

# Deep dive and best practices for Amazon Athena

Roy Hasson

Global Business Development Manager, Amazon Athena

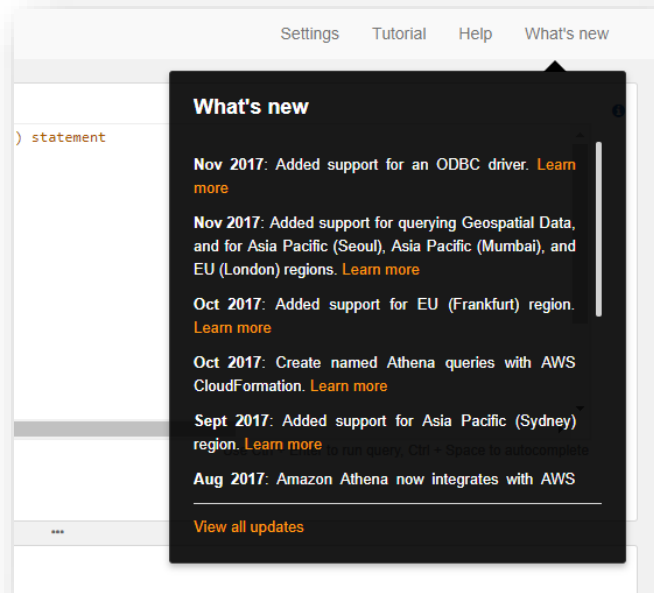


# Agenda

1. Review of the year
2. Use-cases seen since launch
3. Athena, Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

# Many new features and improvements

1. Views
2. Simpler schema evolution for Parquet/ORC
3. Rest API with support for SDK in most languages
4. Auto-complete and Tabs in the Athena console
5. Spill-to-disk for Group By
6. JDBC and ODBC drivers performance improvements
7. Geospatial functions
8. Support for correlated sub-queries
9. Support for Presto Lambda expressions and functions
10. Added ability to skip header lines for CSV formats



<https://docs.aws.amazon.com/athena/latest/ug/release-notes.html>



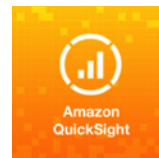
# Amazon Athena



# Easily analyze & visualize your data with your favorite tools



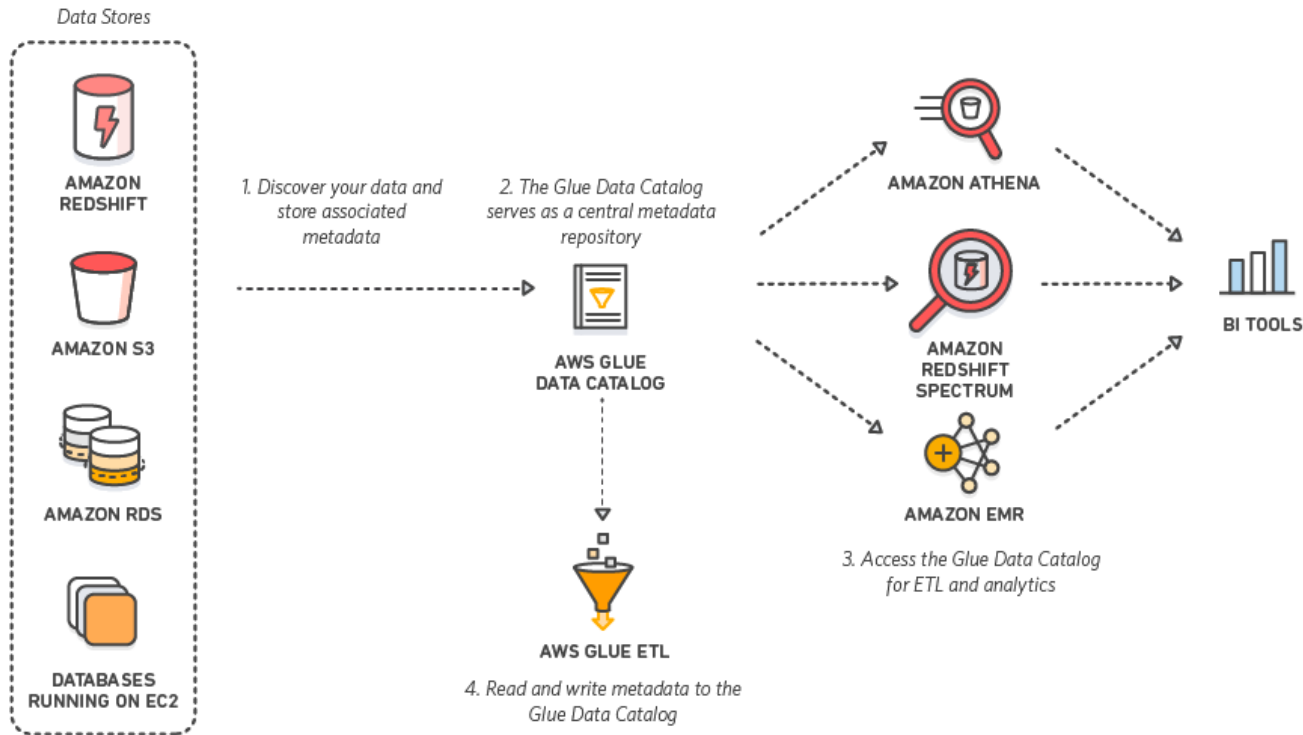
## Featured Athena Partners



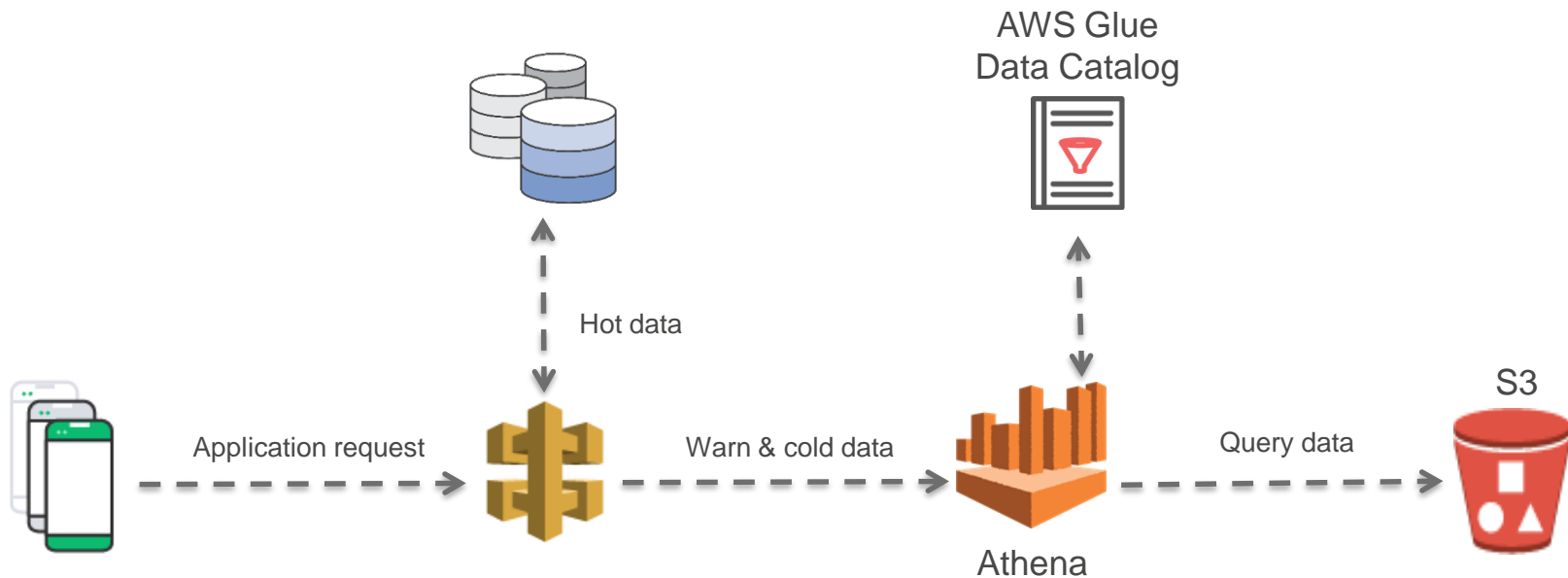
# Agenda

1. Review of the year
2. Use-cases seen since launch
3. Athena, Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

# Use Case 1: Query engine for data lake



## 2: Embed SQL for direct access to data





# Timber.io

## DITCH YOUR NOISY, HARD-TO-USE LOGS

Timber automatically enhances your logs through our [libraries and integrations](#). Converting them from messy, hard-to-use, raw text to [rich, useful, clean events](#).



Server



App



Framework



Log Files

```
I, [2017-06-04T18:04:53.653812 #42348] INFO -- :  
[my.host.com] [df88dbaa-50fd-4178-85d7-d66279ea33b6]  
[192.32.23.12] [bfa8242cd9733bf0211e334be203f0d0]  
Started GET "/" for 127.0.0.1 at 2012-03-10 14:28:14 +0100
```



Platform



Container



System



Database



HTTP

### HTTP Requests

Automatic logging of important HTTP request/response information.



