

ABD339

# AWS re:INVENT

## Deep Dive and Best Practices for Amazon Athena

Raúl Rentería, OLX Brazil, Data Engineering & Services Team  
Abhishek Sinha, Senior Product Manager, Amazon Athena

AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



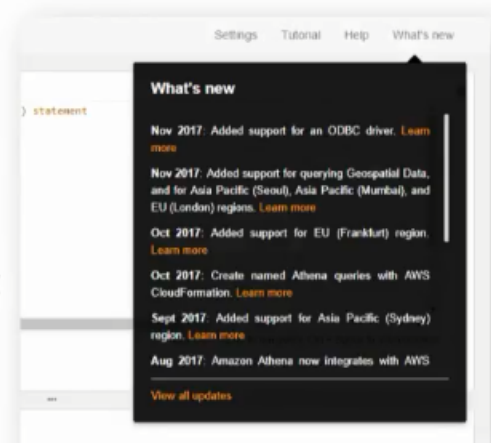
Amazon Athena is an interactive query service that enables you to process data directly from Amazon S3 without the need for infrastructure. Since its launch at re:invent 2016, several organizations have adopted Athena as the central tool to process all their data. In this talk, we dive deep into the most common use cases, including working with other AWS services. We review the best practices for creating tables and partitions and performance optimizations. We also dive into how Athena handles security, authorization, and authentication. Lastly, we hear from a customer who has reduced costs and improved time to market by deploying Athena across their organization.

## Agenda

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

# We launched exactly a year ago !

1. Rest API with support for SDK in most languages
2. AWS CloudTrail Support
3. JDBC and ODBC drivers
4. Launched in 11 AWS regions
5. Added support for AVRO and Geospatial data and Querying using GROK filter
6. Support for processing encrypted data using S3-SSE and CSE with keys in AWS KMS
7. Integration with AWS Glue
8. AWS CloudFormation support for creating Named Queries
9. Audit logging via AWS CloudTrail
10. Integration with S3 Inventory and EC2 Systems manager



