updated

Populate using Hive DDL, bulk import, or automatically through **crawlers**.



Crawlers: auto-populate data catalog

Automatic schema inference:

- Built-in **classifiers** detect file type and **extract schema:** record structure and data types.
- Add your own or share with others in the Glue community - It's all Grok and Python.

Auto-detects Hive-style partitions, grouping similar files into one table.

Run crawlers on schedule to **discover** new data and **schema changes.**

Serverless – only pay when crawls run.

Glue

Data catalog | ETL

Crawlers A crawler connects to a data store, progresses through a prioritized list of classifiers to determine

Tables | Namespaces | Crawlers | Classifiers | Connections

Add crawler | Run crawler | Action | Filter crawlers...

Name	Schedule	Status	Las
S3 Crawler	Every M, W, F at 3:00PM PST	Stopped	10 n
Redshift crawler	Every M at 12:00AM PST	44min elapsed	11 n
Adtech crawler	Every 15 minutes	Schedule paused	21 n
Sales crawler	Monthly on the 15th	Stopped	7 mi
Reporting crawler	Hourly	Scheduled	42 n
On-premisis crawler	0 0 12 * * ?	Stopped	33 n
My crawler	Every M, W, F at 3:00PM PST	Failed	11 n

Crawler history

Start time	Run duration	Median runtime	Tables update
11 Jan 2016, 12:00AM PST	10 min	12 min	2
4 Jan 2016, 12:00AM PST	11 min	12 min	2

Crawlers: automatic schema Inference

enumerate
S3 objects

file 1

file 2

. . .

file N

identify file type
and parse files

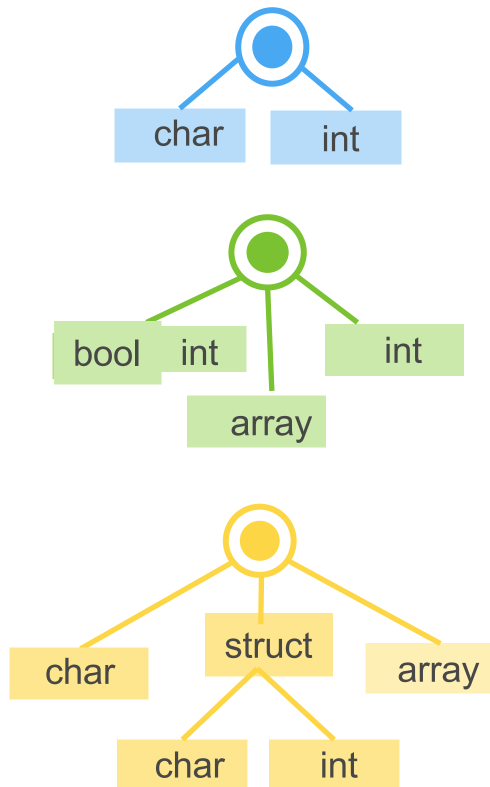
custom classifiers

app log parser
metrics parser
…

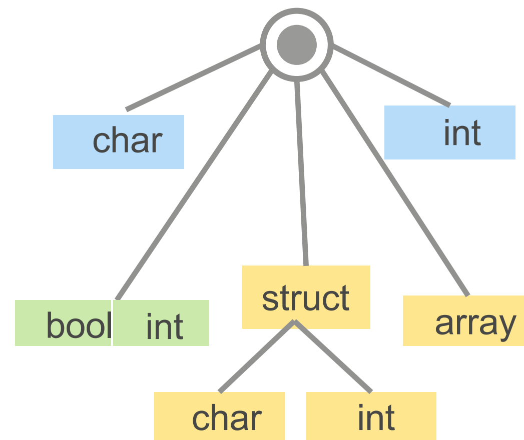
system classifiers

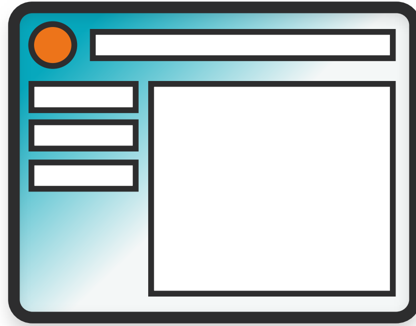
JSON parser
CSV parser
Apache log parser
…

semi-structured
per-file schema



semi-structured
unified schema

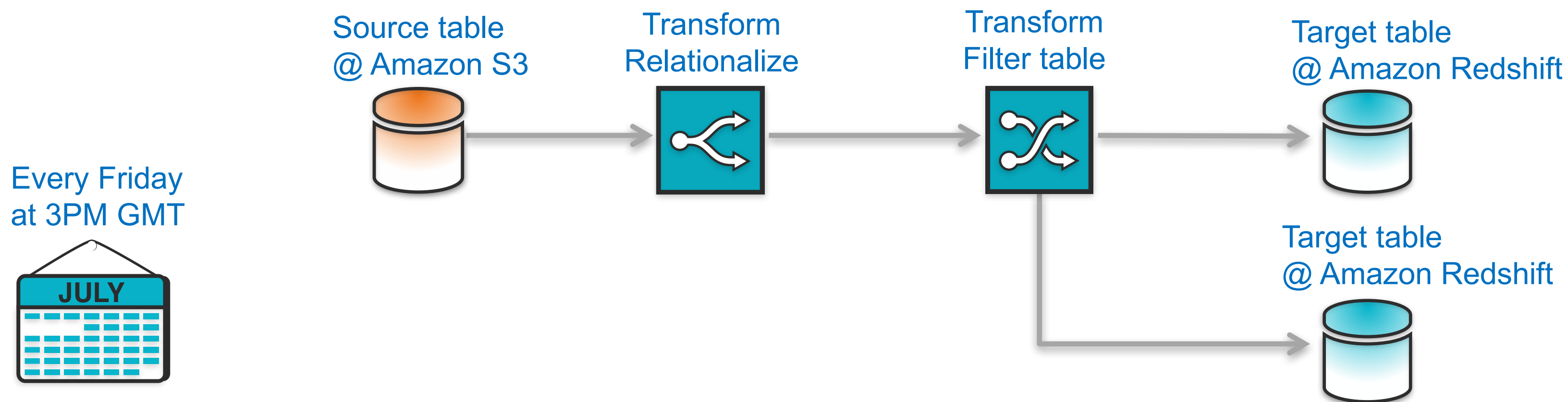




Job authoring in Glue

Make ETL job authoring like code development using your own tools

Automatic code generation



1. Pick sources and targets from the data catalog
2. Glue generates transformation graph and *Python code*
3. Specify trigger condition

Glue ETL scripts are forgiving and flexible

Edit script: MyScriptName

Run job | Save script | Action

Script: Version 0 (current)

Script | Arguments | Table | Logs | Statistics

SourceTable
Relationalize
Filter Tables
EventsTable
S3
Redshift
SessionsTable
Redshift

```
1  from pyspark.context import SparkContext
2  from glue.context import GlueContext
3  from glue.transforms import Relationalize
4  from glue.transforms import SelectFromCollection
5  from glue.transforms import DropNullFields
6
7  sc = SparkContext()
8  glueContext = GlueContext(sc)
9  ## @type: DataSource
10 ## @args: [name_space = "glue-demo-25", table_name =
      "simple_tweets_json_3c62f827"]
11 ## @return: datasource0
12 ## @inputs: []
13 datasource0 = glueContext.create_dynamic_frame.from_catalog(name_space = "glue
      -demo-25", table_name = "simple_tweets_json_3c62f827")
14 ## @type: Relationalize
15 ## @args: [name = "simple_tweets_json_3c62f827"]
16 ## @return: relationalize1
17 ## @inputs: [frame = datasource0]
18 relationalize1 = Relationalize.apply(frame = datasource0, name =
      "simple_tweets_json_3c62f827")
19 ## @type: SelectFromCollection
20 ## @args: [key = "simple_tweets_json_3c62f827_entities.hashtags"]
21 ## @return: selectfromcollection4
22 ## @inputs: [dfc = relationalize1]
23 selectfromcollection4 = SelectFromCollection.apply(dfc = relationalize1, key =
      "simple_tweets_json_3c62f827_entities.hashtags")
24 ## @type: DropNullFields
25 ## @args: []
```

- **Human-readable** code run on a scalable platform, PySpark
- **Forgiving** in the face of failures – handles bad data and crashes
- **Flexible**: handles complex semi-structured data, and adapts to source schema changes

Glue ETL scripts are forgiving

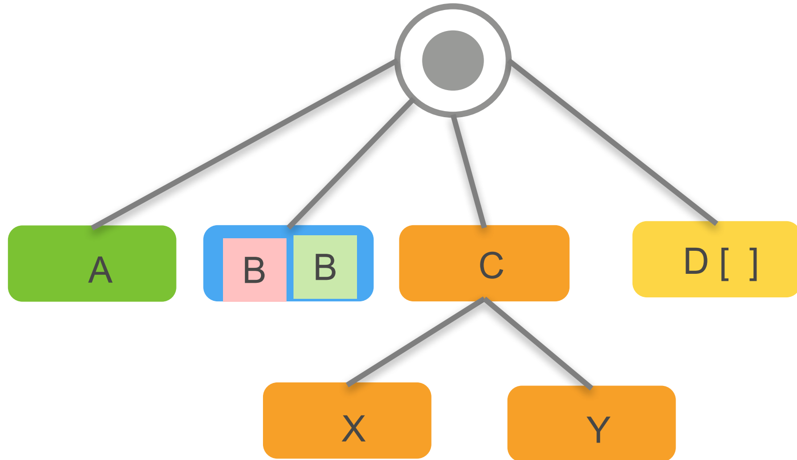
```
## @type: DataSource
## @args: [name_space = "glue", table_name = "tweets"]
## @return: datasource0
## @inputs: []
datasource = glueContext.create_dynamic_frame.from_catalog(name_space =
    "glue", table_name = "tweets")

## @type: Relationize
## @args: [name = "tweets"]
## @return: relations
## @inputs: [frame = datasource]
relations = Relationize.apply(frame = datasource, name = "tweets")
```

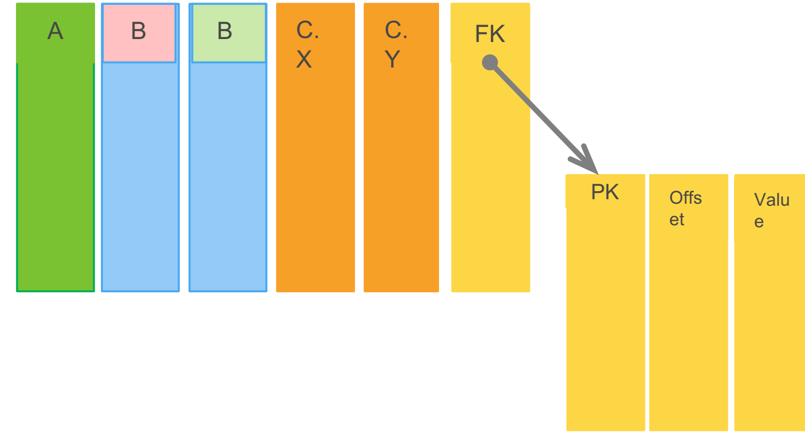
Generated code has human-readable annotations corresponding to the ETL graph

Forgiving in the face of failures:

- **Idempotency** is built-in. After a crash, a job restart picks up from where it left off, and output data is not duplicated.
- **Bad data** is flagged in-situ and does not crash the job. Users can siphon error tuples at any step to S3 buckets for subsequent error processing.

Glue transformations are flexible and adaptive





- Flatten semi-structured objects with arbitrary complexity into relational tables, on-the-fly.
- Pivot arrays and other collection types into a separate table, generating key-foreign key values.
- Modify mapping as the source schema changes, and modify the target schemas as needed.

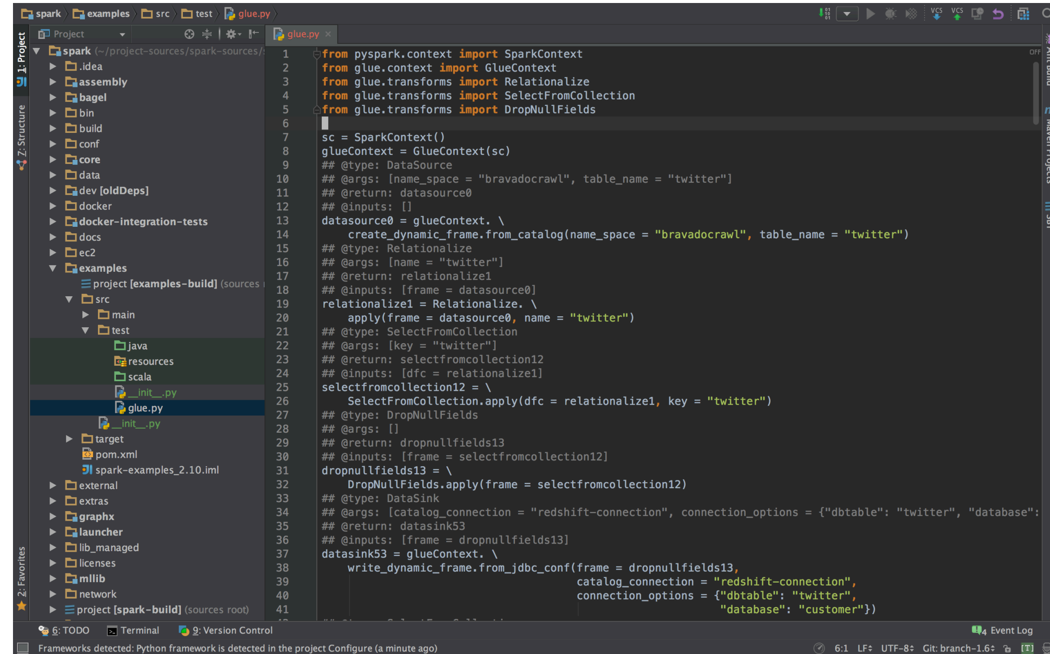
Git integration: development your way

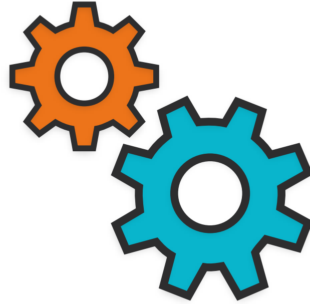
Glue integrates job authoring and execution with your preferred Git services.

Push job code to your Git repository, automatically **pulls** the latest on job invocation.