### Exceptions

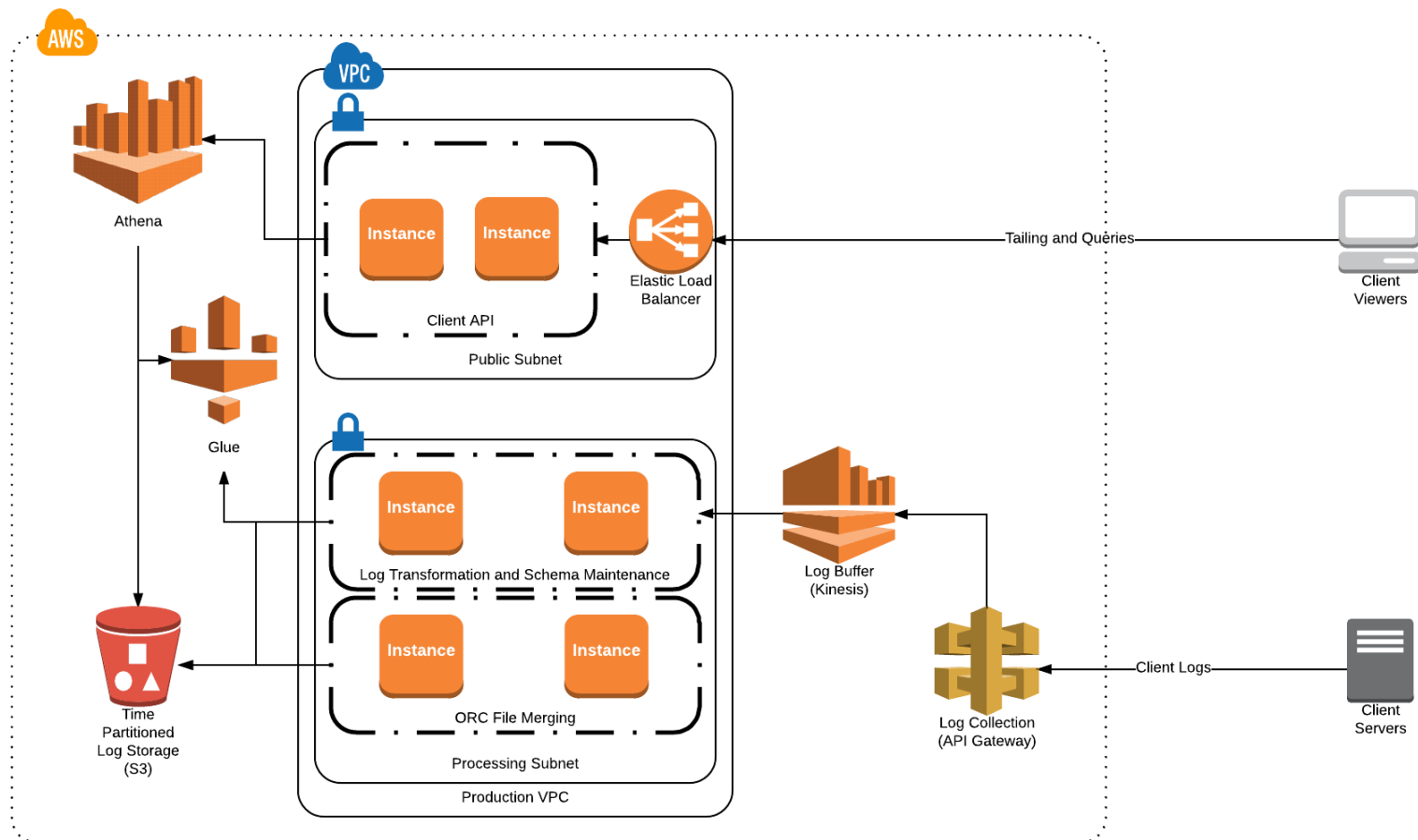
Automatically capture exceptions and view stack traces.



### User Context

Always know which user generated which log with automatic user context.





### 3: Migration from an On-premises infrastructure

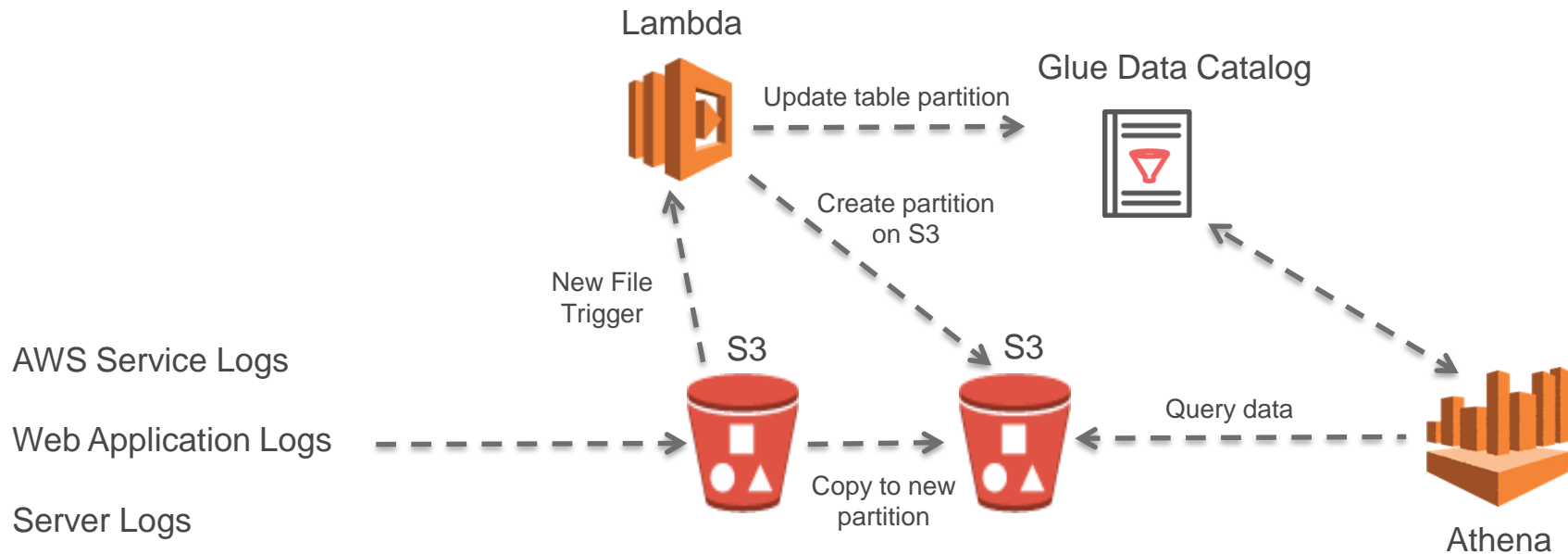


- ETL
- Data Wrangling
- Interactive Querying
- Machine Learning
- Data Science
- Hive Metastore

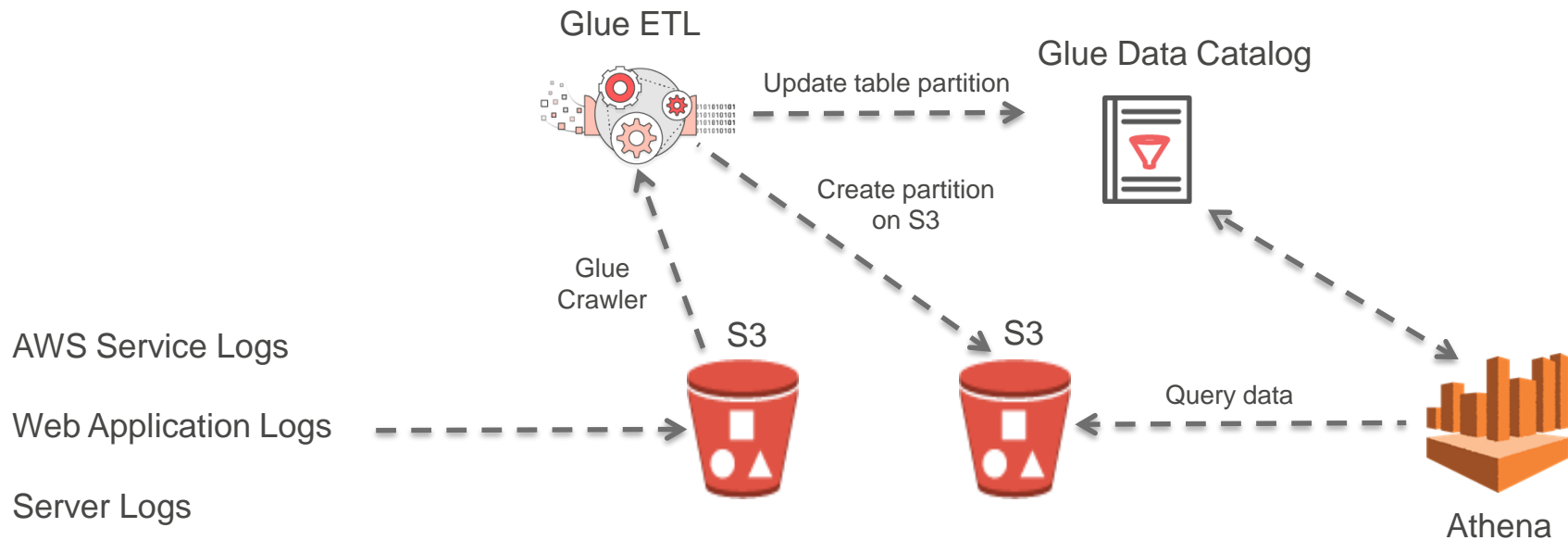


- **Glue ETL** and **EMR** for ETL
- **Athena, EMR** and **Redshift**
- **SageMaker** and **EMR** for ML
- **Glue Data Catalog** Hosted  
Metadata Catalog

## 4: Querying Logs



## 4a. Querying logs with ETL



# Airbnb's StreamAlert

- Deployment is automated: simple, safe and repeatable for any AWS account
- Easily scalable from megabytes to terabytes per day
- Minimal Infrastructure maintenance
- Infrastructure security is a default, no security expertise required
- Supports data from different environments (ex: IT, PCI, Engineering)
- Supports data from different environment types (ex: Cloud, Datacenter, Office)
- Supports different types of data (ex: JSON, CSV, Key-Value, or Syslog)
- Supports different use-cases like security, infrastructure, compliance and more