re:Invent  
Inventory

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

## Amazon Athena

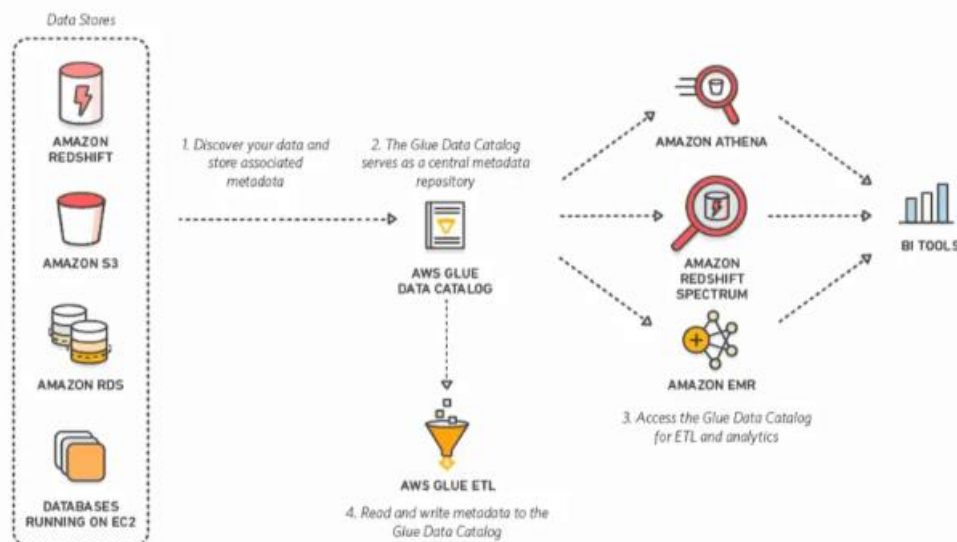


## Integration with several partners



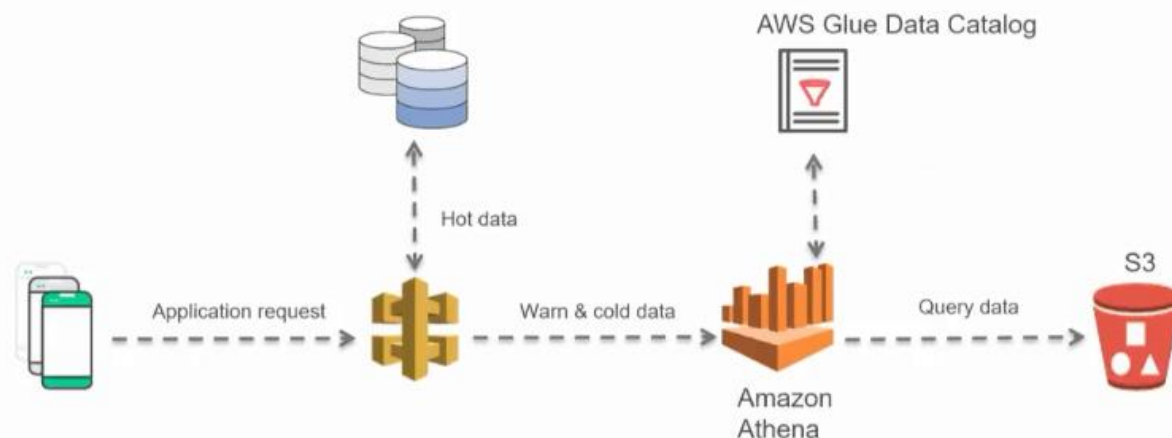
1. Review of the year
2. Use cases seen since launch
3. Amazon Athena, Amazon Redshift Spectrum, and Amazon EMR
4. Connecting with Amazon Athena
5. Creating Tables
6. AWS Glue Data Catalog
7. Partitioning data
8. Running queries and performance optimizations
9. OLX case Study

## Use Case 1: Query engine for data lake



Athena is being used as a query layer on a data lake.

## 2: Embed SQL to allow direct access to data

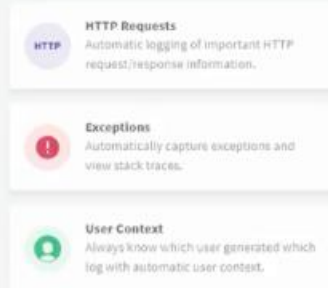
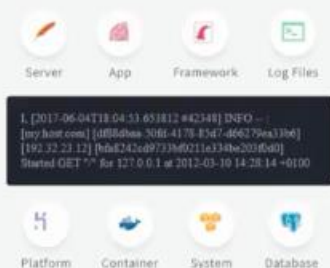


SaaS Customers are also using Athena as a SQL tool behind an API Gateway to expose the dataset that they have to external customers for direct querying. The customers simply push their data into S3 using the SaaS application.