Customize ETL jobs in your favorite IDE – no need to learn new tools

Track and version your jobs to match your development, test, and release process





Orchestration & resource management

Fully managed, serverless job execution

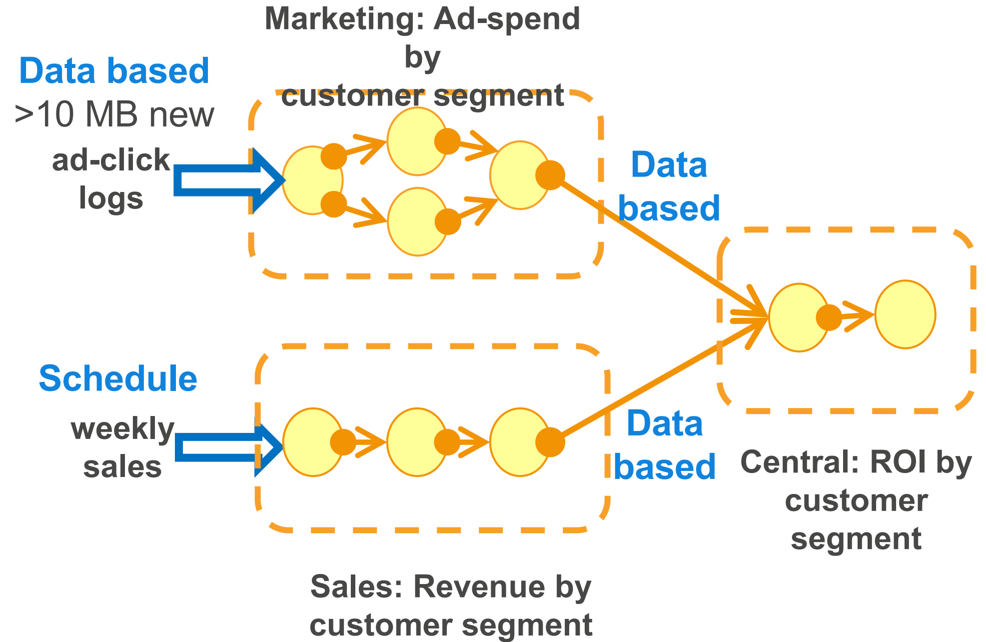
Job composition and triggers

Compose jobs globally with event-based dependencies

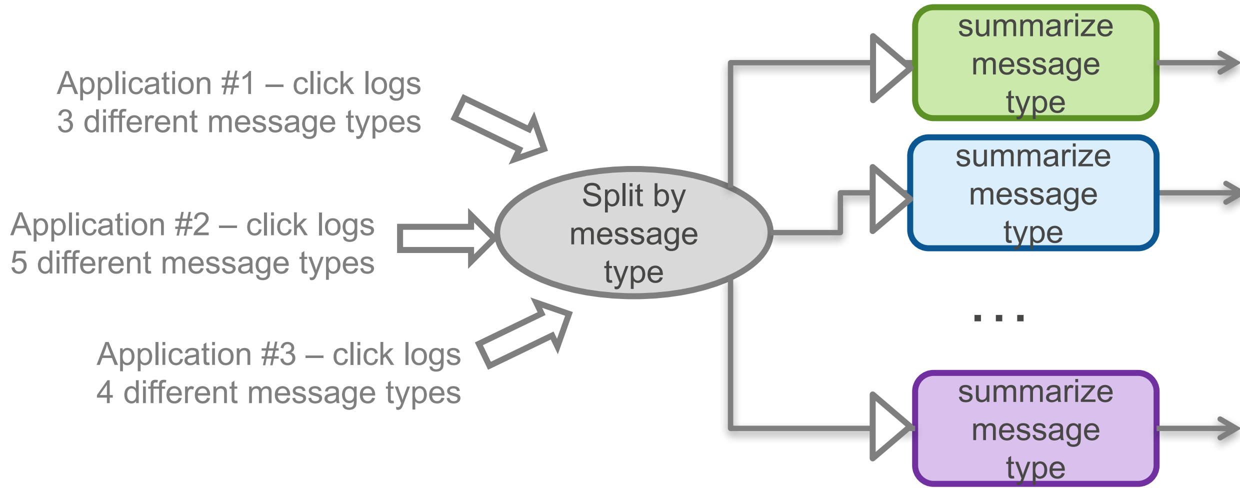
- Easy to reuse and leverage work across organization boundaries

Multiple triggering mechanisms

- **Schedule-based:** e.g., time of day
- **Event-based:** e.g., data availability, job completion
- **External sources:** e.g., AWS Lambda



Dynamic orchestration



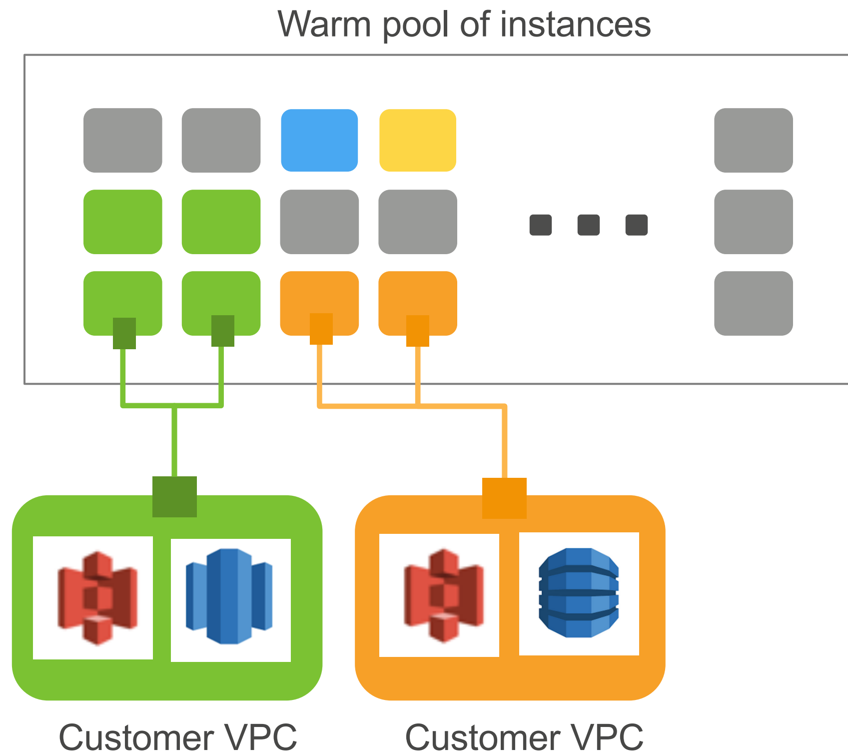
Example: Dynamic number of jobs based on application type and number of message types

- Add jobs dynamically as graph unfolds - makes data dependent orchestration possible
- Glue provides fault-tolerant orchestration - retries on job failure
- Monitoring and metrics - job run history and event tracking for debugging

Serverless job execution

There is no need to provision, configure, or manage servers

- Warm pools: pre-configured fleets of instances to reduce job startup time
- Auto-configure VPC and role-based access
- Automatically scale resources to meet SLA and cost objectives
- You pay only for the resources you consume while consuming them.



Monitoring, metrics, and notification

- Track pipelines, job history
- Drill-down to transform level metrics
  - rows processed
  - input & output schema
- Notifications on: job status, errors, new data, and schema changes

Data catalog ETL Settings Tutorials 7

Jobs A job is your business logic required to perform extract, transform and load (ETL) work. Job runs are initiated by triggers which can be scheduled or driven by events.

Jobs Repositories

Job performance

Add job Edit job Action Filter jobs... Save view Showing 1-20 of 999

Drag a rectangle over a chart to choose jobs you would like to view.

Median run time: 25 mins

0 MINS RUN TIME PER JOB RUN ATTEMPT 1 HR

48 HOURS JOB RUN ATTEMPT HISTORY 0

Median run frequency: 5 Days

10 MINS JOB RUN ATTEMPT FREQUENCY PER JOB 30 DAYS

Legend
Job run
Selection
Job error

Jobs	Trigger status	Most recent Job status	Rows read	Rows written	Errors	Duration	Start time
Unsaved job	Active	Running	1,011	3,456	3,456 (77%)	11 min	12 Nov 2015 2:01 PM PST
Aardvark	Inactive	Completed	9,012	9,012	9,012 (50%)	23 min	12 Nov 2015 2:01 PM PST
Bobcat	Active	Failed	9,012	9,012	9,012 (50%)	1 hr 56 min	12 Nov 2015 2:01 PM PST
Crow	Active	Running	1,234	7,890	7,890 (84%)	55 min	12 Nov 2015 2:01 PM PST
Deer	No trigger	Running	7,890	9,012	9,012 (53%)	13 min	12 Nov 2015 2:01 PM PST
Elephant	Active	Completed	9,012	7,890	7,890 (47%)	13 min	12 Nov 2015 2:01 PM PST

Details Diagram Code History

Name: Aardvark
Description: Lorem ipsum dolor sit amet ...
Version: 32
Tags: Env **Production**
Project **Big Data**

Run successes: 321
Run failures: 27

90% Success

Amazon Athena

© 2015, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

amazon web services

Challenges Customers Faced

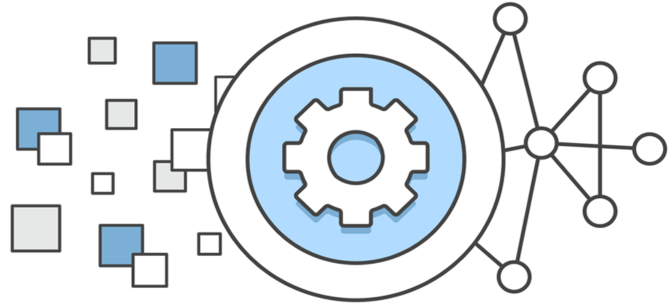
- Significant amount of work required to analyze data in Amazon S3
- Users often only have access to aggregated data sets
- Managing a Hadoop cluster or data warehouse requires expertise

Introducing Amazon Athena

Amazon Athena is an interactive query service that makes it easy to analyze data directly from Amazon S3 using Standard SQL

Athena is Serverless

- No Infrastructure or administration
- Zero Spin up time
- Transparent upgrades



Amazon Athena is Easy To Use

- Log into the Console
- Create a table
    - Type in a Hive DDL Statement
    - Use the console Add Table wizard
- Start querying

Amazon Athena is Highly Available

- You connect to a service endpoint or log into the console
- Athena uses warm compute pools across multiple Availability Zones
- Your data is in Amazon S3, which is also highly available and designed for 99.999999999% durability

Query Data Directly from Amazon S3

- No loading of data
- Query data in its raw format
    - Text, CSV, JSON, weblogs, AWS service logs
    - Convert to an optimized form like ORC or Parquet for the best performance and lowest cost
- No ETL required
- Stream data from directly from Amazon S3
- Take advantage of Amazon S3 durability and availability

Use ANSI SQL

- Start writing ANSI SQL
- Support for complex joins, nested queries & window functions
- Support for complex data types (arrays, structs)
- Support for partitioning of data by any key
  - (date, time, custom keys)
  - e.g., Year, Month, Day, Hour or Customer Key, Date

```sql
WITH q21_tmp1_cached AS
 (SELECT l_orderkey,
         count(DISTINCT l_suppkey) AS count_suppkey,
         max(l_suppkey) AS max_suppkey
  FROM lineitem_parq
  WHERE l_orderkey IS NOT NULL
  GROUP BY l_orderkey),
     q21_tmp2_cached AS
 (SELECT l_orderkey,
         count(DISTINCT l_suppkey) count_suppkey,
                                   max(l_suppkey) AS max_suppkey
  FROM lineitem_parq
  WHERE l_receiptdate > l_commitdate
    AND l_orderkey IS NOT NULL
  GROUP BY l_orderkey)
SELECT s_name,
       count(1) AS numwait
FROM
 (SELECT s_name
  FROM
   (SELECT s_name,
           t2.l_orderkey,
           l_suppkey,
           count_suppkey,
           max_suppkey
    FROM q21_tmp2_cached t2
    RIGHT OUTER JOIN
     (SELECT s_name,
             l_orderkey,
             l_suppkey
      FROM
       (SELECT s_name,
               t1.l_orderkey,
               l_suppkey,
               count_suppkey,
               max_suppkey
        FROM q21_tmp1_cached t1
        JOIN
         (SELECT s_name,
                 l_orderkey,
                 l_suppkey
          FROM orders_parq o
          JOIN
           (SELECT s_name,
                   l_orderkey,
                   l_suppkey
            FROM nation_parq n
            JOIN supplier s ON s.s_nationkey = n.n_nationkey
            AND n.n_name = 'SAUDI ARABIA'
            JOIN lineitem_parq l ON s.s_suppkey = l.l_suppkey
            WHERE l.l_receiptdate > l.l_commitdate
              AND l.l_orderkey IS NOT NULL) l1 ON o.o_orderkey = l1.l_orderkey
          AND o.o_orderstatus = 'F') l2 ON l2.l_orderkey = t1.l_orderkey) a
      WHERE (count_suppkey > 1)
        OR ((count_suppkey=1)
          AND (l_suppkey <> max_suppkey))) l3 ON l3.l_orderkey = t2.l_orderkey) b
  WHERE (count_suppkey IS NULL)
    OR ((count_suppkey=1)
      AND (l_suppkey = max_suppkey))) c
GROUP BY s_name
ORDER BY numwait DESC,
         s_name LIMIT 100;
```

Familiar Technologies Under the Covers

Used for SQL Queries

In-memory distributed query engine

ANSI-SQL compatible with extensions

Used for DDL functionality

Complex data types

Multitude of formats

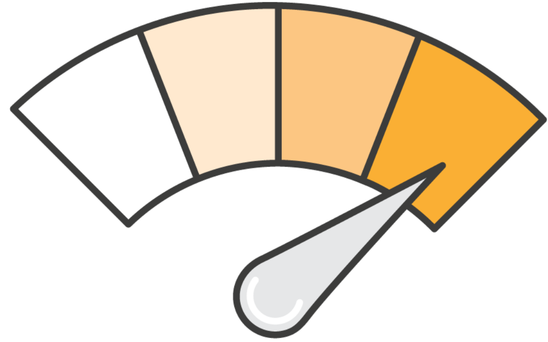
Supports data partitioning

Amazon Athena Supports Multiple Data Formats

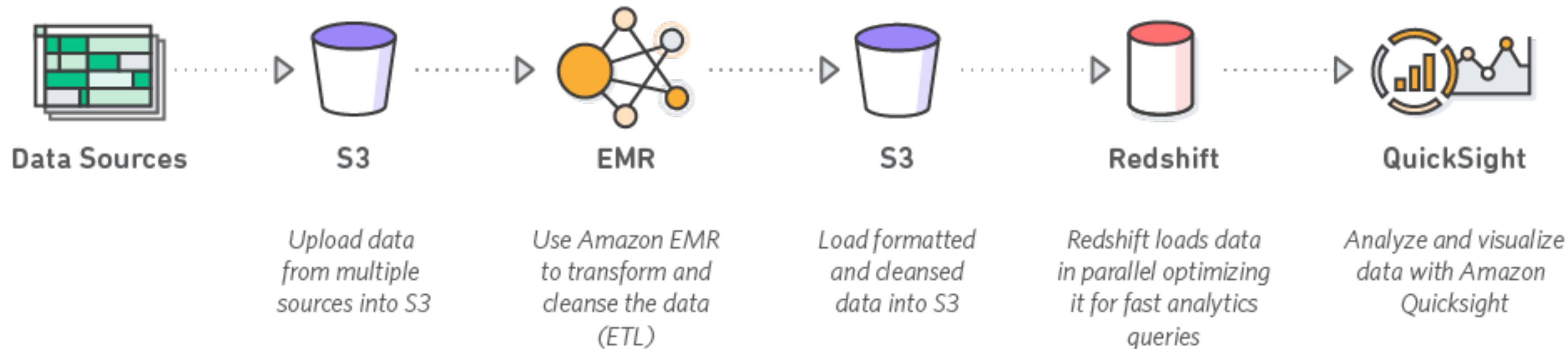
- Text files, e.g., CSV, raw logs
- Apache Web Logs, TSV files
- JSON (simple, nested)
- Compressed files (Snappy, Zlib, GZIP, LZO)
- Columnar formats such as Apache Parquet & Apache ORC
- AVRO support
- Encryption

Amazon Athena is Fast

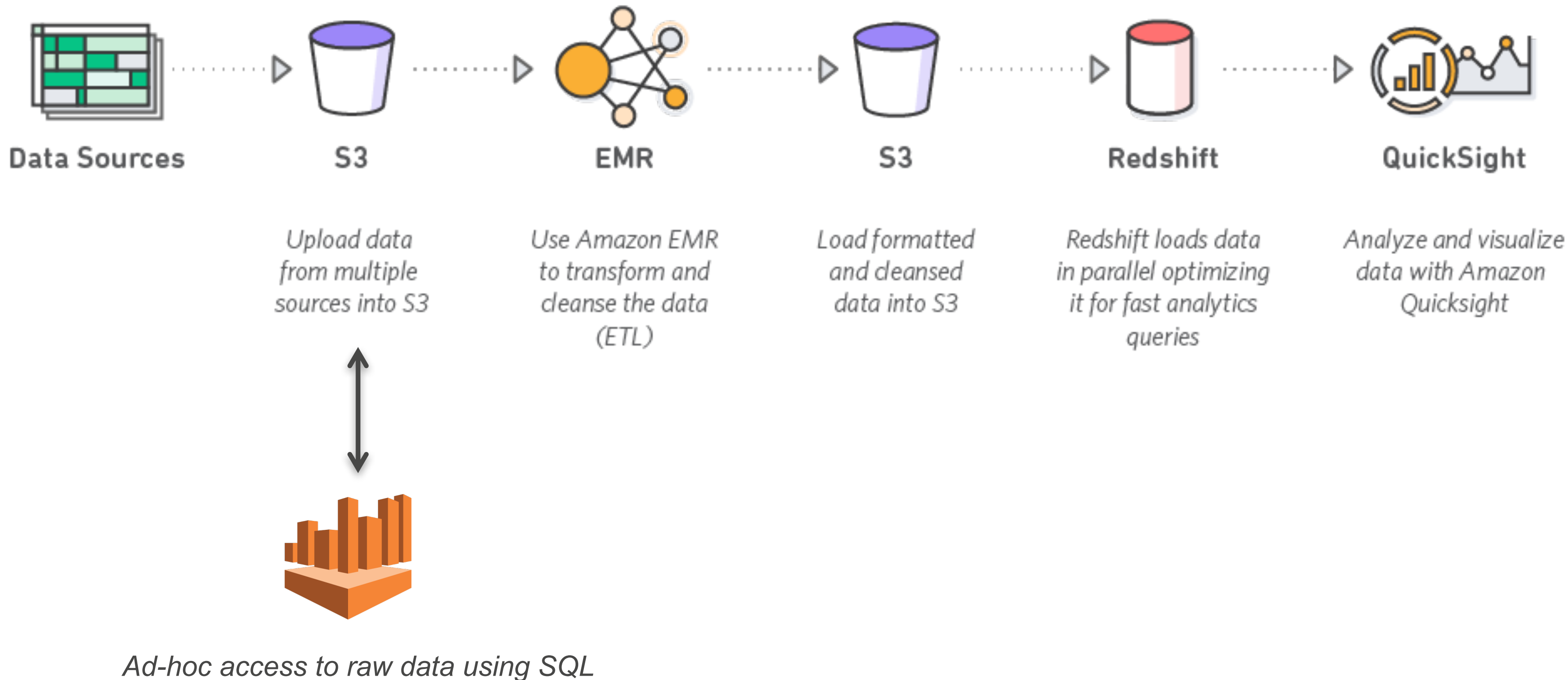
- Tuned for performance
- Automatically parallelizes queries
- Results are streamed to console
- Results also stored in S3
- Improve Query performance
  - Compress your data
  - Use columnar formats