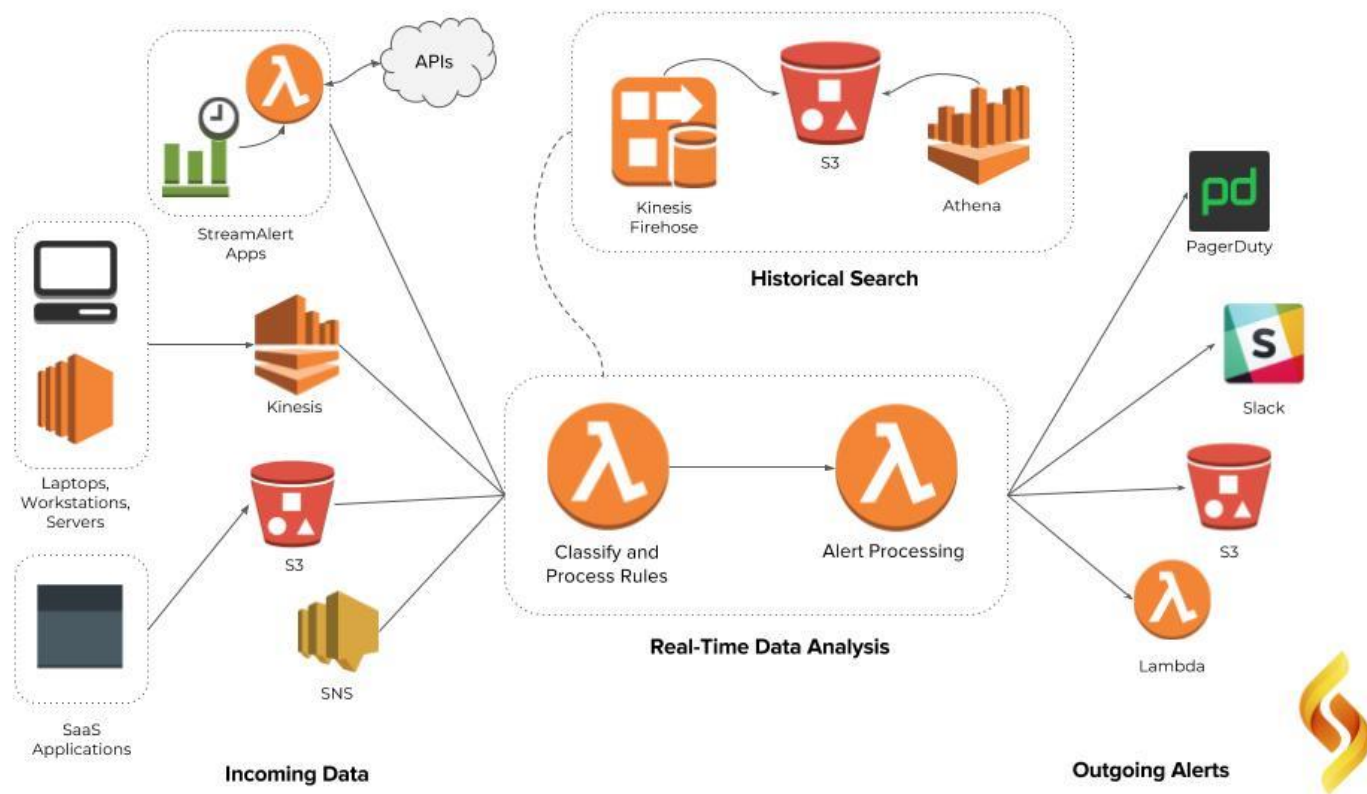
## StreamAlert - Serverless, Realtime Data Analysis Framework

build passing coverage 100%

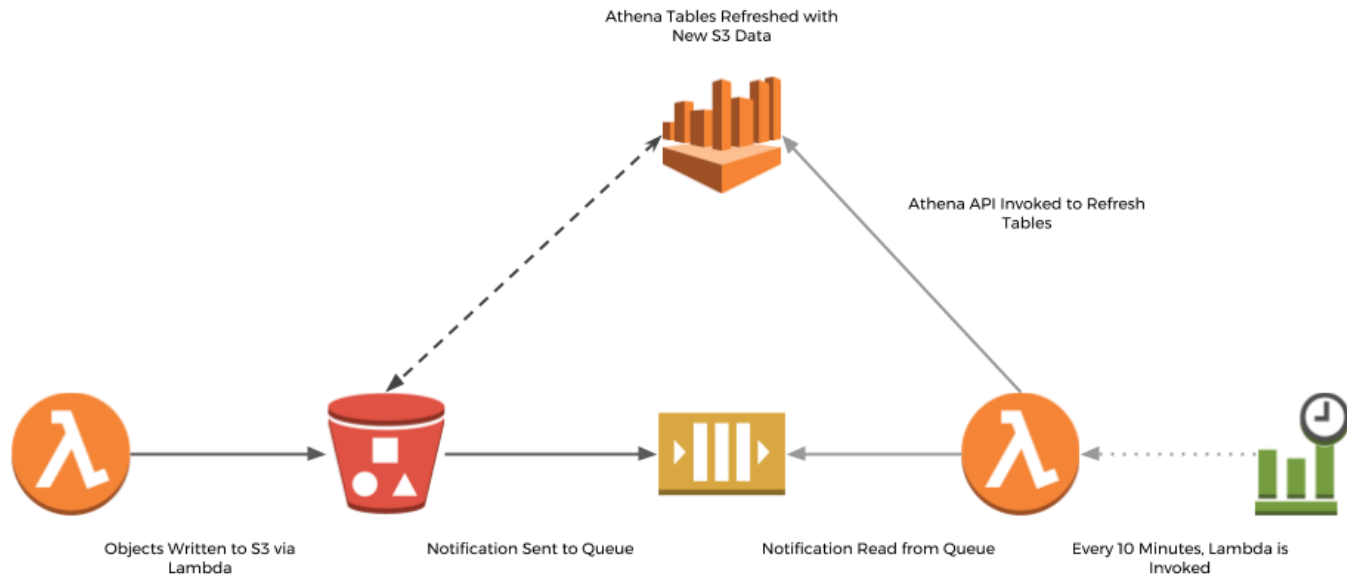


StreamAlert is a serverless, realtime data analysis framework which empowers you to ingest, analyze, and alert on data from any environment, using datasources and alerting logic you define.