## 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.

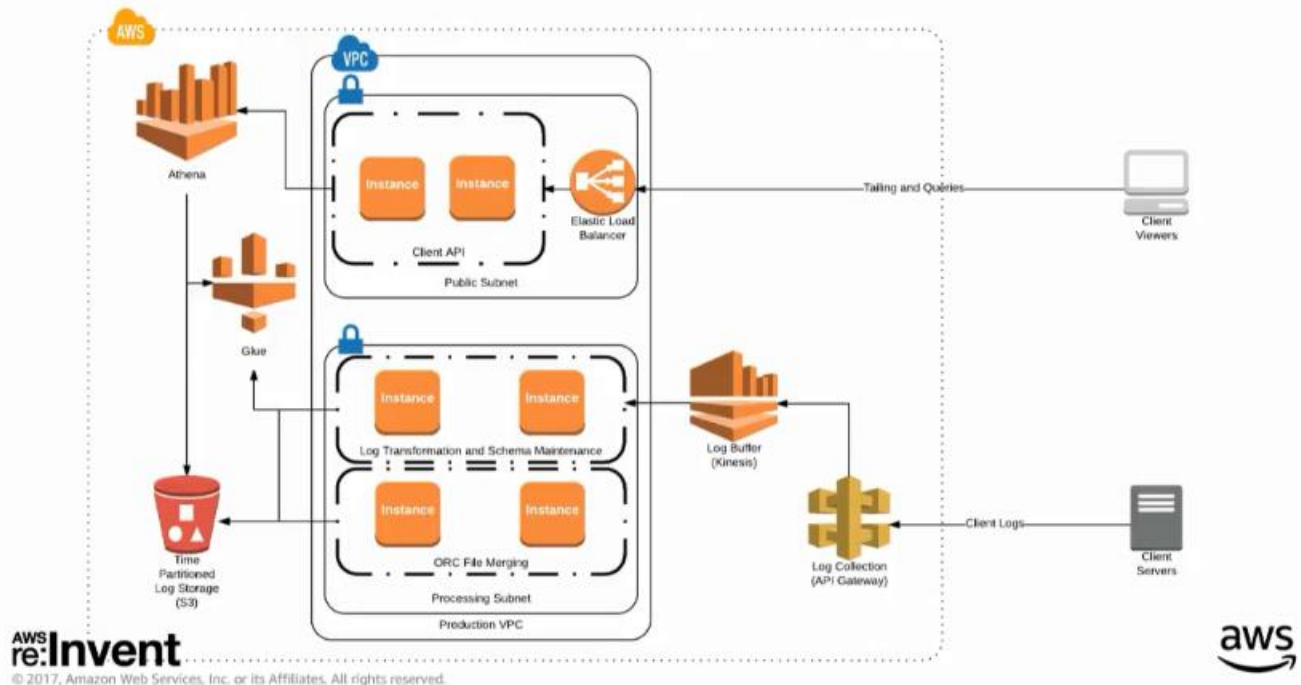


**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.







Timber.io allows customers to put their logs into S3 for log aggregation and log enrichment, their architecture shows that their API is wrapped around Athena. The bottom layer is where they are collecting logs from customer applications via an API Gateway (Log Collection) and Kinesis Streams (Log Buffer).

They take these logs and push them into S3 using some form of ETL to push the logs into ORC format, partition the data, then they allow their customers to query these logs. They allow customers to use a much simpler language in SQL by trapping the query and decomposing the query into smaller Athena queries and then directly talk to the Athena API. Athena scales along with them as their customers grow.

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

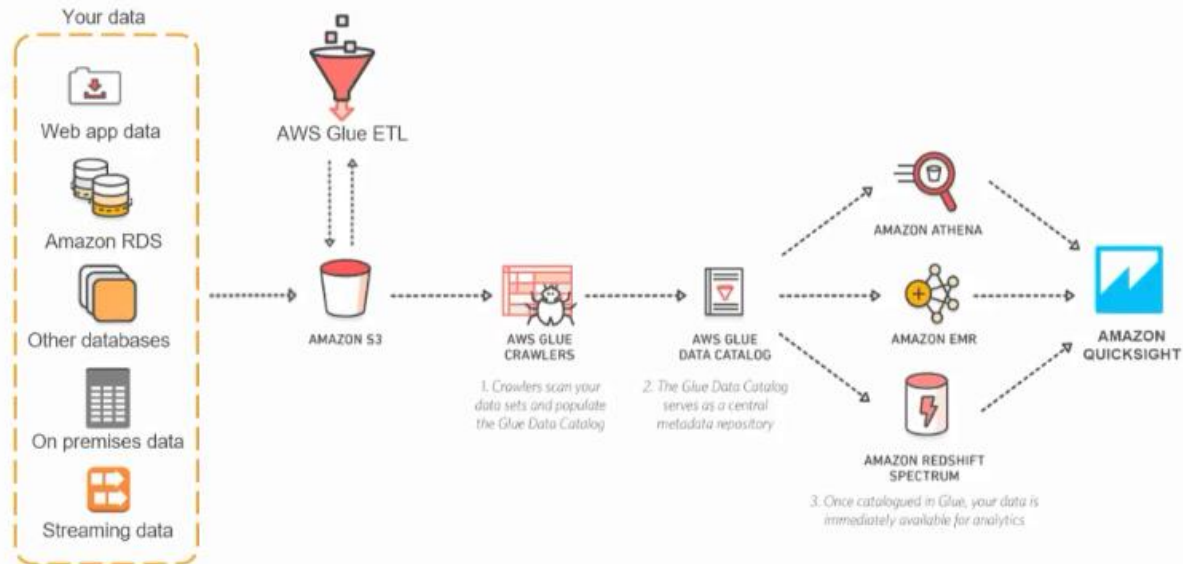


- ETL
- Data Wrangling
- Machine Learning
- Interactive Querying
- Data Science
- Hive Metastore



- **AWS Glue ETL** and **EMR** for ETL
- **Amazon Athena**, **EMR** and **Amazon Redshift** for Interactive querying
- **EMR** for ML
- **AWS Glue Data Catalog** Hosted Metadata Catalog