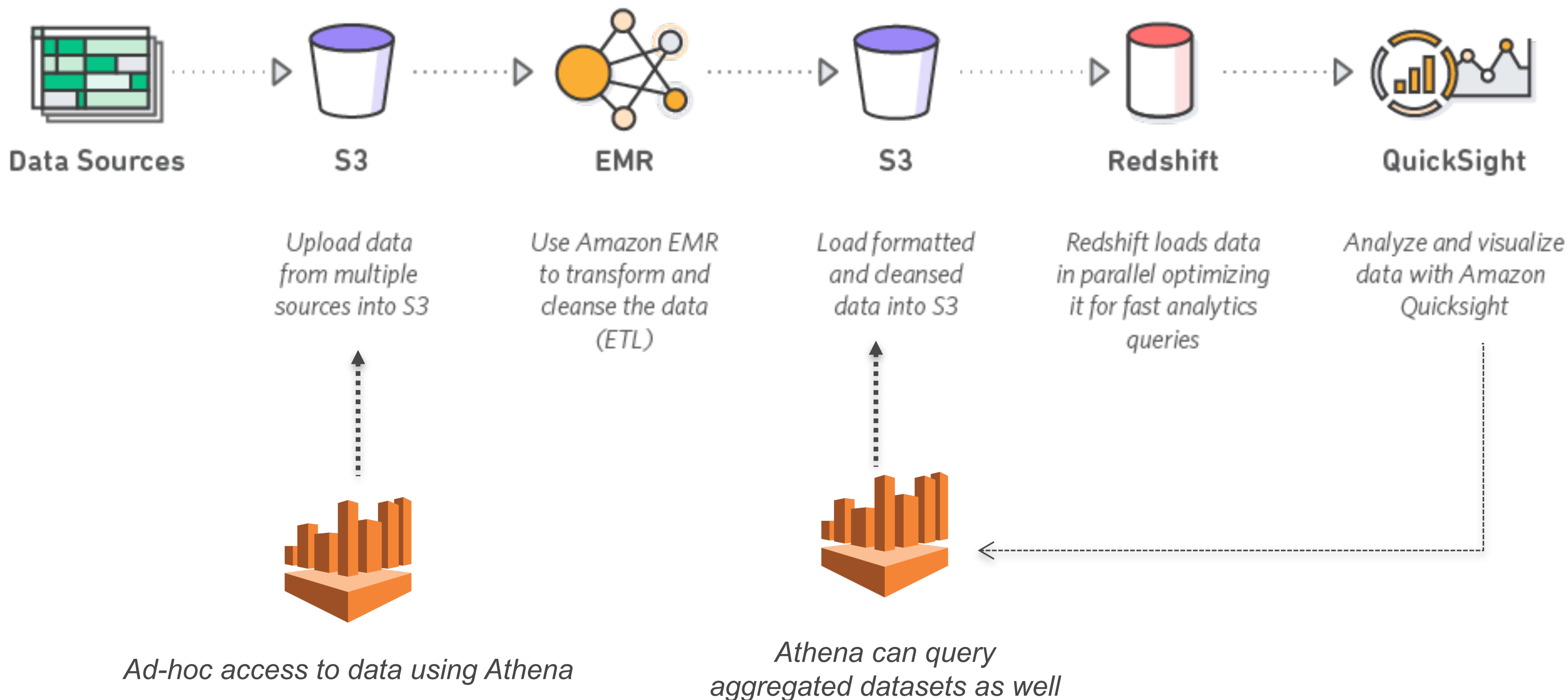
A Sample Pipeline



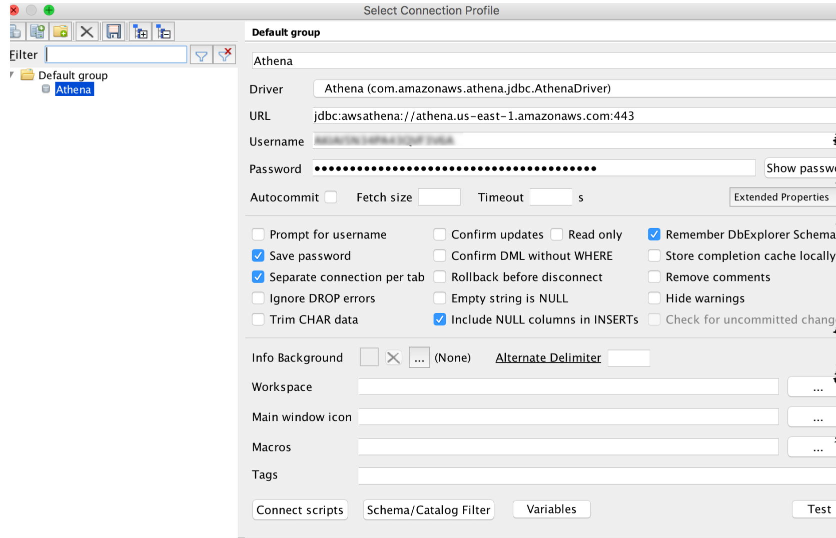
A Sample Pipeline



A Sample Pipeline



Use the JDBC Driver



Columns | SQL source | Data | Indexes | References | Refere

COLUMN_NAME Table: "AwsDataCatalog"."default".clou

COLUMN_NAME	DATA_TYPE	PK	NULLABLE	DEFAULT	AUTOINCREMENT	COMPUTED	REMARKS	JDBC Type
date	date	NO	NO		NO	NO		DATE
time	string	NO	NO		NO	NO		LONGNVARCHAR
location	string	NO	NO		NO	NO		LONGNVARCHAR
bytes	int	NO	NO		NO	NO		INTEGER
requestip	string	NO	NO		NO	NO		LONGNVARCHAR
method	string	NO	NO		NO	NO		LONGNVARCHAR
host	string	NO	NO		NO	NO		LONGNVARCHAR
uri	string	NO	NO		NO	NO		LONGNVARCHAR
status	int	NO	NO		NO	NO		INTEGER
referrer	string	NO	NO		NO	NO		LONGNVARCHAR
os	string	NO	NO		NO	NO		LONGNVARCHAR

JDBC also Provides Programmatic Access

```
/* Setup the driver */
Properties info = new Properties();
info.put("user", "AWSAccessKey");
info.put("password", "AWSSecretAccessKey");
info.put("s3_staging_dir", "s3://S3 Bucket Location/");

Class.forName("com.amazonaws.athena.jdbc.AthenaDriver");

Connection connection =
DriverManager.getConnection("jdbc:awsathena://athena.us-east-
1.amazonaws.com:443/", info);
```

Creating a Table and Executing a Query

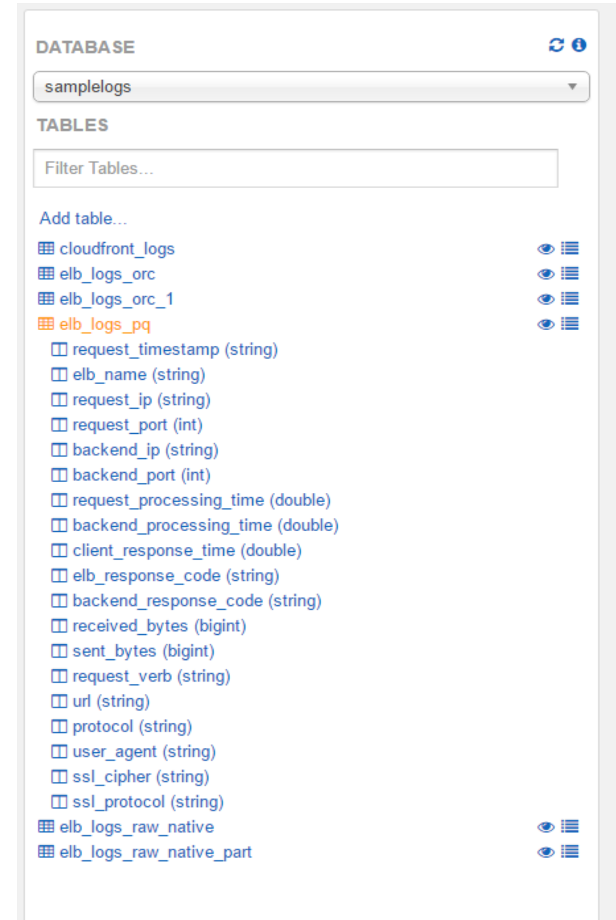
```
/* Create a table */
Statement statement = connection.createStatement();
ResultSet queryResults = statement.executeQuery("CREATE EXTERNAL TABLE
tableName ( Col1 String ) LOCATION ‘s3://bucket/tableLocation");

/* Execute a Query */
Statement statement = connection.createStatement();
ResultSet queryResults = statement.executeQuery("SELECT * FROM
cloudfront_logs");
```

Creating Tables - Concepts

- Create Table Statements (or DDL) are written in Hive
    - High degree of flexibility
    - Schema on Read
    - Hive is SQL like but allows other concepts such “external tables” and partitioning of data
    - Data formats supported – JSON, TXT, CSV, TSV, Parquet and ORC (via Serdes)
    - Data in stored in Amazon S3
    - Metadata is stored in an a metadata store

Athena's Internal Metadata Store

- Stores Metadata
  - Table definition, column names, partitions
- Highly available and durable
- Requires no management
- Access via DDL statements
- Similar to a Hive Metastore



Running Queries is Simple

```
1  SELECT o_year,
2           sum(case
3      WHEN nation = 'PERU' THEN
4      volume
5      ELSE 0 end) / sum(volume) AS mkt_share
6  FROM
7      (SELECT substr(o_orderdate,
8          1,
9          4) AS o_year,
10          l_extendedprice * (1 - l_discount) AS volume,
11          n2.n_name AS nation
12     FROM lineitem_parq, orders_parq, part_parq, customer_parq, supplier_parq, nation_parq n1, nation_parq n2, region_parq
13     WHERE p_partkey = l_partkey
14             AND s_suppkey = l_suppkey
15             AND l_orderkey = o_orderkey
16             AND o_custkey = c_custkey
17             AND c_nationkey = n1.n_nationkey
18             AND n1.n_regionkey = r_regionkey
19             AND r_name = 'AMERICA'
20             AND s_nationkey = n2.n_nationkey
21             AND o_orderdate
22         BETWEEN '1995-01-01'
```

Use Ctrl + Enter to run query, Ctrl + Space to autocomplete

Run Query | Save As | Format Query | New Query (Run time: 17.34 seconds, Data scanned: 12.97GB)

Run time and data scanned

Results

	o_year	mkt_share
1	1995	0.04048235539866028
2	1996	0.03984664161294089

Amazon Athena is Cost Effective

- Pay per query
- $5 per TB scanned from S3
- DDL Queries and failed queries are free
- Save by using compression, columnar formats, partitions

Converting to ORC and PARQUET

- You can use Hive CTAS to convert data
    - CREATE TABLE new_key_value_store
    - STORED AS PARQUET
    - AS
    - SELECT col_1, col2, col3 FROM noncolumartable
    - SORT BY new_key, key_value_pair;

- You can also use Spark to convert the file into PARQUET / ORC

- 20 lines of Pyspark code, running on EMR
    - Converts 1TB of text data into 130 GB of Parquet with snappy conversion
    - Total cost $5

https://github.com/awslabs/aws-big-data-blog/tree/master/aws-blog-spark-parquet-conversion

Pay By the Query - $5/TB Scanned

- Pay by the amount of data scanned per query

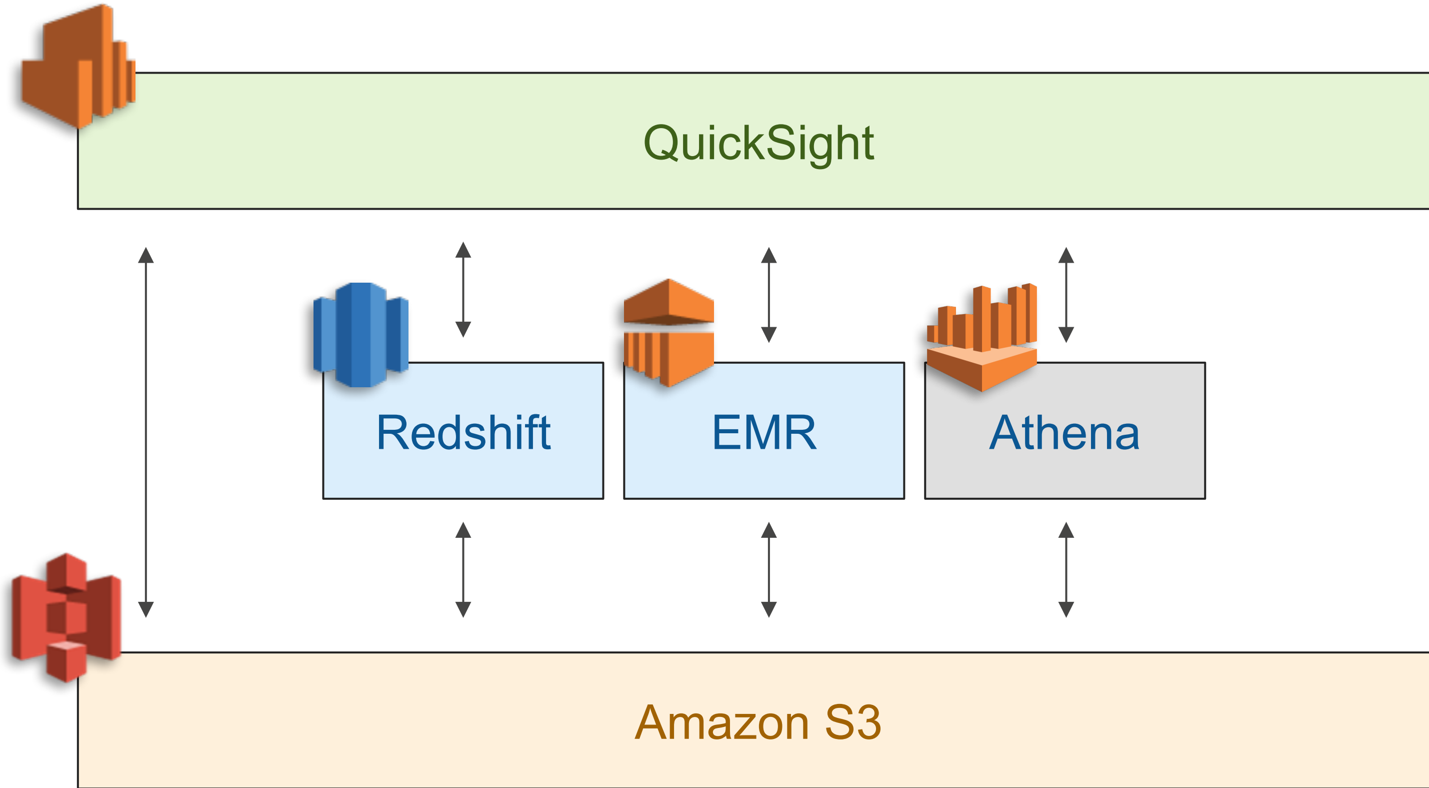
- Ways to save costs
  - Compress
  - Convert to Columnar format
  - Use partitioning

- Free: DDL Queries, Failed Queries

```
SELECT elb_name,
       uptime,
       downtime,
       cast(downtime as DOUBLE)/cast(uptime as DOUBLE) uptime_downtime_ratio
FROM
    (SELECT elb_name,
        sum(case elb_response_code
        WHEN '200' THEN
        1
        ELSE 0 end) AS uptime, sum(case elb_response_code
        WHEN '404' THEN
        1
        ELSE 0 end) AS downtime
    FROM elb_logs_raw_native
    GROUP BY  elb_name)
```