# Overall Architecture

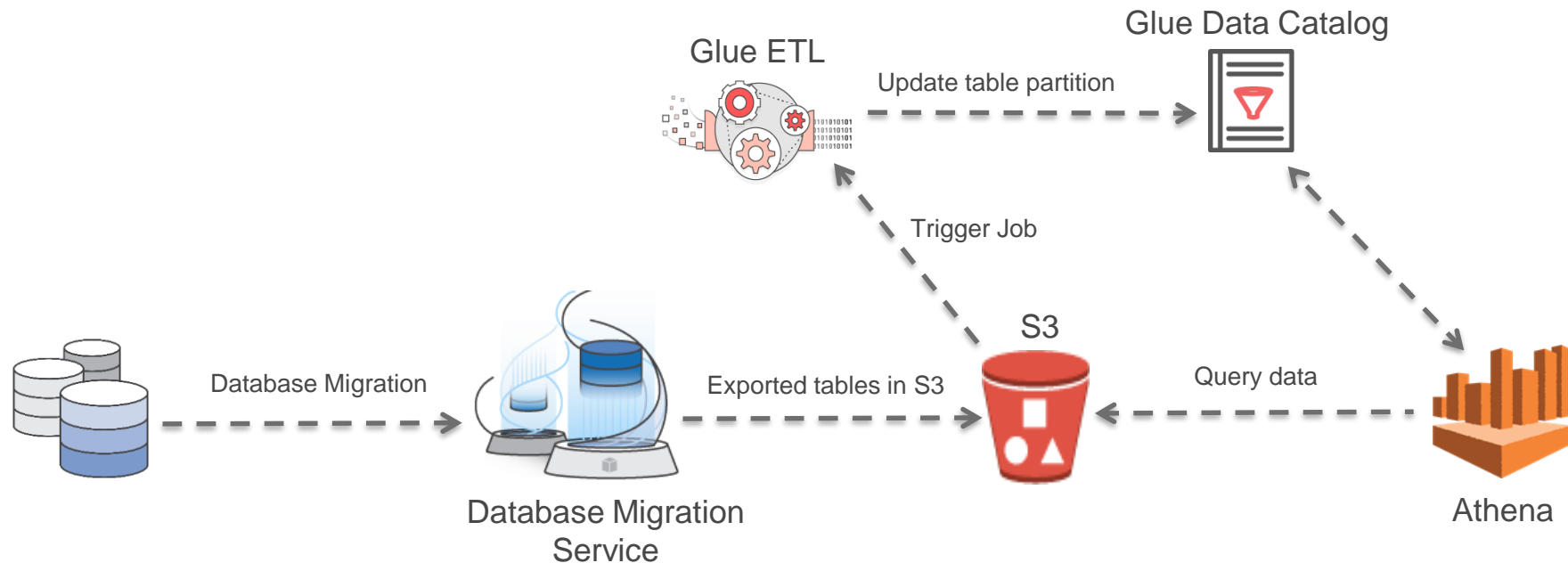


# How is Athena used in Streamalert

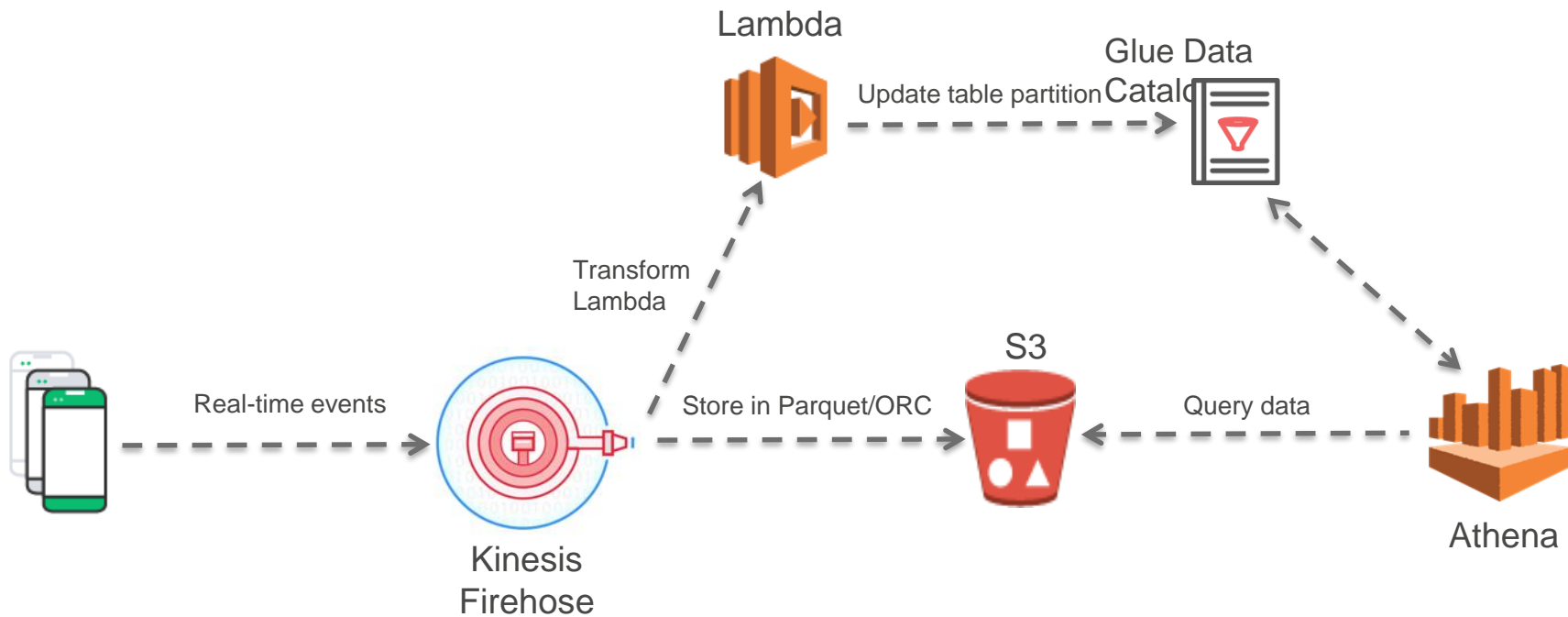




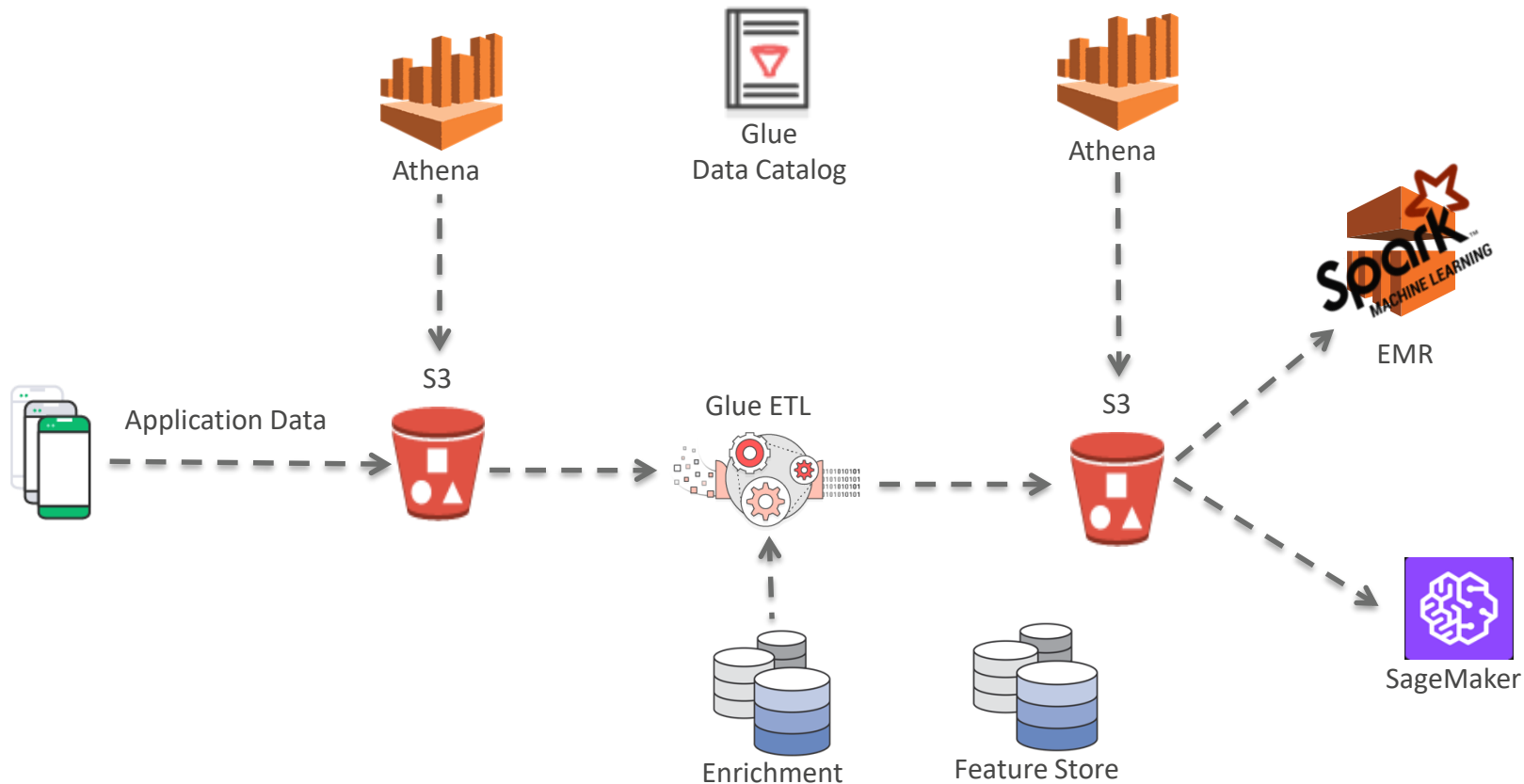
## 5. Query database backups



## 6. Time-series data



# 7. ML Exploration & Validation



# 8. Geospatial Data

## Geometry Data Types supported

- point
- line
- polygon
- multiline
- Multipolygon

Functions supported available in  
[Docs](#)

```
1 SELECT counties.name,  
2       COUNT(*) cnt  
3 FROM counties  
4 CROSS JOIN earthquakes  
5 WHERE ST_CONTAINS (counties.boundaryshape, ST_POINT(earthquakes.longitude, earthqu  
6 GROUP BY counties.name  
7 ORDER BY cnt DESC |
```

Use Ctrl + Enter to run query

Run Query

Save As

Format Query

New Query

...

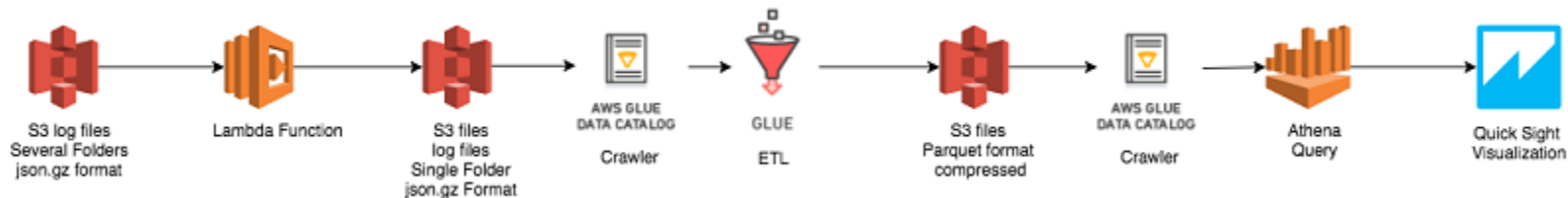


# AWS Big Data Blog

AWS Big Data Blog

## Visualize AWS Cloudtrail Logs using AWS Glue and Amazon QuickSight

by Luis Caro Perez | on 10 NOV 2017 | in Amazon QuickSight\*, AWS CloudTrail\*, AWS Glue\* | [Permalink](#) | [Comments](#) | [Share](#)



<https://aws.amazon.com/blogs/big-data/category/analytics/amazon-athena/>





## Query and Visualize AWS Cost and Usage Data Using Amazon Athena and Amazon QuickSight

by Androski Spicer | on 22 SEP 2017 | in Amazon Athena\*, Amazon QuickSight\* | [Permalink](#) | [Comments](#) | [Share](#)

## Analysis of Top-N DynamoDB Objects using Amazon Athena and Amazon QuickSight

by Rendy Oka | on 30 JUN 2017 | in Amazon Athena\*, Amazon DynamoDB\*, Amazon QuickSight\* | [Permalink](#) | [Comments](#) | [Share](#)

If you run an operation that continuously generates a large amount of data, you may want to know what kind of data is being inserted by your application. The ability to analyze data intake quickly can be very valuable for business units, such as operations and marketing. For many operations, it's important to see what [...]



## Build a Serverless Architecture to Analyze Amazon CloudFront Access Logs Using AWS Lambda, Amazon Athena, and Amazon Kinesis Analytics

by Rajeev Srinivasan and Sai Sriparasa | on 26 MAY 2017 | in Amazon Athena\*, Amazon CloudFront\*, Amazon Kinesis\*, AWS Lambda\* | [Permalink](#) | [Comments](#) | [Share](#)

Nowadays, the common framework concepts for creating high-level content delivery services like Amazon CloudFront. This type of front

## Audit Amazon Aurora Database Logs for Connections, Query Patterns, and More, using Amazon Athena and Amazon QuickSight

by Wendy Neu | on 20 NOV 2017 | in Amazon Athena\*, Amazon Aurora\*, Amazon QuickSight\*, Analytics\*, Aurora, Database\* | [Permalink](#) | [Comments](#) | [Share](#)

Amazon Aurora offers a high-performance advanced auditing feature that logs detailed database activity to the database audit logs. Amazon CloudWatch. If you are using Aurora 1.10.1 or greater, you can use advanced auditing to meet regulatory or compliance



## Analyzing VPC Flow Logs with Amazon Kinesis Firehose, Amazon Athena, and Amazon QuickSight

by Ian Robinson and Ben Snively | on 09 MAR 2017 | in Amazon Athena\*, Amazon Kinesis\*, Amazon QuickSight\* | [Permalink](#) | [Comments](#) | [Share](#)

Many business and operational processes require you to analyze large volumes of frequently updated data. Log analysis, for example, involves querying and visualizing large volumes of log data to identify behavioral patterns, understand application processing flow, and investigate and diagnose issues. VPC flow logs capture information about the IP traffic going to and from network [...]