# Data Lake on S3

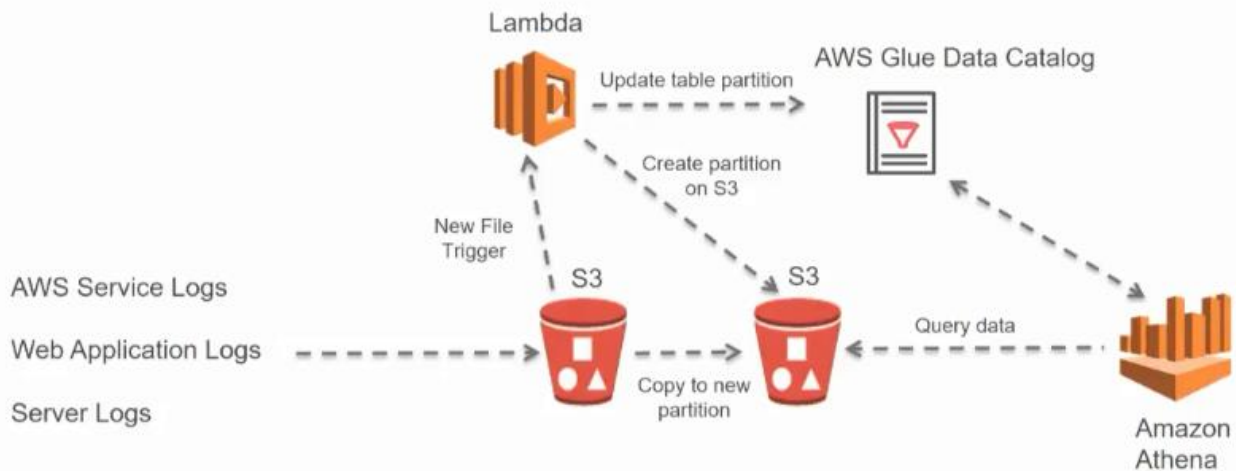


**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## 4: Querying Logs

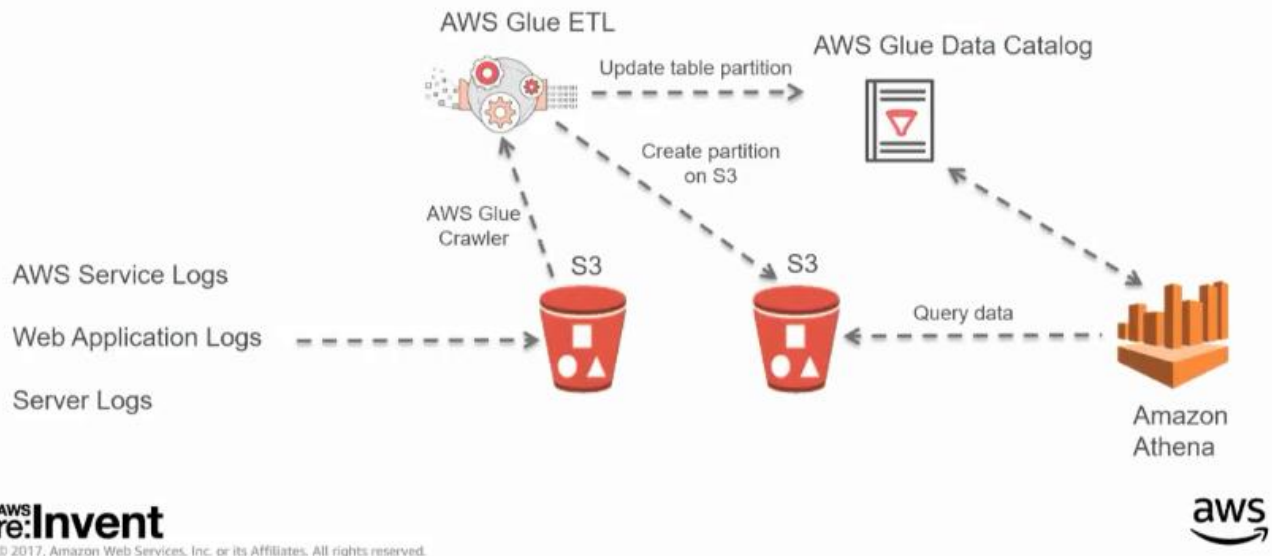


**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## 4a. Querying logs with ETL



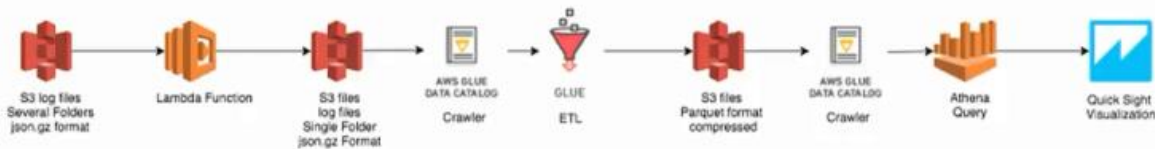
Athena does not charge you for creating tables and adding partitions,

## 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



This shows a bunch of CF templates having the most common queries you will be writing with Athena, you can see the entire pipeline and code for some use cases as below



### 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

If you've ever wondered if a serverless alternative existed for consuming and querying your AWS Cost and Usage report data, then wonder no more. The answer is yes, and this post both introduces you to that solution and illustrates the simplicity and effortlessness of deploying it. This solution leverages AWS serverless technologies Amazon Athena, AWS [...]

[Read More](#)



## 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 [...]

[Read More](#)



## 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 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, it's common for a web server to be fronted by a global content delivery service, like Amazon CloudFront. This type of front end accelerates delivery of websites, APIs, media content, and other web assets to provide a better experience to users across the globe. The insights gained by analysis of Amazon CloudFront access logs [...]

[Read More](#)



## 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, it's common for a web server to be fronted by a global content delivery service, like Amazon CloudFront. This type of front end accelerates delivery of websites, APIs, media content, and other web assets to provide a better experience to users across the globe. The insights gained by analysis of Amazon CloudFront access logs [...]




## 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 requirements by capturing eligible events like tables queried, queries issued, and connections and disconnections. You can [...]

[Read More](#)






**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 Remy 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 know what...



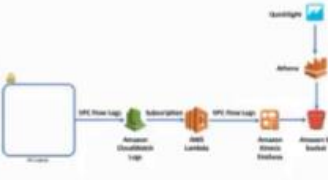
**Build a Serverless Architecture to Analyze Amazon CloudFront Access Logs Using AWS Lambda, Amazon Athena, and Amazon Kinesis Analytics**  
by Rajeev Srinivasan and Sai Sriparasa | on 26 MAY 2017 | in Amazon Athena\*, Amazon CloudFront\*, Amazon Kinesis\*, AWS Lambda\* | Permalink | Comments | Share

Main article: [AWS Lambda Functions for analyzing Amazon CloudFront access logs](#)



**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're 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 time, and investigate and diagnose issues. VPC flow logs capture information about the IP traffic going to and from network [...]

[Read More](#)