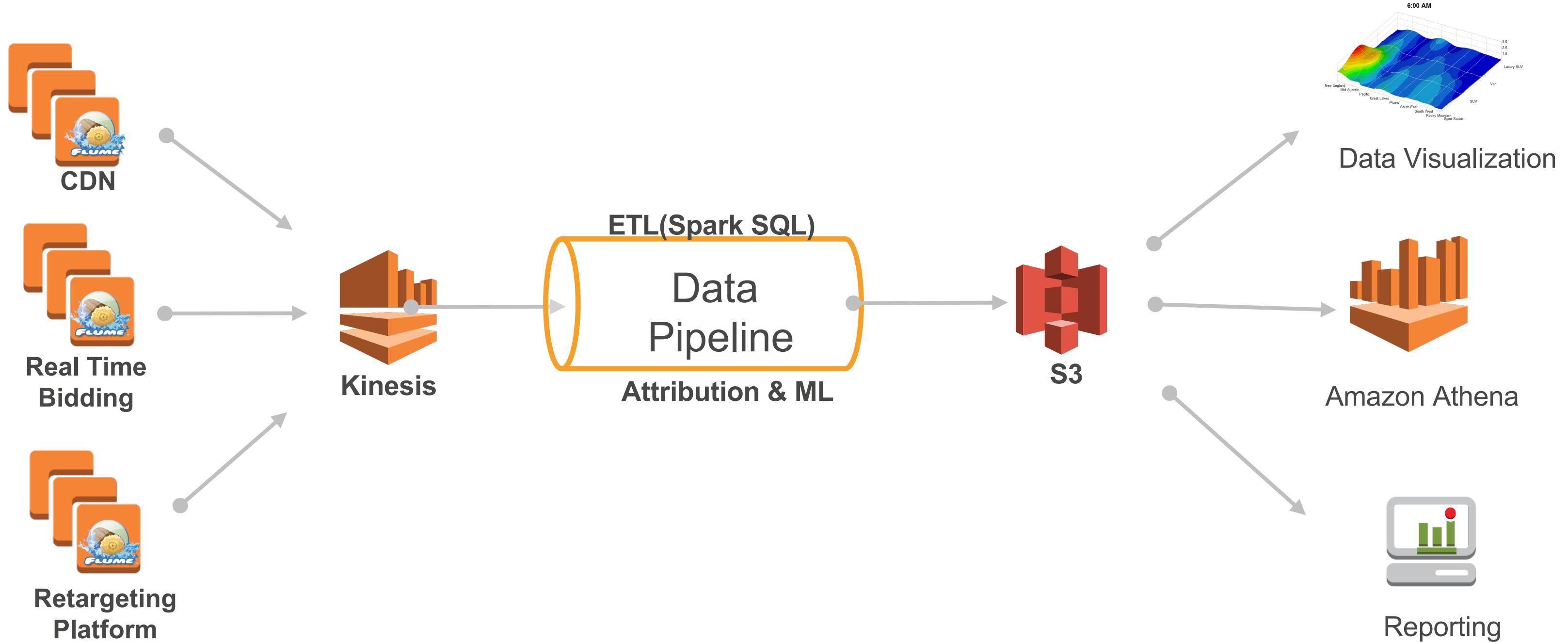
Dataset	Size on Amazon S3	Query Run time	Data Scanned	Cost
Logs stored as Text files	1 TB	237 seconds	1.15TB	$5.75
Logs stored in Apache Parquet format*	130 GB	5.13 seconds	2.69 GB	$0.013
Savings	**87% less with Parquet**	**34x faster**	**99% less data scanned**	**99.7% cheaper**

Athena Complements Amazon Redshift & Amazon EMR



Customers Using Athena

DataXu – 180TB of Log Data per Day



Amazon QuickSight

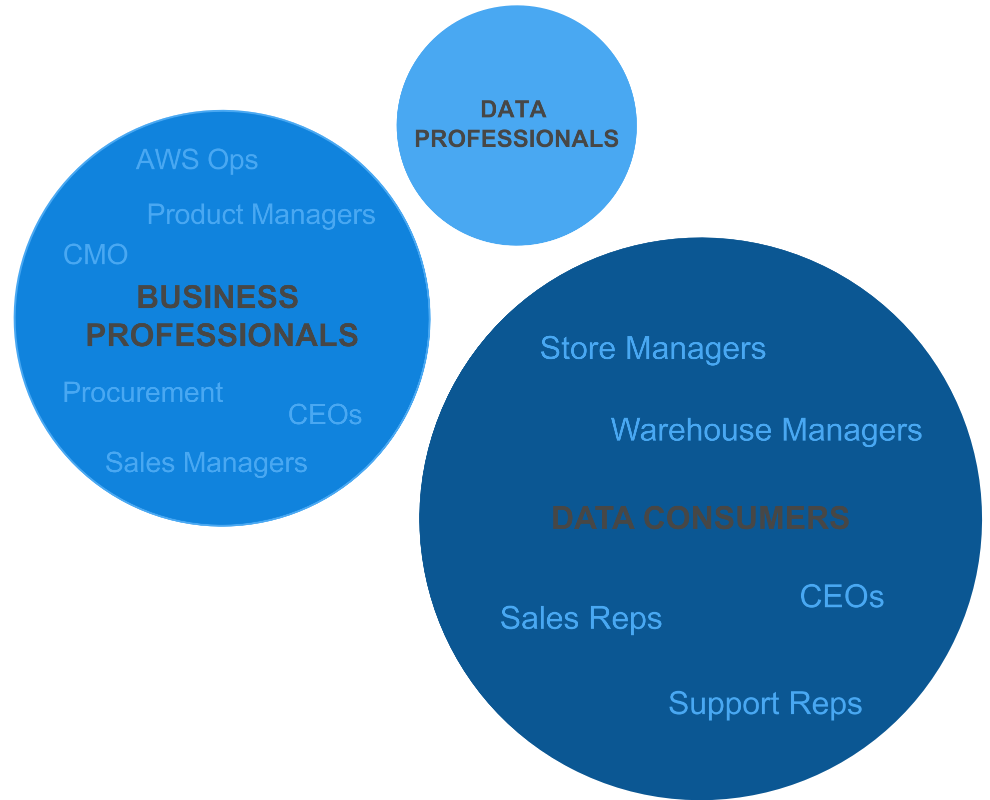
Amazon QuickSight is a Business Analytics Service that lets business users quickly and easily visualize, explore, and share insights from their data.

QuickSight

Who Is QuickSight For?

QuickSight is designed to give everyday business users the **flexibility** to do easy, self-serve analysis on their data.

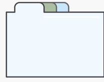
QuickSight is also **perfect** for delivering published dashboards throughout the organization.



Basic Concepts

DATA SOURCES

Relational Databases



Flat Files

Amazon Redshift

Amazon S3

DATA SETS

Retail Data

Ops Data

Marketing Data

ANALYSES

DASHBOARDS & STORIES

Deep Integration with AWS Data Sources

QuickSight is **deeply integrated** with AWS data sources like Redshift, RDS, S3, Athena and others, as well as third-party sources like Excel, Salesforce, as well as on-premises databases.

Super-fast Performance with SPICE

QuickSight is powered by SPICE, a super-fast calculation engine that delivers unprecedented performance and scale, delivering insights at the speed of thought.

Super-fast, **P**arallel, **I**n-memory optimized, **C**alculation **E**ngine

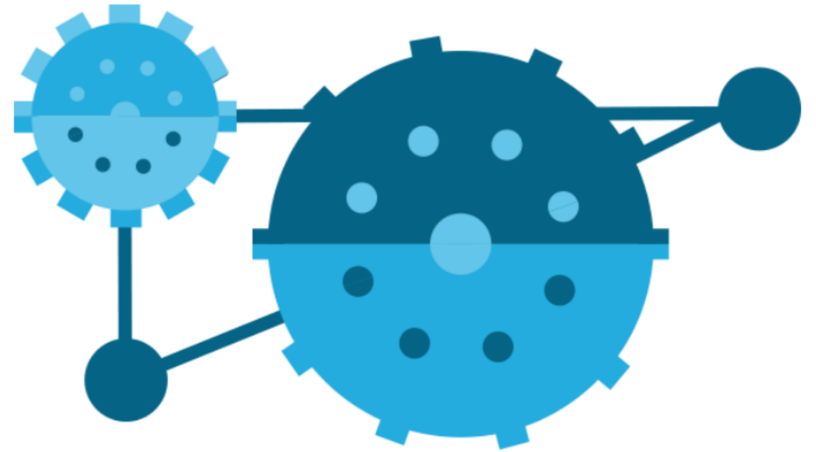
Compiled queries with machine code generation

Rich calculations

Rapid response time to queries

Fully managed—no hardware or software to license

SPICE



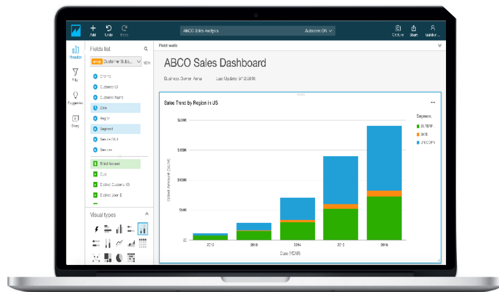
Data Prep

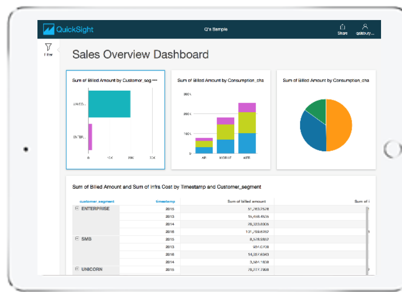
Optional step when creating data sets.

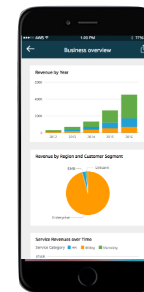
- Preview data
- Rename, remove fields, change data types
- Create new calculated fields
- Filter rows
- Issue direct query or ingest to SPICE
- Visually join tables from the same relational DB
- Push down custom SQL queries

Collaborate, Share, and Publish

QuickSight lets users create and share data sets, collaborate on your live analyses, and share read-only dashboards and storyboards that can be accessed on any device, anytime, anywhere.







Analyses

Analyses are visual explorations of your data. Multiple users can collaborate on an analyses with the ability to modify and change them in any way.

Dashboards

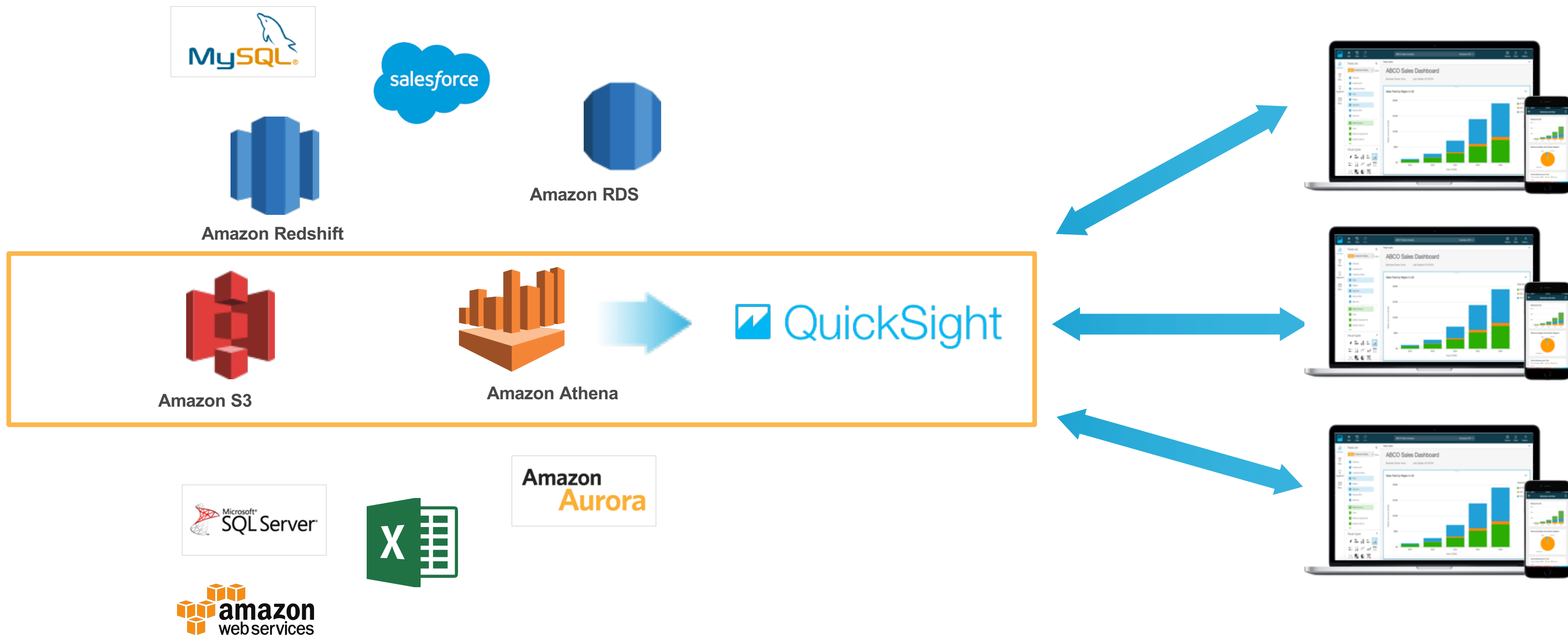
You can share your analyses as read only dashboards. Viewers can interact with and filter the visualizations without modifying them.

Storyboards

Let you combine visualizations into a guided tour that you can share with other users.

Using Amazon Athena with Amazon QuickSight

QuickSight allows you to connect to data from a wide variety of AWS, third-party, and on-premises sources including Amazon Athena



Demo

Database State of the Union

Database Services Usage

- Amazon Aurora is the fastest growing service in AWS history

- More than 14,000 databases have been migrated using AWS Database Migration Service

- DynamoDB served over 56 billion extra requests worldwide on Prime Day compared to the same day the previous week.

Amazon RDS

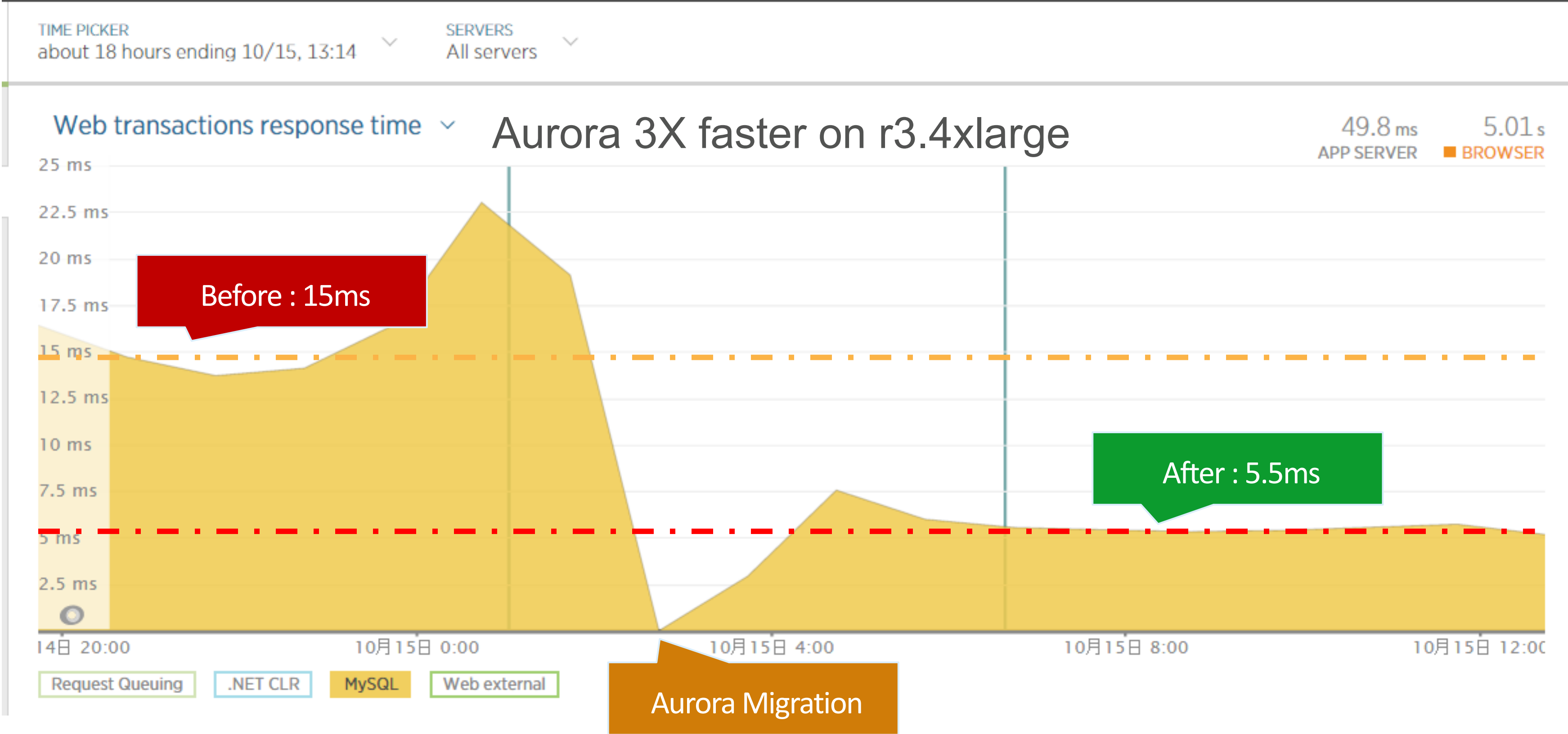
- Multi-engine support: Aurora, MySQL, MariaDB, PostgreSQL, Oracle, SQL Server
- Automated provisioning, patching, scaling, backup/restore, failover
- Use with GP2 or Provisioned IOPS storage
- High availability with RDS Multi-AZ
  - 99.95% SLA for Multi-AZ deployments