[Read More](#)

# Agenda

1. Review of the year
2. Use-cases seen since launch
3. Athena, Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

# Athena, Redshift Spectrum & EMR



Amazon Redshift



Amazon EMR



Amazon Athena



Amazon  
S3

Expand your use-case to allow processing data from a S3-based Data lake





# Agenda

1. Review of the year
2. Use-cases seen since launch
3. How does Athena compare to Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

# Connecting with Athena

1. API
2. JDBC and ODBC driver
3. Console

# Connecting to Amazon Athena - API

## Asynchronous Query API

### StartQueryExecution

```
client.startQueryExecution({
  QueryString: 'SELECT * FROM table_name LIMIT 100',
  ResultConfiguration: { OutputLocation: 's3://bucket/output/' },
  EncryptionConfiguration: { EncryptionOption: 'SSE_S3' },
  QueryExecutionContext: { Database: 'default_db' }
}, (err, result) => {})
```

### GetQueryResults

```
client.getQueryResults({
  QueryExecutionId: '2ef5d590-025a-48ec-895e-6bedfe72bc95',
  MaxResults: 1000,
  NextToken: null
}, (err, data) => {})
```



# Agenda

1. Review of the year
2. Use-cases seen since launch
3. How does Athena compare to Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

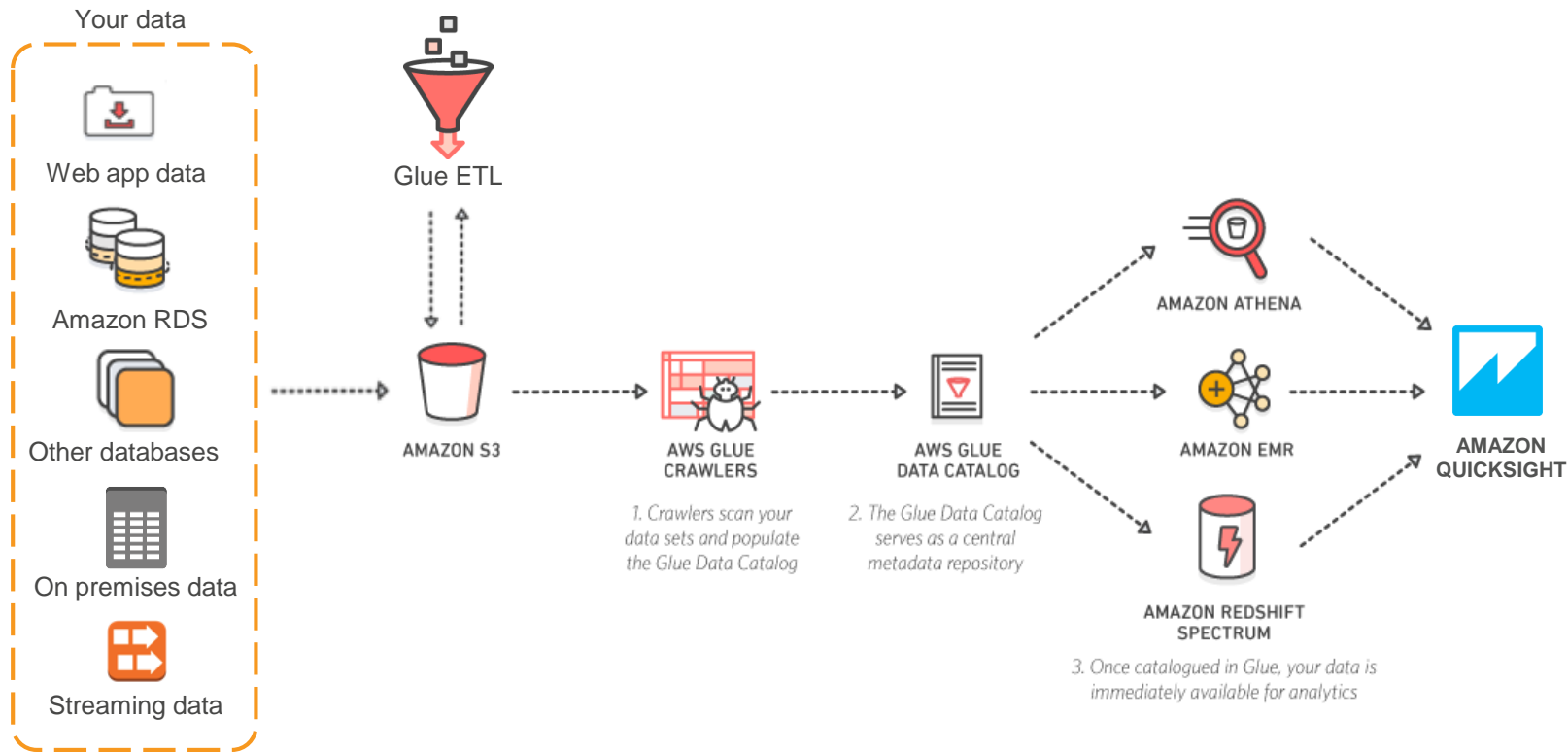
# What is the AWS Glue Data Catalog?

**Unified metadata repository** across relational databases, Amazon RDS, Amazon Redshift, and Amazon S3...with support for more coming soon!

- Get a **single view** into your data, no matter where it is stored
- Automatically **classify** your data in one central list that is **searchable**
- Track data evolution using **schema versioning**
- **Query** your data using Amazon Athena or Amazon Redshift Spectrum
- **Hive metastore compatible**; can be used as an external Hive Metastore for applications running on Amazon EMR



# Data Lake on Amazon S3 using AWS Glue

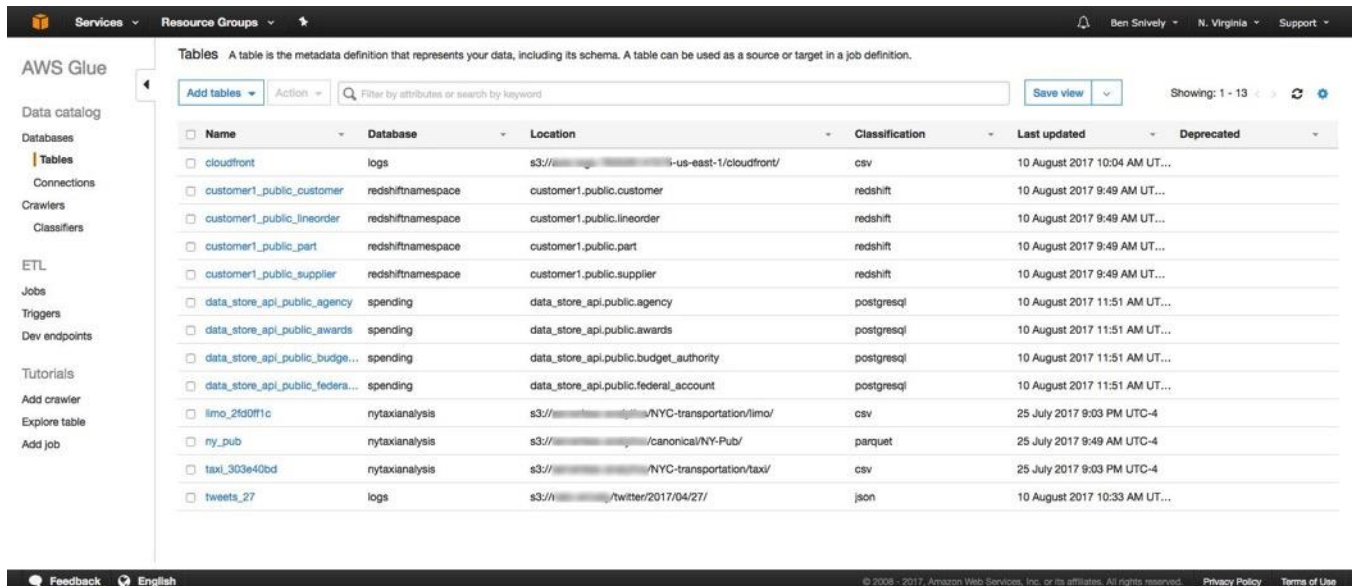


# What do I do to setup my Data Catalog

Its Easy!

1. Tell us where your data is
2. Tell us how often you want to check for updates

And you are done! **Your Data Catalog** is ready for search and querying



The screenshot displays the AWS Glue Data Catalog interface. On the left, a sidebar contains navigation links for 'Data catalog', 'Databases', 'Tables', 'Connections', 'Crawlers', 'Classifiers', 'ETL', 'Jobs', 'Triggers', 'Dev endpoints', 'Tutorials', 'Add crawler', 'Explore table', and 'Add job'. The main panel is titled 'Tables' and includes a description: 'A table is the metadata definition that represents your data, including its schema. A table can be used as a source or target in a job definition.' Below this, there is a search bar and a 'Filter by attributes or search by keyword' option. A table lists various data sources with columns for Name, Database, Location, Classification, Last updated, and Deprecated. The table contains 13 entries, including cloudfront, customer1\_public\_customer, customer1\_public\_lineorder, customer1\_public\_part, customer1\_public\_supplier, data\_store\_api\_public\_agency, data\_store\_api\_public\_awards, data\_store\_api\_public\_budge..., data\_store\_api\_public\_federa..., limo\_2f90f1c, ny\_pub, taxi\_303e40bd, and tweets\_27.