Amazon Aurora

Amazon Aurora

Real-life data – gaming workload

Aurora vs. RDS MySQL – r3.4XL, MAZ



Real-life data - read replica latency

Replica Lag Maximum (ms)

Statistic: Average Time Range: Last 3 Days Period: 1 Hour

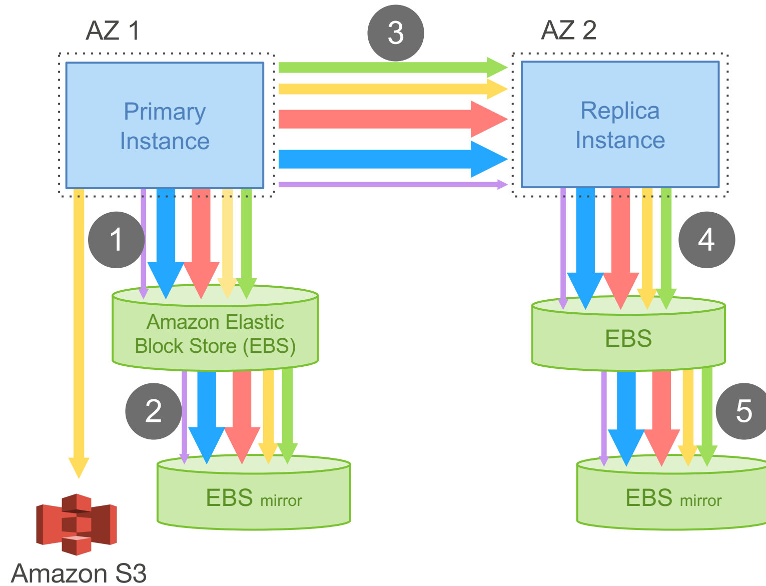
20
15
10
5
0

11/30 00:00 | 11/30 08:00 | 11/30 16:00 | 12/01 00:00 | 12/01 08:00 | 12/01 16:00 | 12/02 00:00 | 12/02 08:00 | 12/02 16:00

"In MySQL, we saw replica lag spike to almost 12 minutes which is almost absurd from an application's perspective. With Aurora, the maximum read replica lag across 4 replicas never exceeded 20 ms."

I/O traffic in MySQL

MYSQL WITH REPLICA



I/O FLOW

Issue write to EBS – EBS issues to mirror, ack when both done
Stage write to standby instance through DRBD
Issue write to EBS on standby instance

OBSERVATIONS

Steps 1, 3, 4 are sequential and synchronous
This amplifies both latency and jitter
Many types of writes for each user operation
Have to write data blocks twice to avoid torn writes

PERFORMANCE

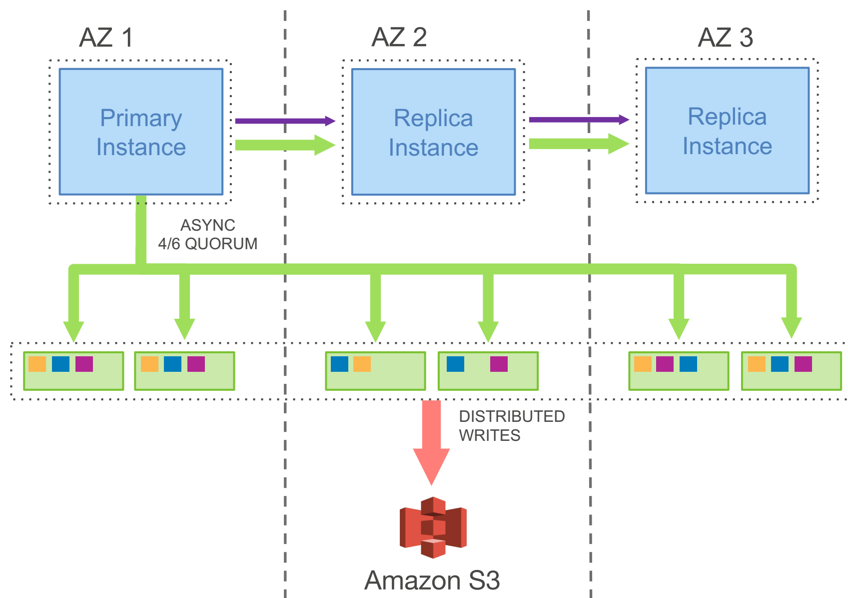
780K transactions
7,388K I/Os per million txns (excludes mirroring, standby)
Average 7.4 I/Os per transaction

30 minute SysBench writeonly workload, 100GB dataset, **RDS MultiAZ**, 30K PIOPS

TYPE OF WRITE

I/O traffic in Aurora

AMAZON AURORA



I/O FLOW

Boxcar redo log records – fully ordered by LSN
Shuffle to appropriate segments – partially ordered
Boxcar to storage nodes and issue writes

OBSERVATIONS

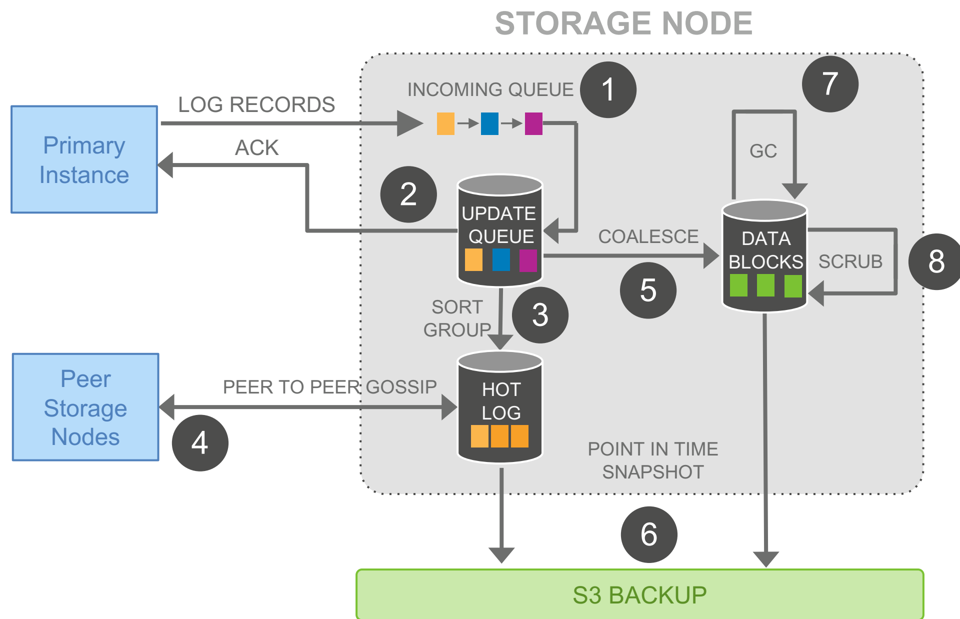
Only write redo log records; all steps asynchronous
No data block writes (checkpoint, cache replacement)
6X more log writes, but **9X less** network traffic
Tolerant of network and storage outlier latency

PERFORMANCE

27,378K transactions	**35X MORE**
950K I/Os per 1M txns (6X amplification)	**7.7X LESS**

TYPE OF WRITE

I/O traffic in Aurora (storage node)



I/O FLOW

1. Receive record and add to in-memory queue
2. Persist record and acknowledge
3. Organize records and identify gaps in log
4. Gossip with peers to fill in holes
5. Coalesce log records into new data block versions
6. Periodically stage log and new block versions to S3
7. Periodically garbage collect old versions
8. Periodically validate CRC codes on blocks

OBSERVATIONS

All steps are asynchronous

Only steps 1 and 2 are in foreground latency path

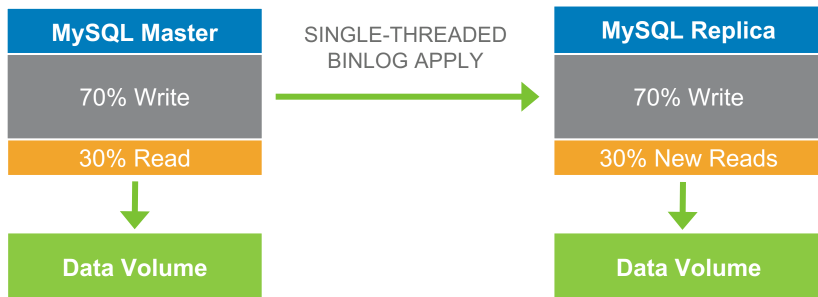
Input queue is **46X less** than MySQL (unamplified, per node)

Favor latency-sensitive operations

Use disk space to buffer against spikes in activity

I/O traffic in Aurora Replicas

MYSQL READ SCALING



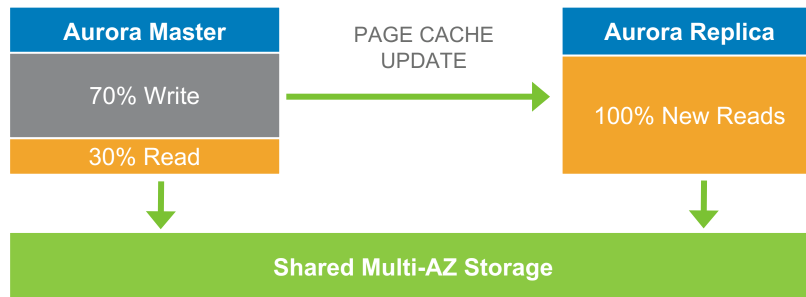
Logical: Ship SQL statements to Replica

Write workload similar on both instances

Independent storage

Can result in data drift between Master and Replica

AMAZON AURORA READ SCALING



Physical: Ship redo from Master to Replica

Replica shares storage. No writes performed

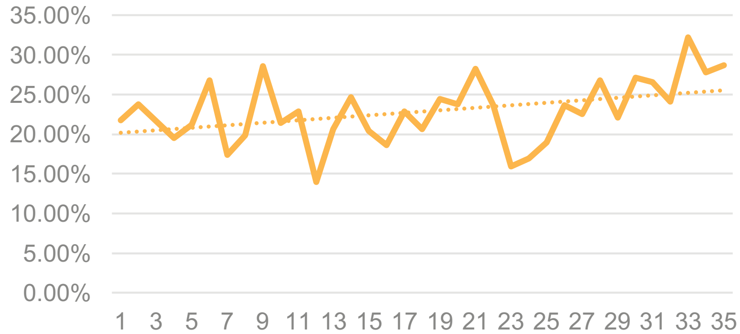
Cached pages have redo applied

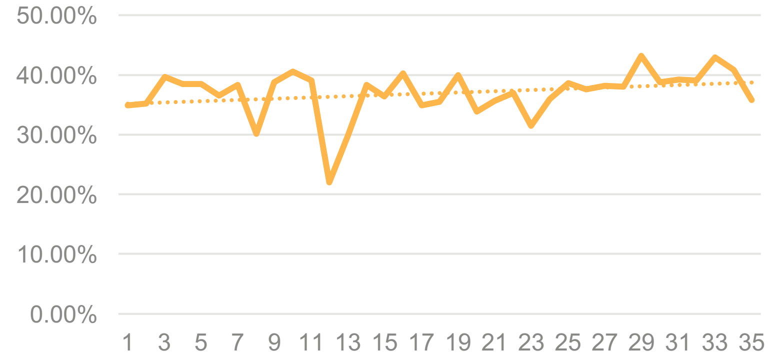
Advance read view when all commits seen

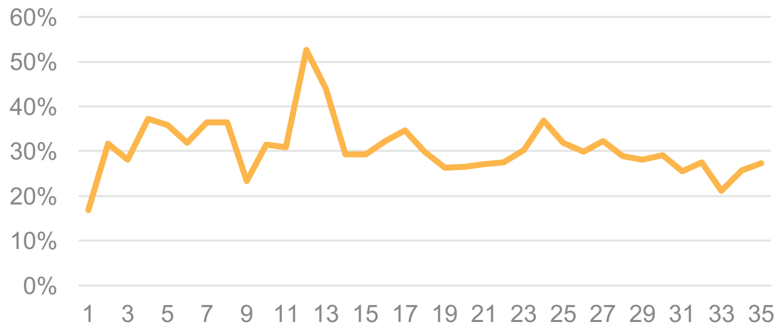
Availability

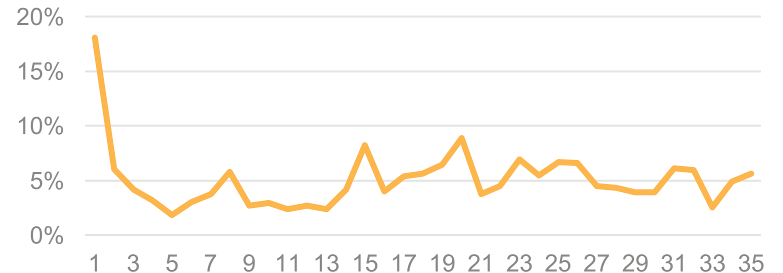
"Performance only matters if your database is up"

Database failover time









Survivable caches

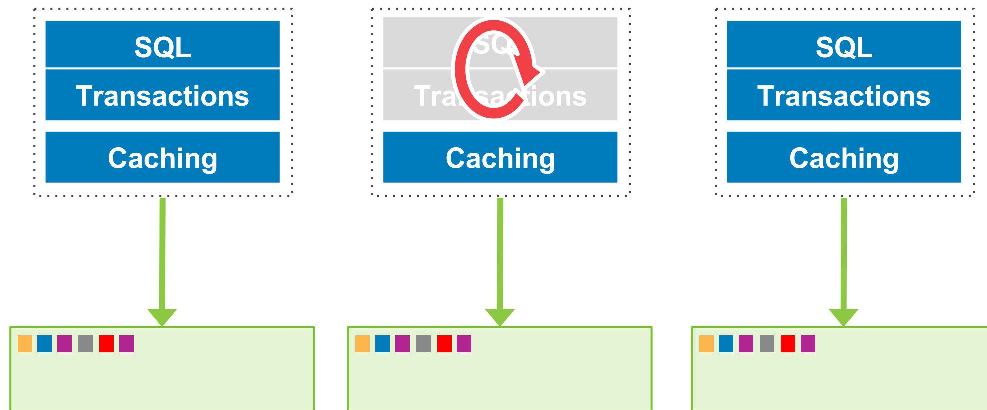
We moved the cache out of the database process

Cache remains warm in the event of database restart

Lets you resume fully loaded operations much faster

Instant crash recovery + survivable cache = quick and easy recovery from DB failures

Caching process is outside the DB process and remains warm across a database restart

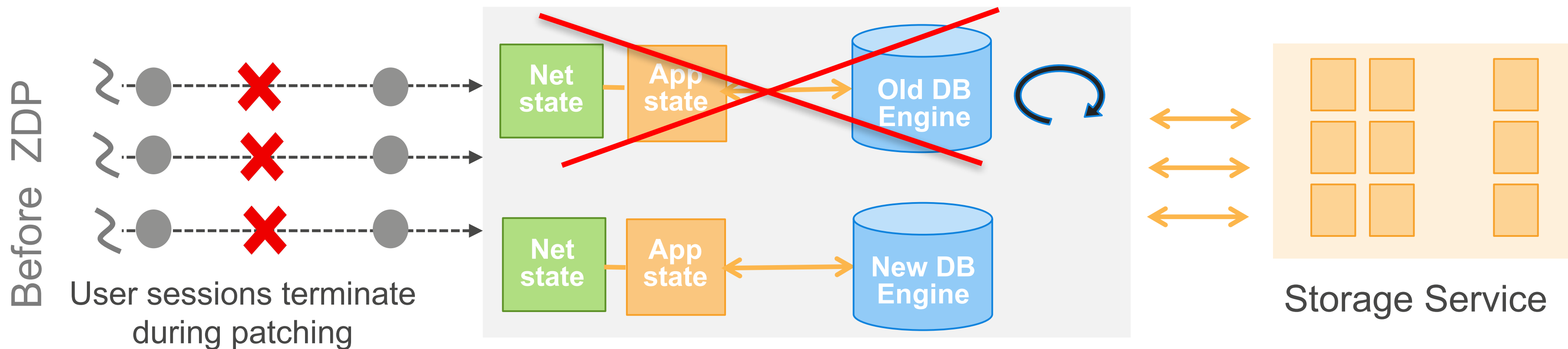


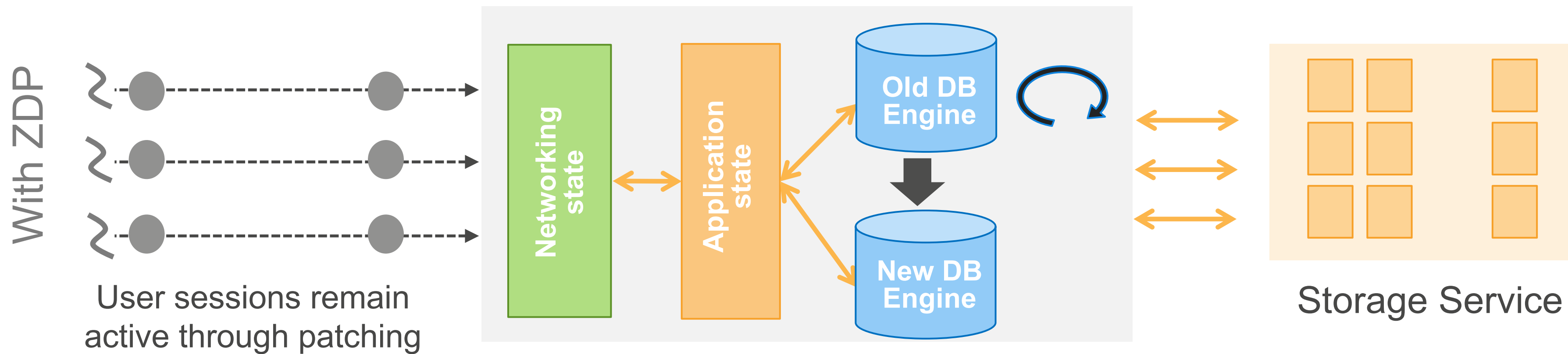
Availability is about more than HW failures

You also incur availability disruptions when you

1. Patch your database software
2. Modify your database schema
3. Perform large scale database reorganizations
4. Restore a database after a user error

Zero downtime patching





Zero downtime patching – current constraints

We have to go to our current patching model when we can’t park connections:

- Long running queries
- Open transactions
- Bin-log enabled
- Parameter changes pending
- Temporary tables open
- Locked tables
- SSL connections open
- Read replicas instances

We are working on addressing the above.