Name	Database	Location	Classification	Last updated	Deprecated
cloudfront	logs	s3://[redacted]-us-east-1/cloudfront/	csv	10 August 2017 10:04 AM UT...	
customer1_public_customer	redshiftnamespace	customer1.public.customer	redshift	10 August 2017 9:49 AM UT...	
customer1_public_lineorder	redshiftnamespace	customer1.public.lineorder	redshift	10 August 2017 9:49 AM UT...	
customer1_public_part	redshiftnamespace	customer1.public.part	redshift	10 August 2017 9:49 AM UT...	
customer1_public_supplier	redshiftnamespace	customer1.public.supplier	redshift	10 August 2017 9:49 AM UT...	
data_store_api_public_agency	spending	data_store_api.public.agency	postgresql	10 August 2017 11:51 AM UT...	
data_store_api_public_awards	spending	data_store_api.public.awards	postgresql	10 August 2017 11:51 AM UT...	
data_store_api_public_budge...	spending	data_store_api.public.budget_authority	postgresql	10 August 2017 11:51 AM UT...	
data_store_api_public_federa...	spending	data_store_api.public.federal_account	postgresql	10 August 2017 11:51 AM UT...	
limo_2f90f1c	nytaxianalysis	s3://[redacted]/NYC-transportation/limo/	csv	25 July 2017 9:03 PM UTC-4	
ny_pub	nytaxianalysis	s3://[redacted]/canonical/NY-Pub/	parquet	25 July 2017 9:49 AM UTC-4	
taxi_303e40bd	nytaxianalysis	s3://[redacted]/NYC-transportation/taxi/	csv	25 July 2017 9:03 PM UTC-4	
tweets_27	logs	s3://[redacted]/twitter/2017/04/27/	json	10 August 2017 10:33 AM UT...	

# What are crawlers?

Crawlers automatically build your Data Catalog and keep it in sync

- Scan your data stored in various data stores, extract metadata and data statistics, and add table definitions to your Data Catalog
  - Classify data using built-in and custom classifiers
  - You can write your own using Grok expressions
- Discover new data, extracts schema definitions
  - Detect schema changes and version tables
  - Detect Hive style partitions on Amazon S3
- Run ad hoc or on a schedule; serverless – only pay when crawler runs





# A table in the Glue Data Catalog

The screenshot displays the AWS Glue console interface. On the left, a navigation pane lists various services and actions. Orange brackets and lines are used to group these items into four categories: 'Table properties' (Tables, Connections, Crawlers, Classifiers), 'Data statistics' (Jobs, Triggers, Dev endpoints), 'Table schema' (Tutorials, Add crawler, Explore table, Add job), and 'Nested fields' (a bracket pointing to the schema details). The main content area shows the details for the 'simpletweets\_json' table, including its name, description, database, classification, location, connection, deprecated status, and last updated date. Below this, a table lists the schema columns: 'entities' (struct), 'id' (bigint), 'retweeted' (boolean), 'text' (string), and 'user' (struct). An orange arrow points from the 'user' column to a modal window titled 'user schema details', which shows a nested JSON structure for a user profile.

**Table properties**

**Data statistics**

**Table schema**

**Nested fields**

**Table details:**

- Name: simpletweets\_json
- Description: analytics
- Database: analytics
- Classification: json
- Location: s3://gluesampleddata/simpletweets.json
- Connection: No
- Deprecated: No
- Last updated: Thu Aug 10 16:25:24 GMT-700 2017

**Properties:**

Property	Value
sizeKey	456580
objectCount	1
UPDATED_BY_CRAWLER	S3Crawler
CrawlerSchemaSerializeVersion	1.0
recordCount	1001
averageRecordSize	456
CrawlerSchemaDeserializeVersion	1.0
compressionType	none
typeOfData	file

**Schema:**

Column name	Data type
1 entities	struct
2 id	bigint
3 retweeted	boolean
4 text	string
5 user	struct

**user schema details:**



```
STRUCT
  contributors_enabled: BOOLEAN
  description: STRING
  favourites_count: INT
  followers_count: INT
  friends_count: INT
  id: INT
  lang: STRING
  location: STRING
  name: STRING
  profile_background_tile: BOOLEAN
```

# Automatically detected partitions

12	errorcode	string	
13	region	string	Partition (0)
14	year	string	Partition (1)
15	month	string	Partition (2)
16	day	string	Partition (3)

Table  
partitions

## Automatically register available partitions

region				Showing: 1 - 3 < >	
region	year	month	day		
us-west-2	2016	02	18	<a href="#">View files</a> 	<a href="#">View properties</a>
us-west-2	2016	02	17	<a href="#">View files</a> 	<a href="#">View properties</a>
us-west-2	2016	02	16	<a href="#">View files</a> 	<a href="#">View properties</a>

# How is my data classified?

Glue applies a set of classifiers to the data as it scans it and adds the metadata as Tables to the Data Catalog.

A **classifier** recognizes the format of your data and generates a schema.

It returns a certainty number between 0.0 and 1.0 which helps Glue determine if there is a match

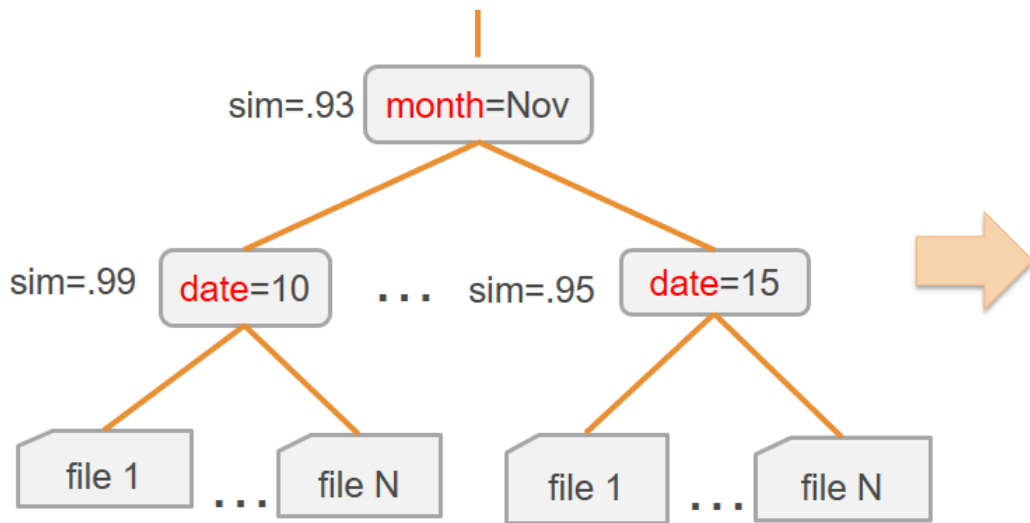
Glue has a list of in-build classifiers that are applied with every crawl. But you can **write your own!**

You can set up your crawler with an ordered set of classifiers. Crawlers invoke classifiers in the order they were provided until a match is found.



# How are partitions detected?

## S3 bucket hierarchy



## Table definition

Column	Type
<code>month</code>	str
<code>date</code>	str
<code>col 1</code>	int
	float
⋮	⋮

Estimate schema similarity among files at each level to handle semi-structured logs, schema evolution...



# How can I write my own classifiers?

You can write a custom classifier by providing a grok pattern and a classification string for the matched schema

A grok pattern is a named set of regular expressions (regex) that are used to match data one line at a time.

Example:

```
%{TIMESTAMP_ISO8601:timestamp}  
\[%{MESSAGEPREFIX:message_prefix}\]  
%{CRAWLERLOGLEVEL:loglevel} :  
%{GREEDYDATA:message}
```

## Classifier name

Id Crawler logs

## Classification

crawlerlogs

Describes the format of the data classified or a custom label.

## Grok pattern

%{TIMESTAMP\_ISO8601:timestamp} \[%{MESSAGEPREFIX:message\_prefix}\]

Built-in and custom named patterns used to parse your data into a structured schema. For more information, see the [list of built-in patterns](#).

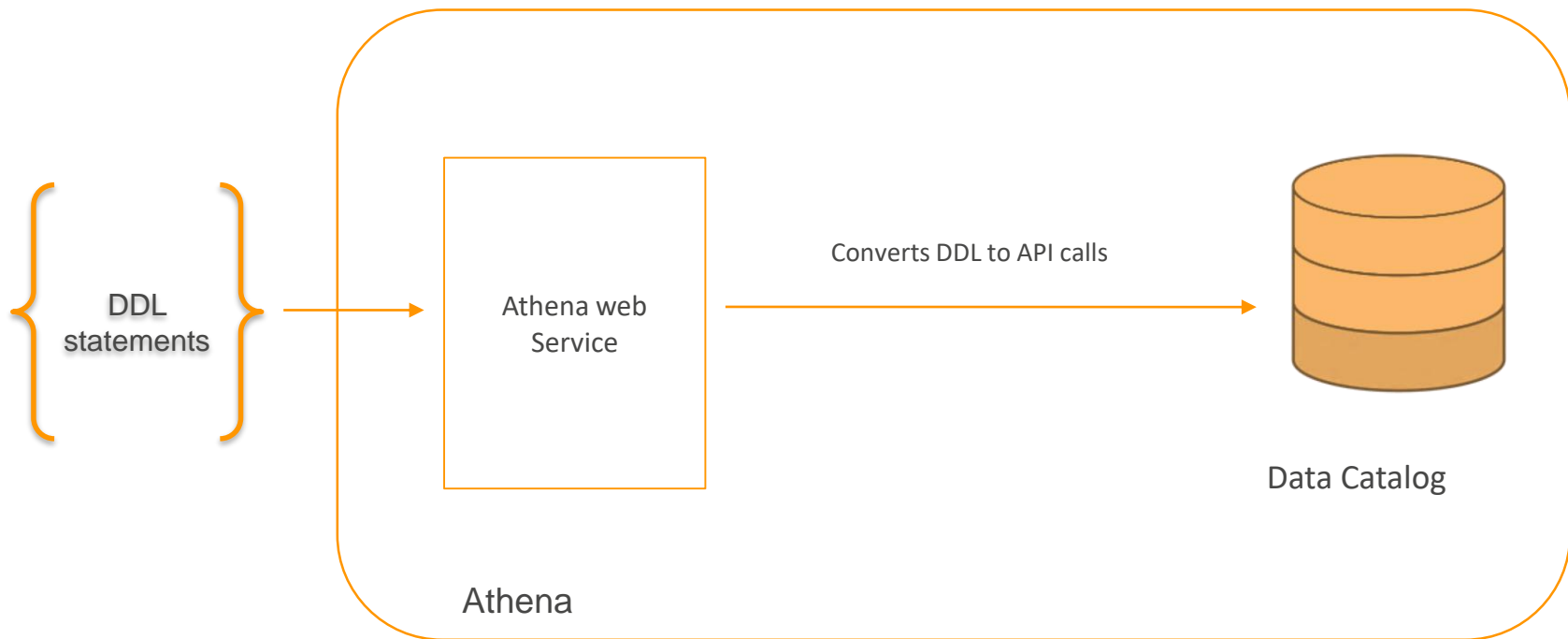
## Custom patterns

```
1 CRAWLERLOGLEVEL (BENCHMARK|ERROR|WARN|INFO|TRACE)  
2 MESSAGEPREFIX .*-.*-.*-.*-.*  
3
```