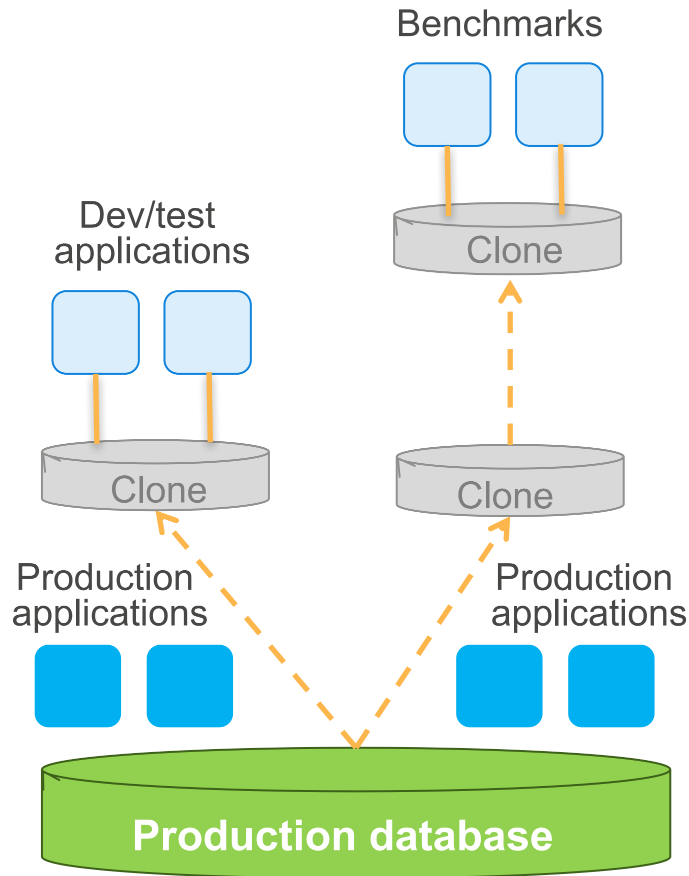
Database cloning

Create a copy of a database without duplicate storage costs

- Creation of a clone is nearly instantaneous – we don't copy data
- Data copy happens only on write – when original and cloned volume data differ

Typical use cases:

- Clone a production DB to run tests
- Reorganize a database
- Save a point-in-time snapshot for analysis without impacting production system



Reader Endpoint

New Reader Endpoint for Amazon Aurora – Load Balancing & Higher Availability

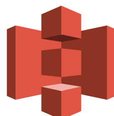
by Jeff Barr | on 08 SEP 2016 | in Amazon Aurora, Launch | Permalink | Comments

Feature-by-feature, Amazon Aurora has become more powerful and easier to use. Over the past months we have given you the ability to create a cluster from a MySQL backup, create cross-region read replicas, share snapshots across accounts, exercise additional control over failover, and migrate from other in-cloud or on-premises databases to Aurora.

Today, as an extension of Aurora's existing read replica model, we are introducing a new cluster-level read endpoint. Your application can continue to direct read queries to the individual replicas as before. However, you can also update it to make use of the new endpoint. Doing so will give you two important advantages: load balancing and higher availability:

AWS ecosystem

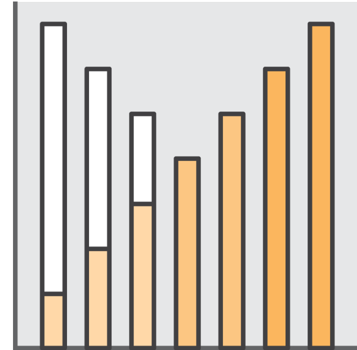
Lambda Generate AWS Lambda events from Aurora stored procedures.

S3  Load data from Amazon S3, store snapshots and backups in S3.

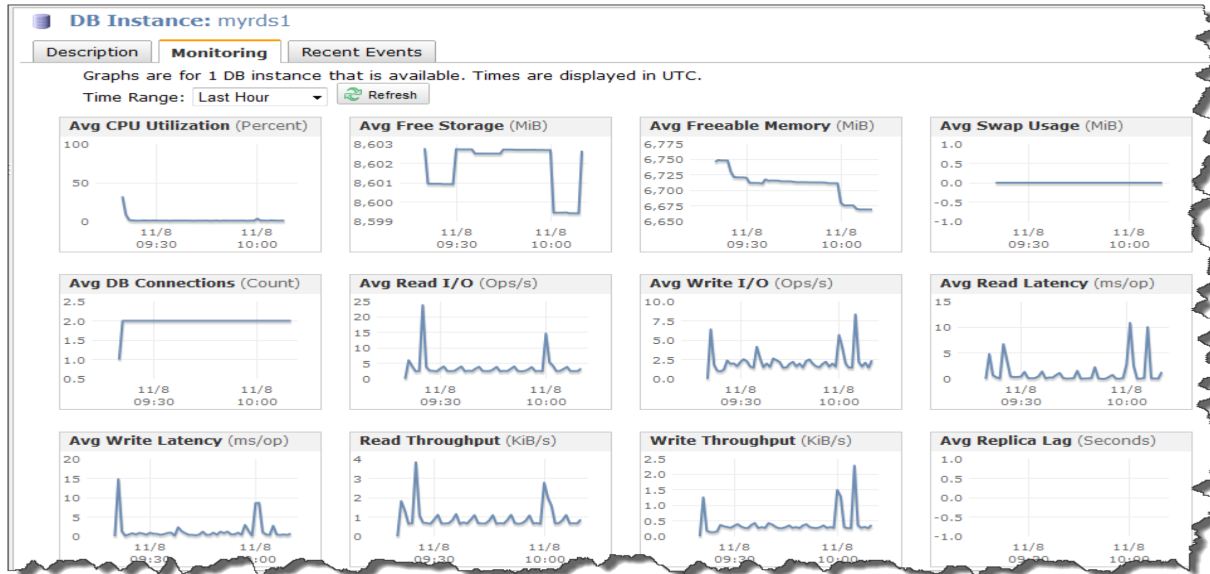
IAM Use AWS IAM roles to manage database access control.

CloudWatch Upload systems metrics and audit logs to Amazon CloudWatch.

RDS Metrics and monitoring



Standard monitoring



Amazon CloudWatch metrics for Amazon RDS

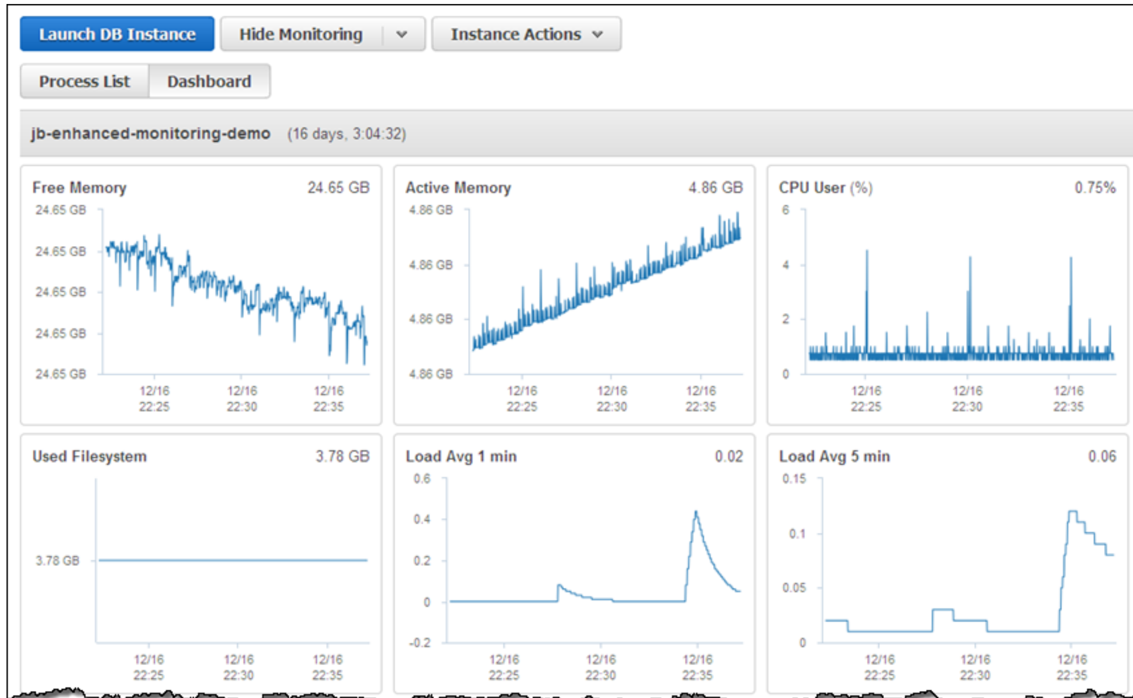
- CPU utilization
- Storage
- Memory
- Swap usage
- DB connections
- I/O (read and write)
- Latency (read and write)
- Throughput (read and write)
- Replica lag
- Many more

Amazon CloudWatch Alarms

- Similar to on-premises custom monitoring tools

Enhanced Monitoring

Access to over 50 new CPU, memory, file system, and disk I/O metrics as low as 1 second intervals



Monitoring

Enable Enhanced Monitoring: Yes

Monitoring Role: Default

Granularity: 1 second(s)

☑ I authorize RDS to create the IAM role rds-monitoring-role.

Amazon DynamoDB

Fast, Flexible, Scalable NoSQL

History of NoSQL at Amazon

Dynamo: Amazon's Highly Available Key-value Store

Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall and Werner Vogels

Amazon.com

Original

ABSTRACT

Reliability at massive scale is one of the biggest challenges we face at Amazon.com, one of the largest e-commerce operations in the world; even the slightest outage has significant financial consequences and impacts customer trust. The Amazon.com platform, which provides services for many web sites worldwide, is implemented on top of an infrastructure of tens of thousands of servers and network components located in many datacenters around the world. At this scale, small and large components fail continuously and the way persistent state is managed in the face of these failures drives the reliability and scalability of the software systems.

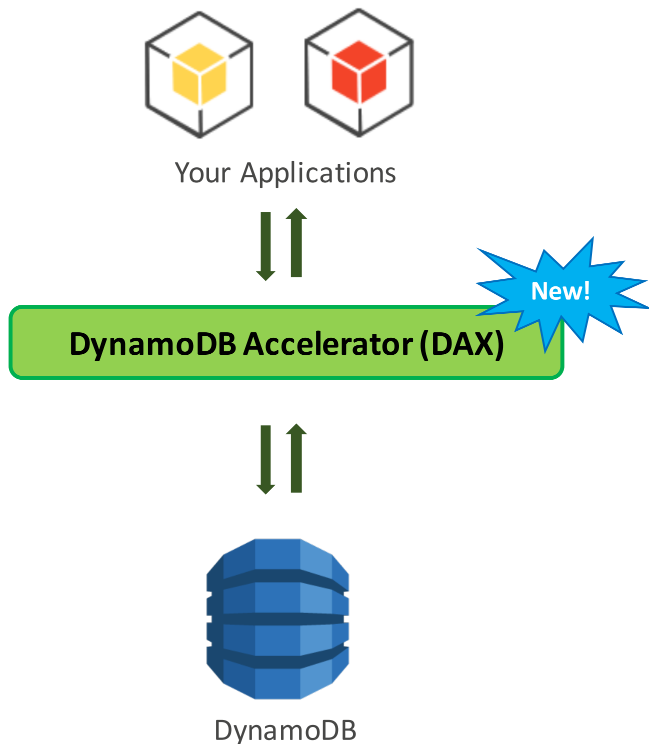
This paper presents the design and implementation of Dynamo, a highly available key-value storage system that some of Amazon's core services use to provide an "always-on" experience. To achieve this level of availability, Dynamo sacrifices consistency under certain failure scenarios. It makes extensive use of object versioning and application-assisted conflict resolution in a manner that provides a novel interface for developers to use.

One of the lessons our organization has learned from operating Amazon's platform is that the reliability and scalability of a system is dependent on how its application state is managed. Amazon uses a highly decentralized, loosely coupled, service oriented architecture consisting of hundreds of services. In this environment there is a particular need for storage technologies that are always available. For example, customers should be able to view and add items to their shopping cart even if disks are failing, network routes are flapping, or data centers are being destroyed by tornados. Therefore, the service responsible for managing shopping carts requires that it can always write to and read from its data store, and that its data needs to be available across multiple data centers.

Dealing with failures in an infrastructure comprised of millions of components is our standard mode of operation; there are always a small but significant number of server and network components that are failing at any given time. As such Amazon's software systems need to be constructed in a manner that treats failure handling as the normal case without impacting availability or performance.

NEW

DynamoDB Accelerator (DAX)



Features

- **Fully managed:** handle all of the upgrades, patching, and software management
- **Flexible**: Configure DAX for one table or many
- **Highly available:** fault tolerant, replication across multi-AZs within a region
- **Scalable**: scales-out to any workload with up to 10 read replicas
- **Manageability:** fully integrated AWS service: Amazon CloudWatch, Tagging for DynamoDB, AWS Console
- **Security:** Amazon VPC, AWS IAM, AWS CloudTrail, AWS Organizations

NEW

Time To Live (TTL)

TTL Attribute

ID	Name	Size	Expiry
1234	A	100	1456702305
2222	B	240	1456702400
3423	C	150	1459207905

TTL Value

Features

Generally Available

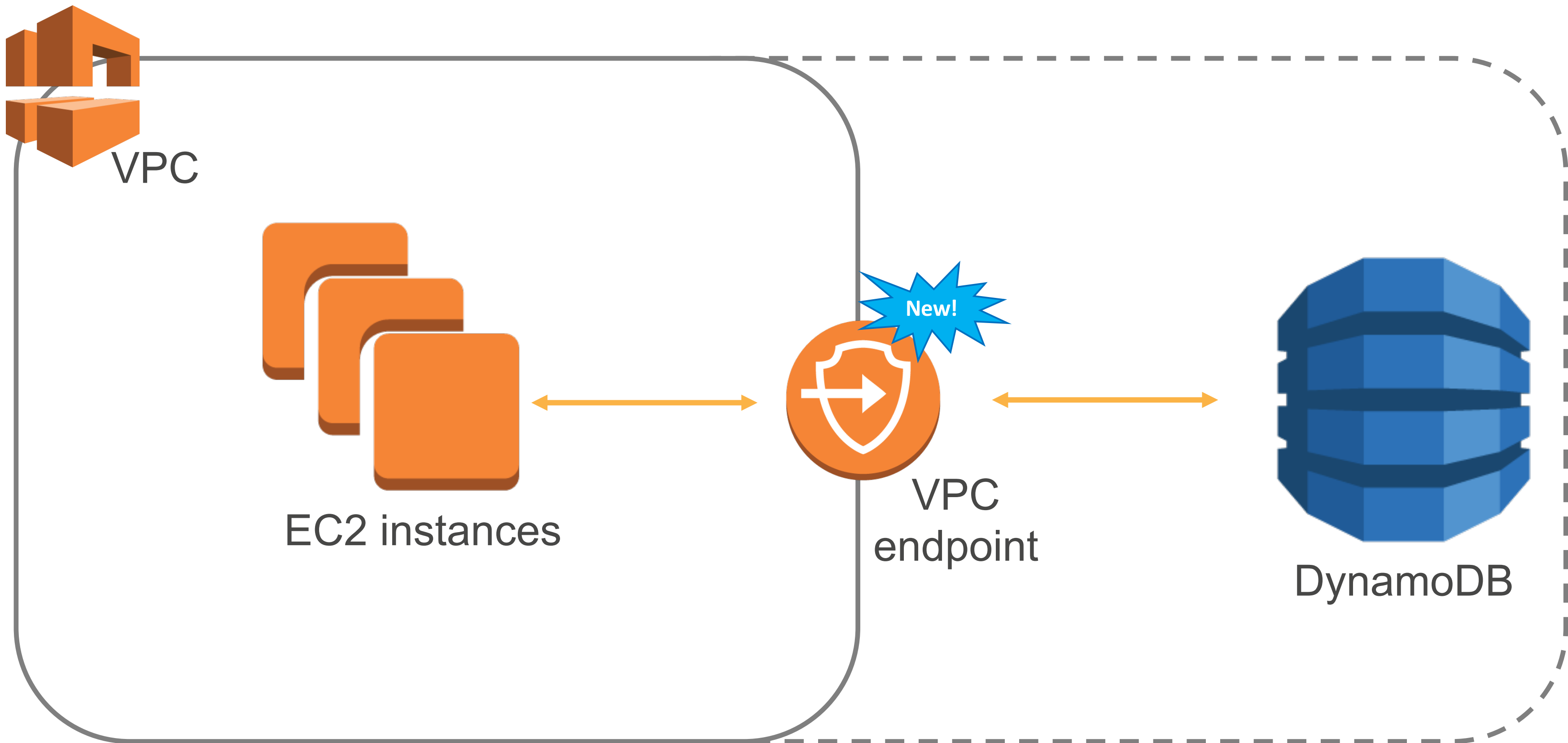
- **Automatic:** Deletes items from a table based on expiration timestamp
- **Customizable:** User-defined TTL attribute in epoch time format
- **Audit Log:** TTL activity recorded in DynamoDB Streams

Key Benefits

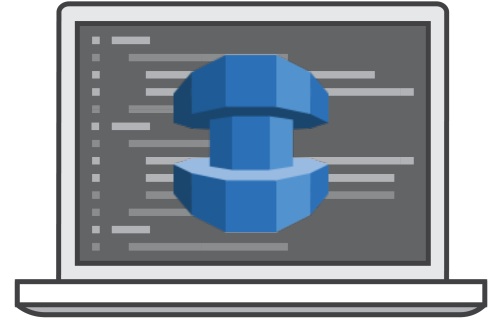
- **Reduce costs:** Delete items no longer needed
- **Performance:** Optimize application performance by controlling table size growth
- **Extensible:** Trigger custom workflows with DynamoDB Streams and Lambda

NEW

VPC Endpoints for DynamoDB



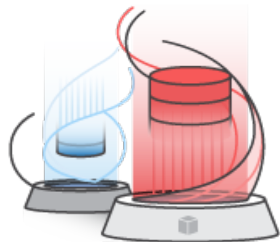
AWS Database Migration Service



AWS Schema Conversion Tool

What are DMS and SCT?

AWS Database Migration Service (DMS) easily and securely migrate and/or replicate your databases *and* data warehouses to AWS



AWS Schema Conversion Tool (SCT) convert your commercial database and data warehouse schemas to open-source engines or AWS-native services, such as Amazon Aurora and Redshift

We migrated over 19,000 unique databases. And counting…

When to use DMS and SCT?

Modernize

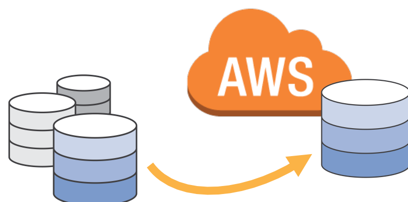
Modernize your database tier –

- Commercial to open-source
- Commercial to Amazon Aurora

Modernize your Data Warehouse –

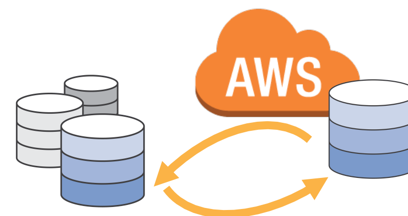
- Commercial to Redshift

Migrate



- Migrate business-critical applications
- Migrate from Classic to VPC
- Migrate data warehouse to Redshift
- Upgrade to a minor version
- Consolidate shards into Aurora

Replicate



- Create cross-regions Read Replicas
- Run your analytics in the cloud
- Keep your dev/test and production environment sync

When to use DMS and SCT? *Modernize*

Modernize your Database Tier

- Commercial to open-source
- Commercial to Amazon Aurora

ORACLE

Amazon Aurora

Modernize your Warehouse

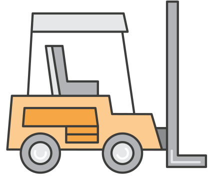
- Commercial to Redshift

TERADATA

ORACLE

Amazon Redshift

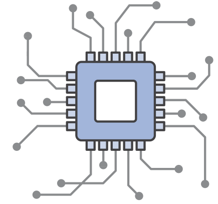
Why use DMS and SCT?



Remove Barriers to Entry

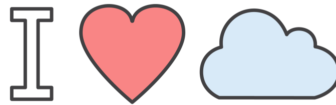
Near-Zero Downtime

Secure



Easy to Use, but Sophisticated…

Allow DB Freedom



Keep a Leg in the Cloud

Cost Effective

AWS re:Invent

How does it work?

© 2016, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

amazon web services

Database migration process

STEP 1:

STEP 2:

SCT helps with converting tables, views, and code

- Sequences
- User-defined types
- Synonyms
- Packages
- Stored procedures
- Functions
- Triggers
- Schemas
- Tables
- Indexes
- Views
- Sort and distribution keys

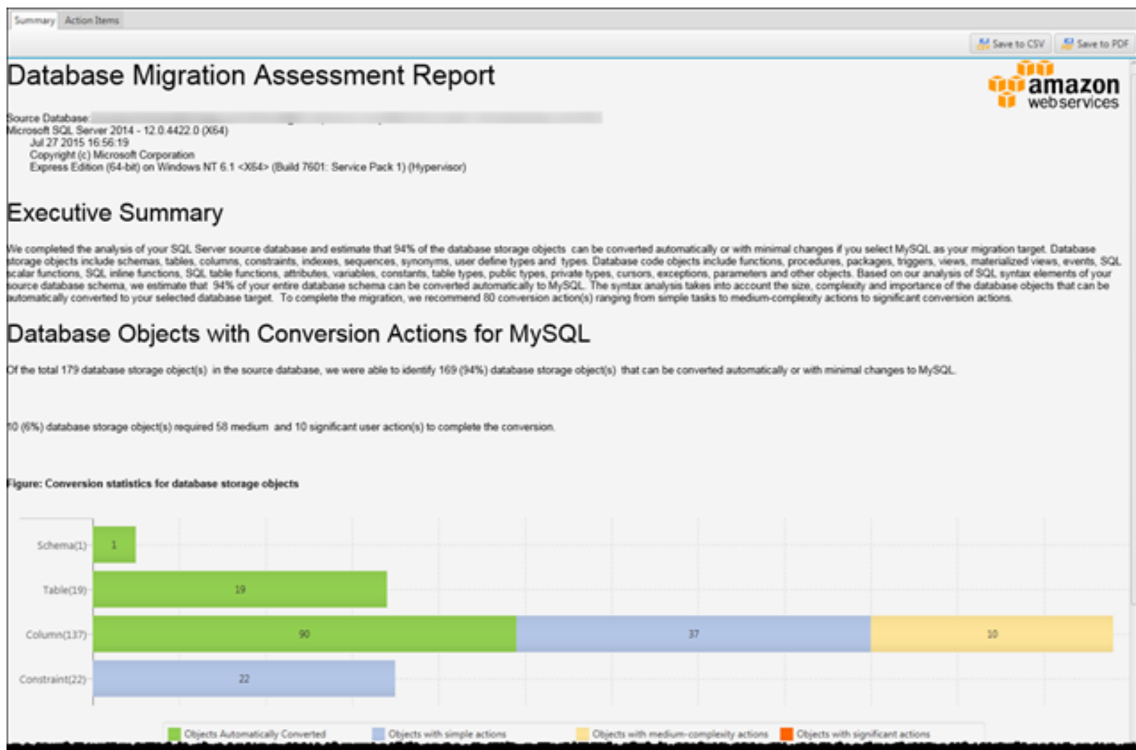
SCT supported OLTP sources and targets

Source Database	Target Database on Amazon RDS
Microsoft SQL Server (version 2008 and later)	Amazon Aurora (MySQL or PostgreSQL), Microsoft SQL Server, MySQL, PostgreSQL
MySQL (version 5.5 and later)	Amazon Aurora (PostgreSQL), MySQL, PostgreSQL
Oracle (version 10.2 and later)	Amazon Aurora (MySQL or PostgreSQL), MySQL, Oracle, PostgreSQL
PostgreSQL (version 9.1 and later)	Amazon Aurora (MySQL), MySQL, PostgreSQL

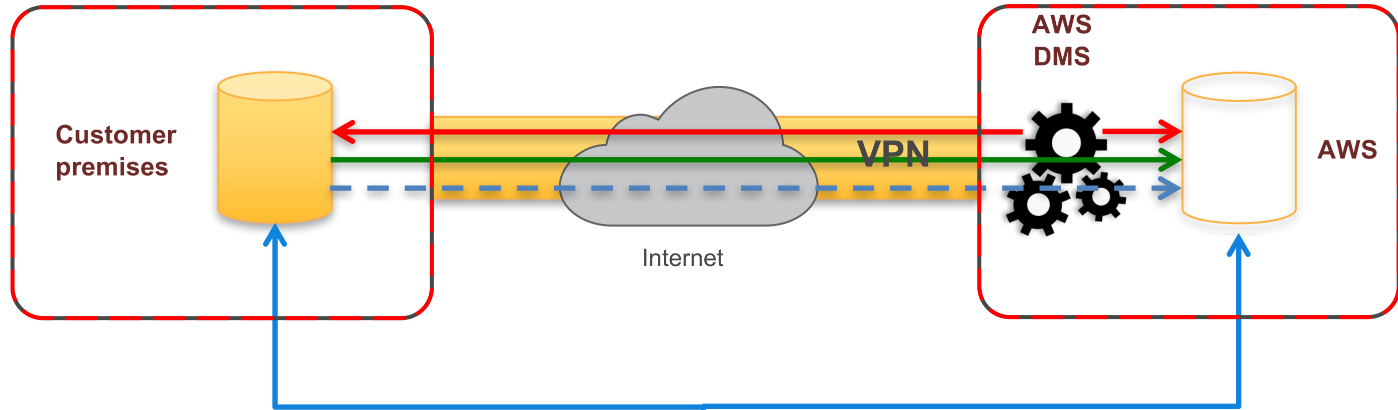
SCT supported DW sources and targets

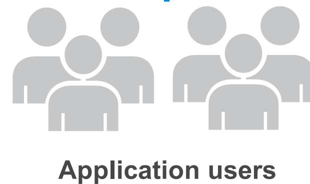
Source Database	Target Database on Amazon Redshift
Greenplum Database (version 4.3 and later)	Amazon Redshift
Microsoft SQL Server (version 2008 and later)	Amazon Redshift
Netezza (version 7.0.3 and later)	Amazon Redshift
Oracle (version 10 and later)	Amazon Redshift
Teradata (version 13 and later)	Amazon Redshift
Vertica (version 7.2.2 and later)	Amazon Redshift

SCT guidance



Keep your apps running during the migration





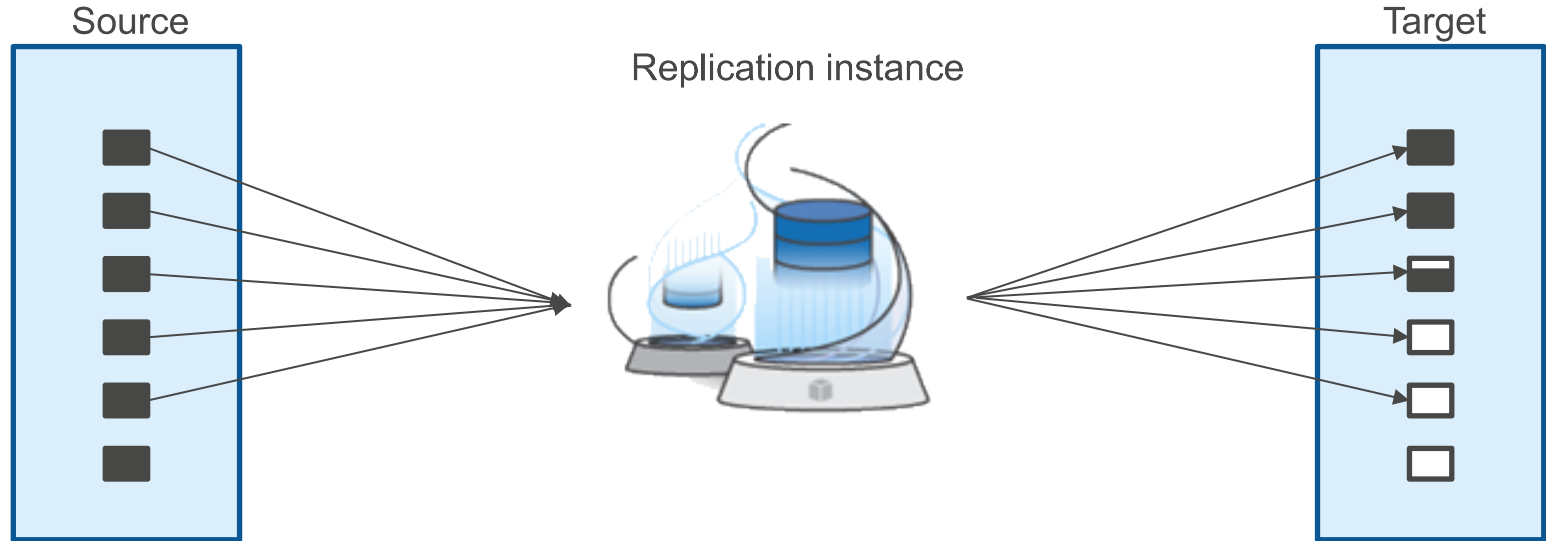
Start a replication instance

Connect to source and target databases

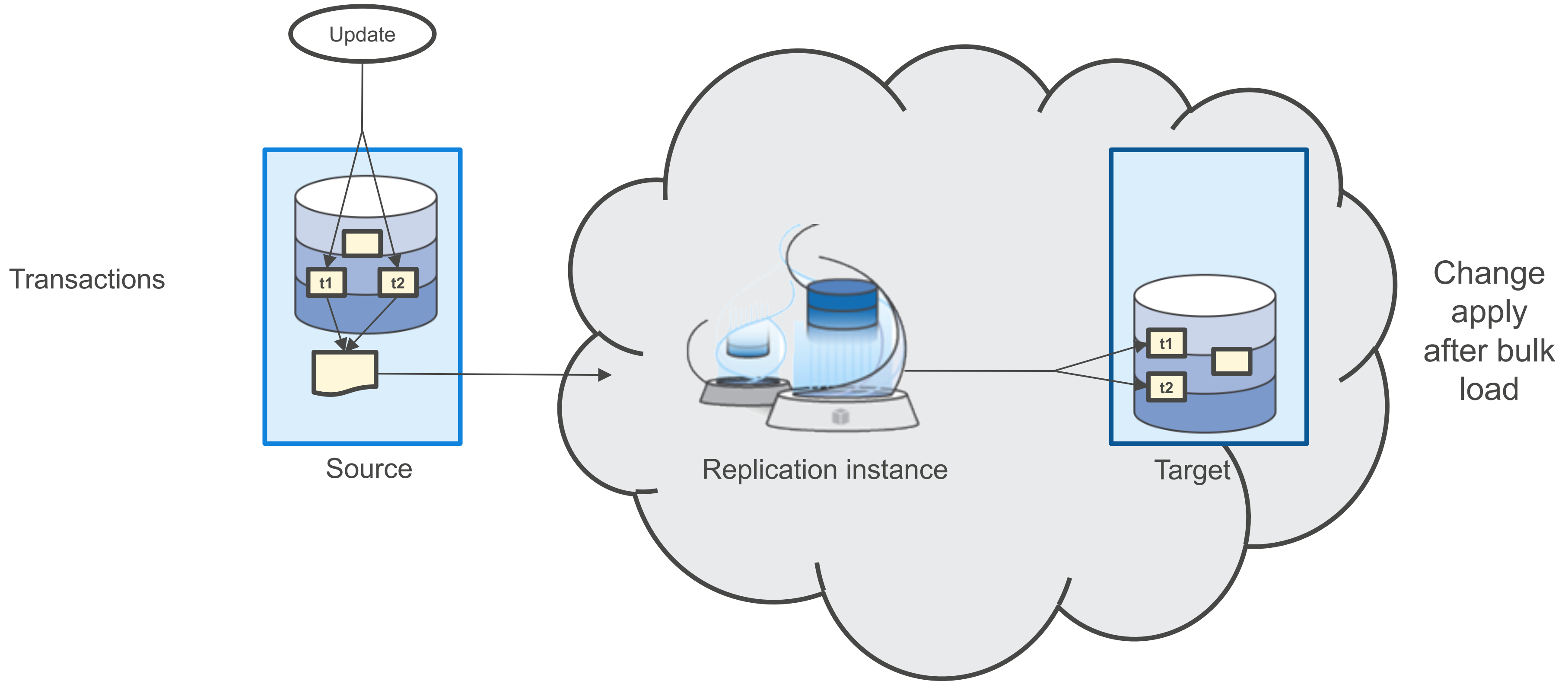
Select tables, schemas, or databases

- Let AWS DMS create tables, load data, and keep them in sync
- Switch applications over to the target at your convenience