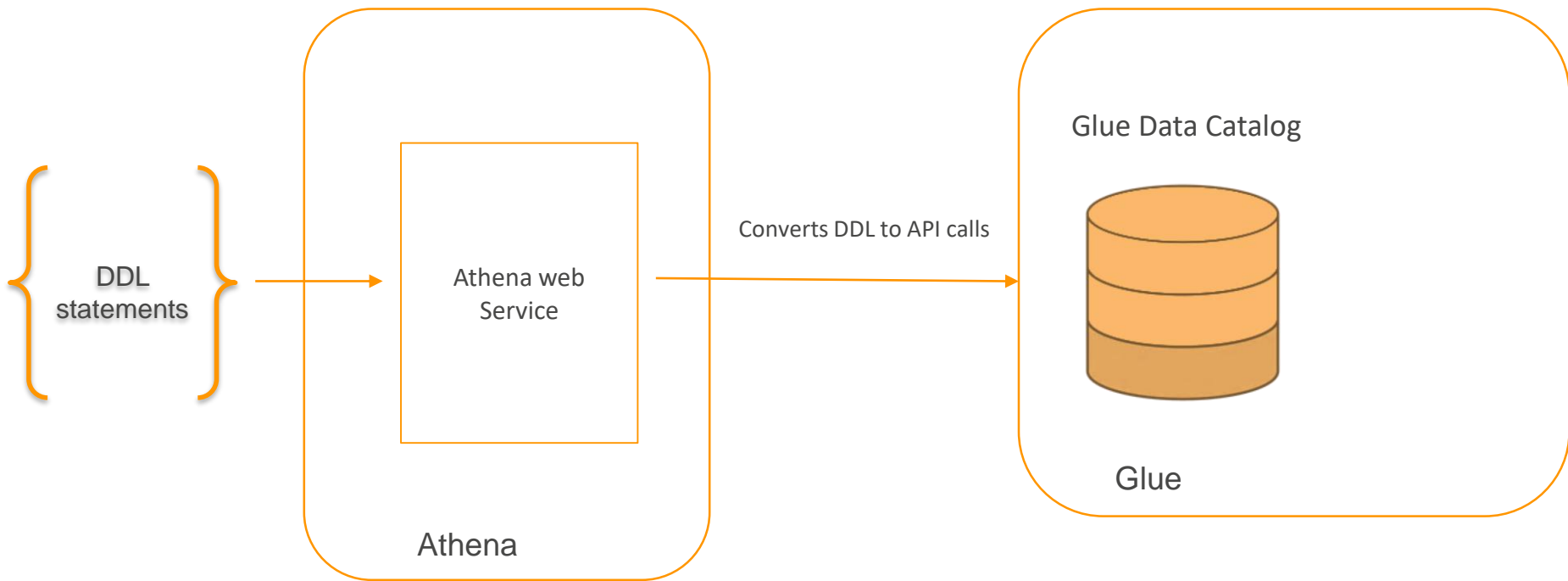
Optional custom building blocks for the grok pattern.



# Athena Data Catalog or Glue Data Catalog



# Upgrading to Glue Data Catalog



# Upgrading is a simple process

1. Add Glue-specific policies to your existing Athena policies
2. Add permission to **Glue:ImportCatalogToGlue**
3. Upgrade the catalog



# Benefits of upgrading

1. You can now share metadata between EMR (Hive, Spark and Presto), Athena and Redshift Spectrum
2. You can use the Glue API to directly run DDL at high concurrency (table creation, partitions)
3. Use the crawler for automatic recognition of schema and partitioning
4. Use Glue to ETL data and query in Athena



# Agenda

1. Review of the year
2. Use-cases seen since launch
3. How does Athena compare to Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

# Creating Databases and Tables

1. Databases are logical collection of Tables
2. Database and Table service limits can be raised
3. Many ways to write DDL Statement
  1. Use the Hive DDL statement directly from the console
  2. Use the Athena API
  3. Use the JDBC/ODBC
  4. Use the Glue Crawlers
  5. Use the Glue API to create Tables
  6. If using the Glue Data Catalog, You can also share metadata between EMR running Spark/Hive/Presto and Athena



# Creating Database and Tables

```
CREATE EXTERNAL TABLE access_logs
```

```
(  
  ip_address String,  
  request_time Timestamp,
```

```
)  
ROW FORMAT DELIMITED FIELDS TERMINATED BY  
'\t'
```

```
STORED AS TEXTFILE
```

```
LOCATION 's3://mydatalogs/data/'
```



**Data Catalog**

Table name: **access\_logs**  
Location: s3://mydatalogs/data  
Serde information:

An orange line starts from the 'Data Catalog' section, goes left, then down, and finally branches into four horizontal lines pointing to the S3 object list.

s3://mydatalogs/data/object1  
s3://mydatalogs/data/object2  
s3://mydatalogs/data/object3  
s3://mydatalogs/data/object4



# Schema on Read Versus Schema on Write

## Schema on Read

- Store raw data
- Schema is applied to the data

Good for query flexibility and control over data access

## Schema on write

- Create the schema
- Fit the data to the schema

Good for schema enforcement and performance optimization

# SerDes and Compression formats

1. Regex
2. GROK
3. JSON
4. OpenCSV
5. TSV
6. Parquet
7. ORC
8. AVRO
9. Geospatial

Algorithm	Splittable?	Compression ratio	Compress + Decompress speed
Gzip (DEFLATE)	No	High	Medium
bzip2	Yes	Very high	Slow
LZO	No	Low	Fast
Snappy	No	Low	Very fast

# Schema Evolution

Expected Type of Schema Update	Summary	CSV (with and without headers) and TSV	JSON	AVRO	PARQUET: Read by Name (default)	PARQUET: Read by Index	ORC: Read by Index (default)	ORC: Read by Name
Rename columns	Store your data in CSV and TSV, or in ORC and Parquet if they are read by index.	Y	N	N	N	Y	Y	N
Add columns at the beginning or in the middle of the table	Store your data in JSON, AVRO, or in Parquet and ORC if they are read by name. Do not use CSV and TSV.	N	Y	Y	Y	N	N	Y
Add columns at the end of the table	Store your data in CSV or TSV, JSON, AVRO, and in ORC and Parquet if they are read by name.	Y	Y	Y	Y	N	N	Y
Remove columns	Store your data in JSON, AVRO, or Parquet and ORC, if they are read by name. Do not use CSV and TSV.	N	Y	Y	Y	N	N	Y
Reorder columns	Store your data in AVRO, JSON or ORC and Parquet if they are read by name.	N	Y	Y	Y	N	N	Y
Change a column's data type	Store your data in any format, but test your query in Athena to make sure the data types are compatible.	Y	Y	Y	Y	Y	Y	Y

<https://docs.aws.amazon.com/athena/latest/ug/handling-schema-updates-chapter.html>



# Agenda

1. Review of the year
2. Use-cases seen since launch
3. How does Athena compare to Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations



# Data Partitioning

- Read only data the query needs
- Reduce amount of data scanned
  - decrease query completion time
  - Reduce query cost
- Define Partitions at the time of creating tables.
- Separates data files by any column
- Partitions are virtual columns that you can reference in your query
- Athena prefers Hive-style partitioning
- Works on both text and columnar data



# Partitioning of data has cost advantages

Query	Non- Partitioned Table		Cost	Partitioned table		Cost	Savings
	Run time	Data scanned		Run time	Data scanned		
SELECT count(*) FROM lineitem WHERE l_shipdate = '1996-09-01'	9.71 seconds	74.1 GB	\$0.36	2.16 seconds	29.06 MB	\$0.0001	99% cheaper 77% faster
SELECT count(*) FROM lineitem WHERE l_shipdate >= '1996-09-01' AND l_shipdate < '1996-10-01'	10.41 seconds	74.1 GB	\$0.36	2.73 seconds	871.39 MB	\$0.004	98% cheaper 73% faster



# But it also has an overhead

Query	Non- Partitioned Table		Cost	Partitioned table		Cost	Savings
	Run time	Data scanned		Run time	Data scanned		
<code>SELECT count(*) FROM lineitem;</code>	8.4 seconds	74.1 GB	\$0.36	10.65 seconds	74.1 GB	\$0.36	27% slower



# Layout of data for partitioned tables

```
CREATE EXTERNAL TABLE access_logs
```

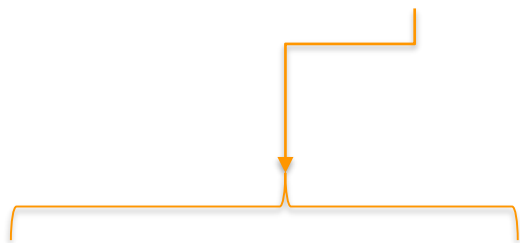
```
(  
  ip_address String,  
  request_time Timestamp  
)
```

```
PARTITIONED by
```

```
(string year,  
string month)
```

```
STORED AS PARQUET
```

```
LOCATION 's3://yourBucket/pathToData/
```



```
s3://yourBucket/pathToData/<PARTITION_COLUMN_NAME1>=<VALUE>/<PARTITION_COLUMN_NAME2>=<VALUE>/
```



# Layout of data for partitioned tables

```
CREATE EXTERNAL TABLE access_logs  
(  
  ip_address String,  
  request_time Timestamp  
)  
PARTITIONED BY  
(string year,  
 string month)  
STORED AS PARQUET  
LOCATION 's3://yourBucket/pathToTable/
```



`s3://yourBucket/pathToTable/year=2017/month=11/`

Alter Table `access_logs`

Add Partition

`(year='2017',  
month='11')`

location 's3://yourBucket/pathToTable/year=2017/month=11/

MSCK Repair Table

GLUE API

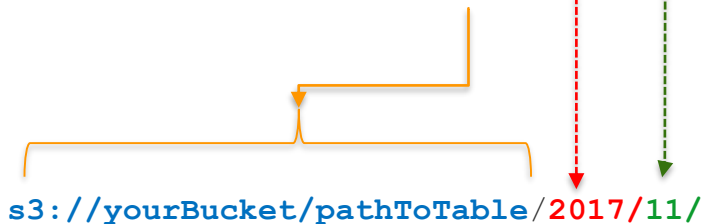


# Layout of data for partitioned tables

```
CREATE EXTERNAL TABLE access_logs
```

```
(  
  ip_address String,  
  request_time Timestamp,
```

```
)  
PARTITIONED BY  
(string year,  
 string month)  
STORED AS PARQUET  
LOCATION 's3://yourBucket/pathToTable/
```



```
Alter Table access_logs
```

```
Add Partition
```

```
(year='2017',  
 month='11')
```

```
location 's3://yourBucket/pathToTable/2017/11/'
```

```
MSCK Repair Table
```

```
GLUE API
```



# Partition loading and S3 prefixes

1

`s3://yourBucket/pathToTable/year=2017/month=11/` - MSCK REPAIR TABLE

2

`s3://yourBucket/pathToTable/2017/11/`

```
ALTER TABLE access_log ADD PARTITION (year='2017',month='11')  
location 's3://yourBucket/pathToTable/2017/11/'
```

3

`s3://yourBucket/pathToTable/232a-2013-26-05/hhdsy1129s/`

```
ALTER TABLE access_log ADD PARTITION (year='2017',month='11')  
location 's3://yourBucket/pathToTable/232a-2013-26-05/hhdsy1129s/'
```



# Loading Partitions

- Schedule a Glue Crawler to automatically detect partitions.
- Use the Glue Data Catalog API/CLI – This is preferred for high concurrency
- Use Alter Table Add Partition – You can add 99 partitions at the same time.
- Use MSCK – MSCK is an time consuming operation
- Use Lambda – either on a schedule or based on event
- Pre-populate at the start/end of the day – these are just key-value lookups



# Choose partitions to get performance even on text data

1. Columns that are commonly used as filters (e.g. date, category)
2. You can partition by any number of columns
3. You can partition on any column not just date
4. {Partition key 1, Partition key 2, Partition 3} → unique location in S3
5. Look at your query patterns – What data do you want to query and what do you want to filter out



# Agenda

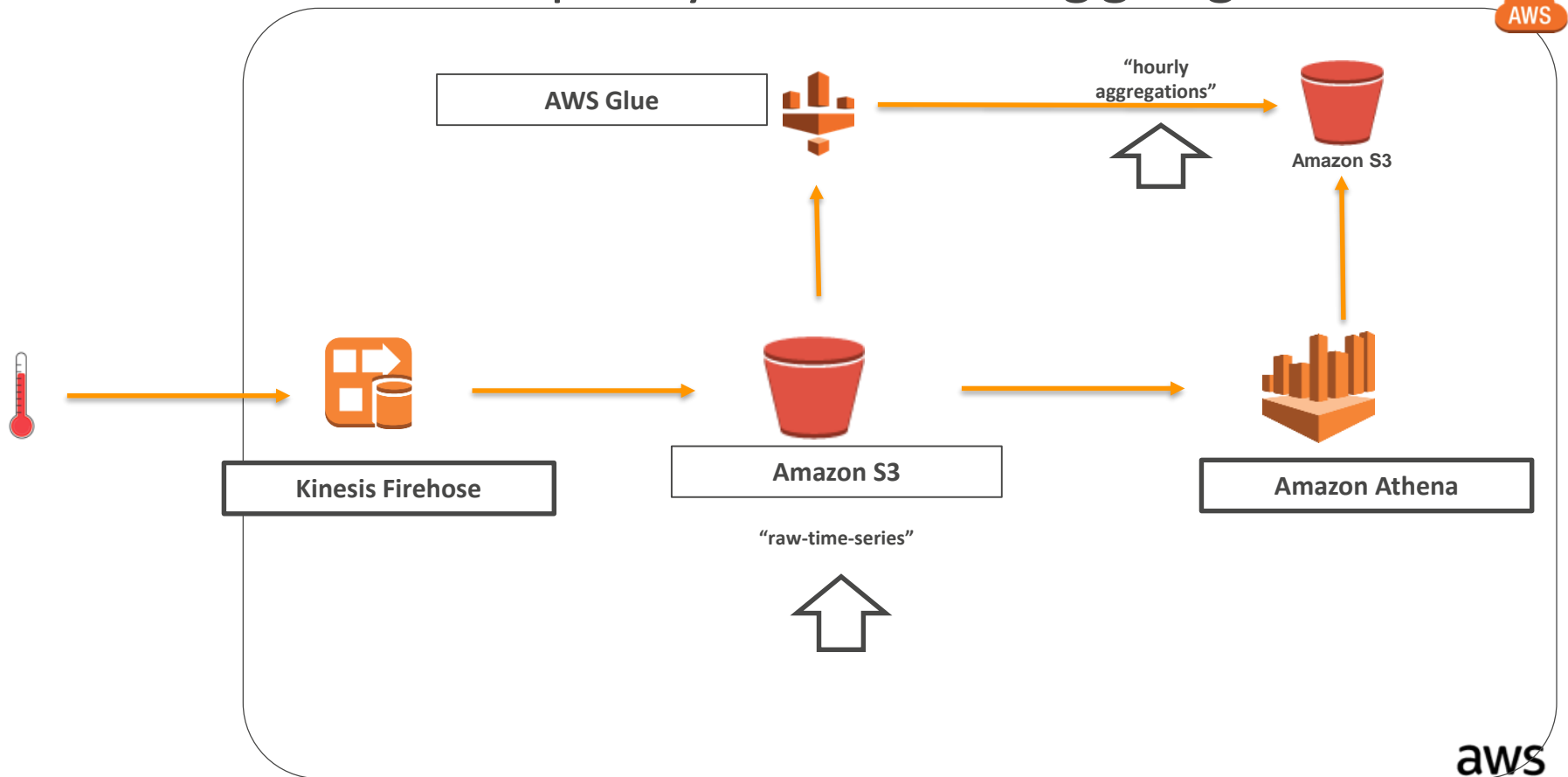
1. Review of the year
2. Use-cases seen since launch
3. How does Athena compare to Redshift Spectrum, and EMR
4. Connecting with Amazon Athena
5. Glue Data Catalog
6. Creating Tables
7. Partitioning data
8. Running queries and performance optimizations

# Query Performance – bigger files help

Query	Number of files	Run time
<code>SELECT count(*) FROM lineitem</code>	5000 files	8.4 seconds
<code>SELECT count(*) FROM lineitem</code>	1 file	2.31 seconds
Speedup		72% faster



# Allow users to query raw and aggregates



# Query performance- Columnar Formats

Dataset	Size on S3	Query run time	Data Scanned	Cost
Text	1.15TB	236 seconds	1.15 TB	\$5.75
Columnar (same data)	130GB	6.78 seconds	2.51 <u>GB</u>	\$0.013
Savings/Speed up	87% less with Columnar	34 x	99% less data scanned	99.7% savings

# Query Performance – Limit Results

## Example:

Dataset: 7.25 GB table, uncompressed, text format, ~60M rows

Query	Run time
<code>SELECT * FROM lineitem ORDER BY l_shipdate</code>	528 seconds
<code>SELECT * FROM lineitem ORDER BY l_shipdate LIMIT 10000</code>	11.15 seconds
Speedup	98% faster

# Query Performance – Join Reordering

Dataset: 74 GB total data, uncompressed, text format, ~602M rows

Query	Run time
<code>SELECT count(*) FROM lineitem, part WHERE lineitem.l_partkey = part.p_partkey</code>	22.81 seconds
<code>SELECT count(*) FROM part, lineitem WHERE lineitem.l_partkey = part.p_partkey</code>	10.71 seconds
Savings / Speedup	~53% speed up

Keep the larger Table on the Left Side of the join



# Query Performance – Regex better than Like

## Example:

Dataset: 74 GB table, uncompressed, text format, ~600M rows

Query	Run time
<code>SELECT count(*) FROM lineitem WHERE l_comment LIKE '%wake%' OR l_comment LIKE '%regular%' OR l_comment LIKE '%express%' OR l_comment LIKE '%sleep%' OR l_comment LIKE '%hello%'</code>	20.56 seconds
<code>SELECT count(*) FROM lineitem WHERE regexp_like(l_comment, 'wake regular express sleep hello')</code>	15.87 seconds
Speedup	17% faster





THANK YOU!