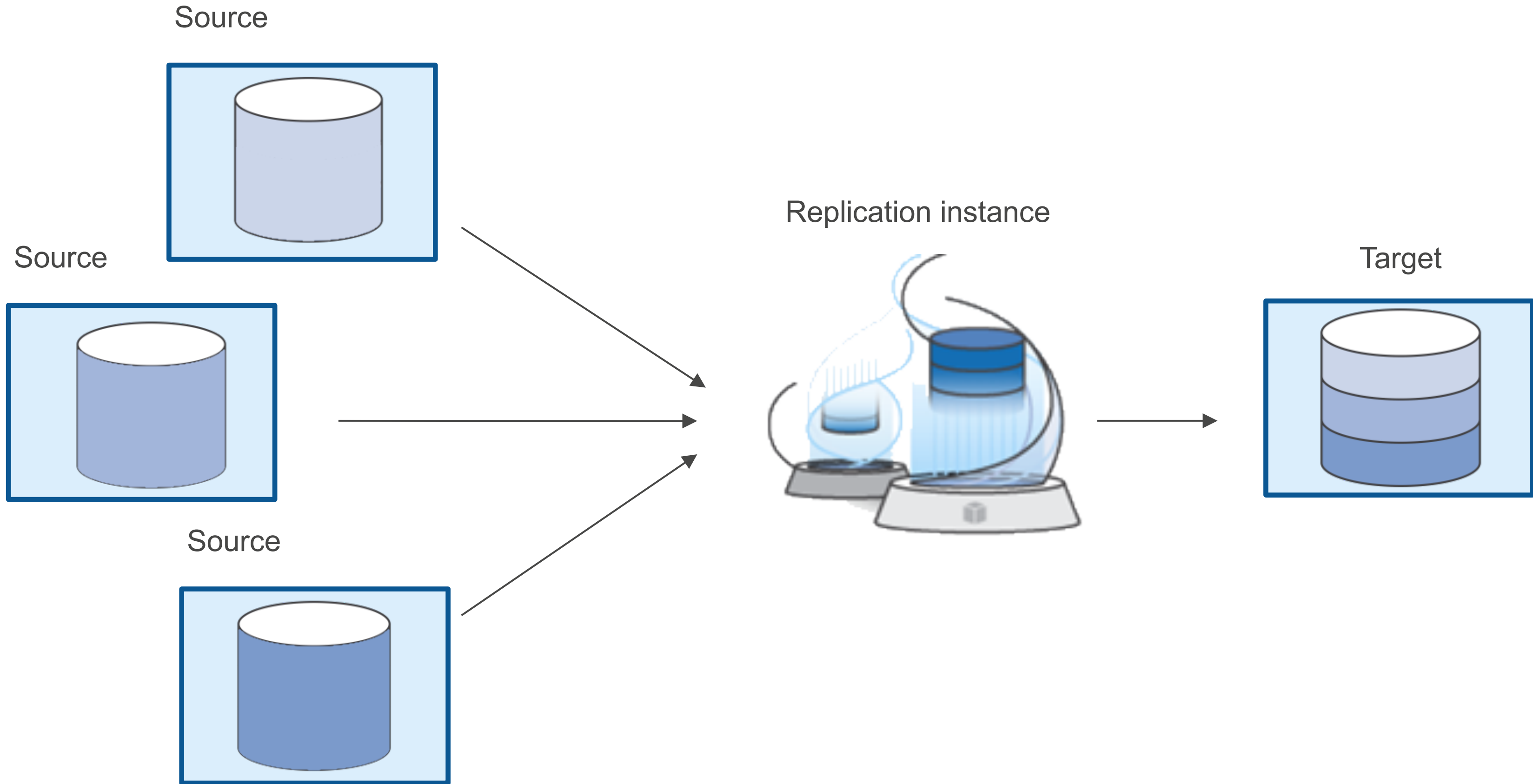
Load is table by table



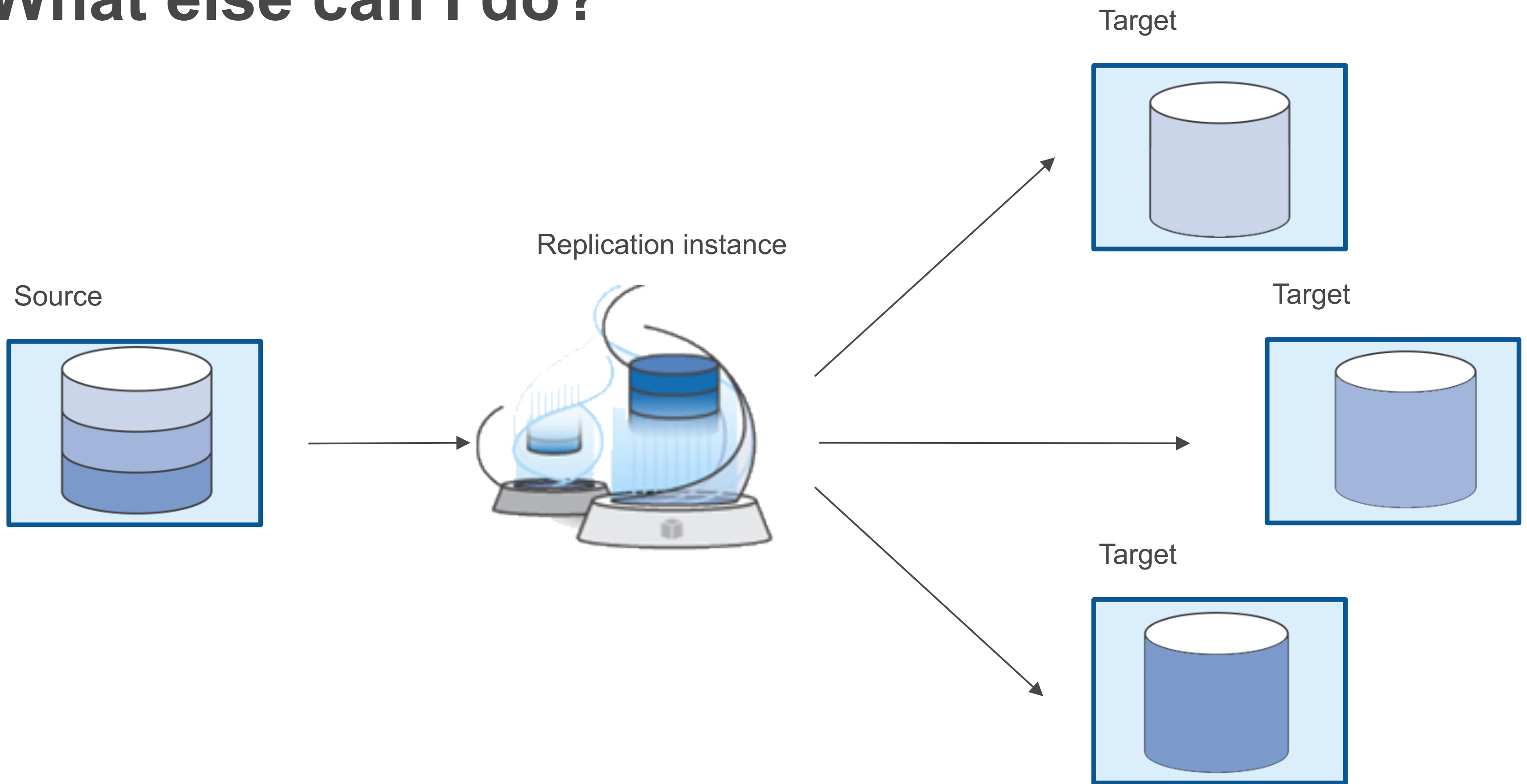
Change data capture (CDC) and apply



What else can I do?

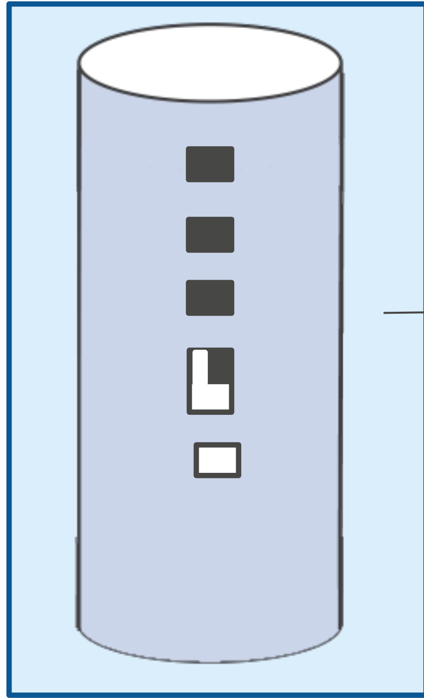


What else can I do?



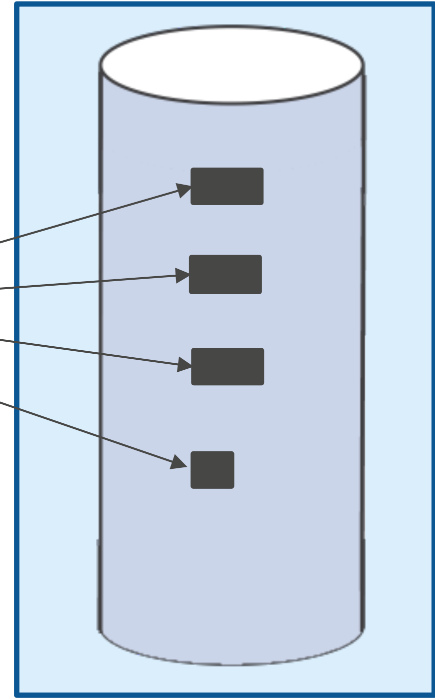
Take it all—or not

Source

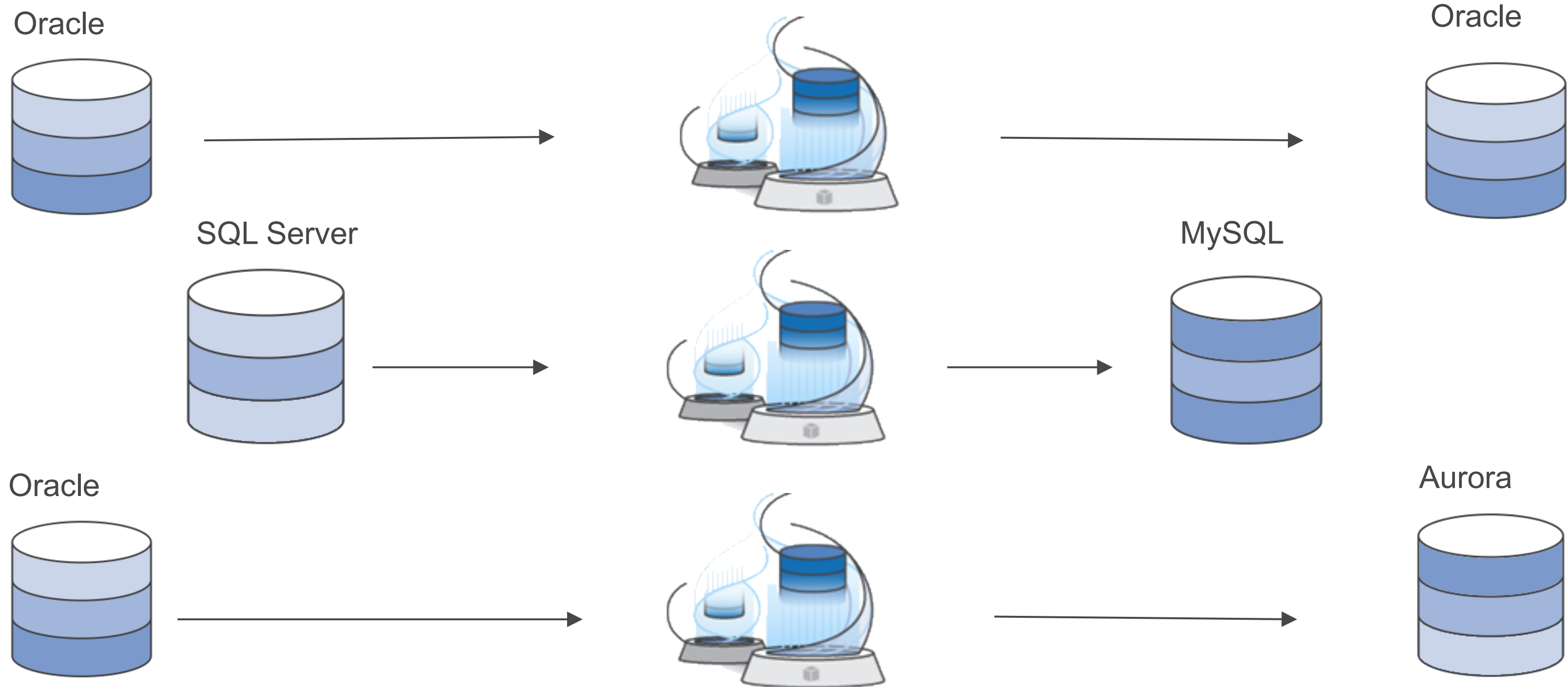


Replication instance

Target



Homogenous or heterogeneous



Other database migration use cases

Migration of business-critical applications
Migration from Classic to VPC
Cheap Read Replicas for Oracle
Read Replicas for other engines
Cross-region Read Replicas for Oracle and SQL Server
Analytics in the cloud
Dev/test and production environment sync
Ongoing replication for BI
Minor version upgrade

Sources for AWS Database Migration Service

Customers use the following databases as a source for data migration using AWS DMS:

On-premises and Amazon EC2 instance databases:

- Oracle Database 10g–12c
- Microsoft SQL Server 2005–2014
- MySQL 5.5–5.7
- MariaDB (MySQL-compatible data source)
- PostgreSQL 9.4–9.5
- SAP ASE 15.7+

RDS instance databases:

- Oracle Database 11g–12c
- Microsoft SQL Server 2008R2–2014. CDC operations are not supported yet.
- MySQL versions 5.5–5.7
- MariaDB (MySQL-compatible data source)
- PostgreSQL 9.4–9.5. CDC operations are not supported yet.
- Amazon Aurora (MySQL-compatible data source)

Targets for AWS Database Migration Service

Customers can use the following databases as a target for data replication using AWS DMS:

On-premises and EC2 instance databases:

- Oracle Database 10g–12c
- Microsoft SQL Server 2005–2014
- MySQL 5.5–5.7
- MariaDB (MySQL-compatible data source)
- PostgreSQL 9.3–9.5
- SAP ASE 15.7+

RDS instance databases:

- Oracle Database 11g–12c
- Microsoft SQL Server 2008 R2 - 2014
- MySQL 5.5–5.7
- MariaDB (MySQL-compatible data source)
- PostgreSQL 9.3–9.5
- Amazon Aurora (MySQL-compatible data source)

Amazon Redshift

Q & A

https://aws.amazon.com/blogs/big-data/

https://aws.amazon.com/big-data

amazon web services