## The Airbnb 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

**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

**StreamAlert - Serverless, Realtime Data Analysis Framework**

Build | Deploy | Monitor

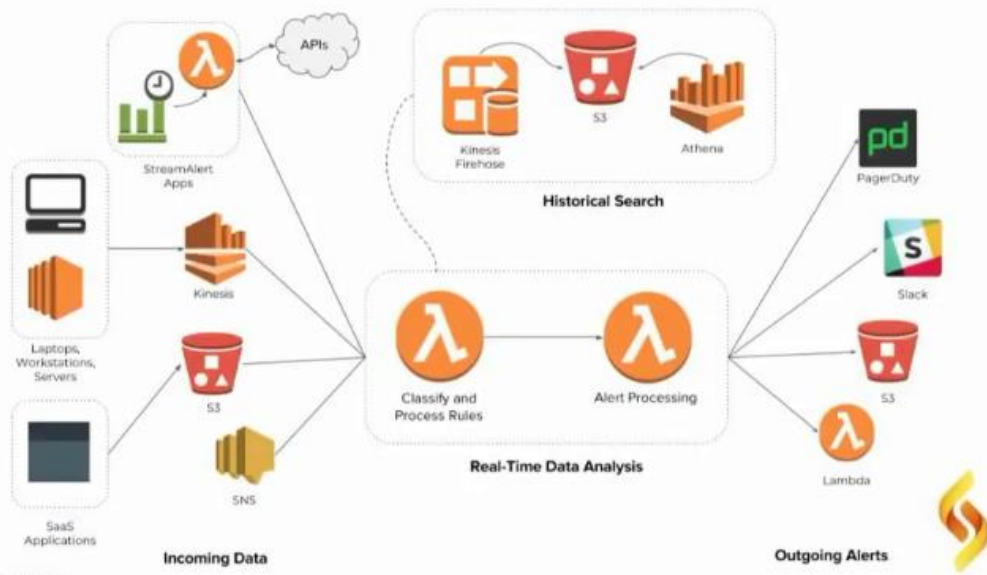


**StreamAlert**

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



# Overall Architecture

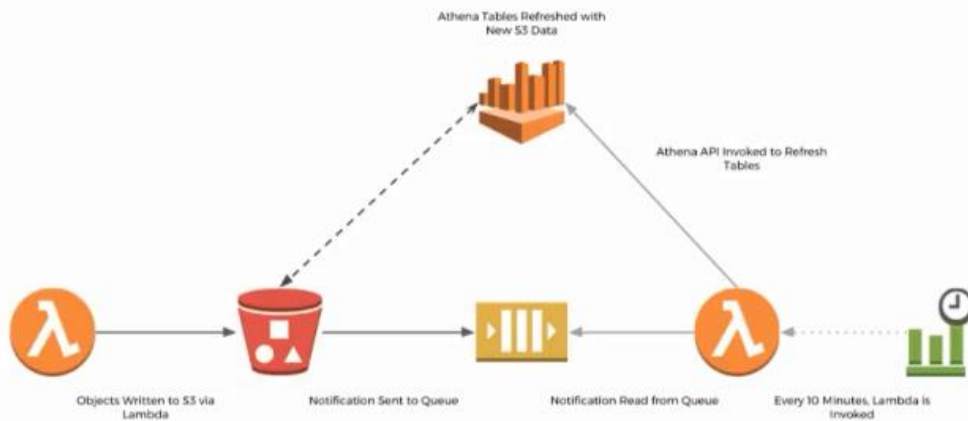


AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## How is Amazon Athena used in StreamAlert



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## 5. Query database backups

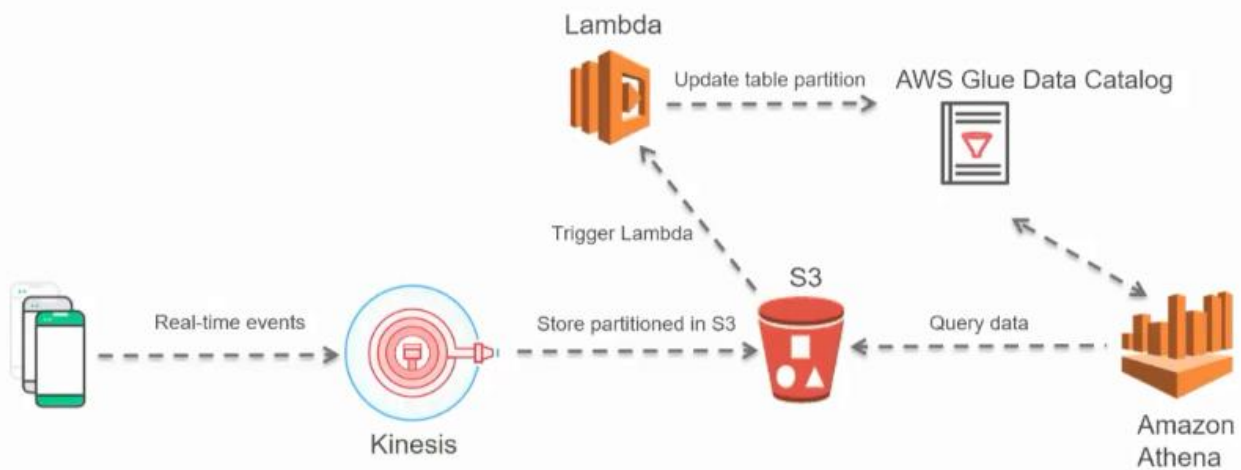


**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## 6. Time-series data



**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## 7. 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 |
```

Run Query

Save As

Format Query

New Query

Use Ctrl + Enter to run query

**AWS**  
**re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

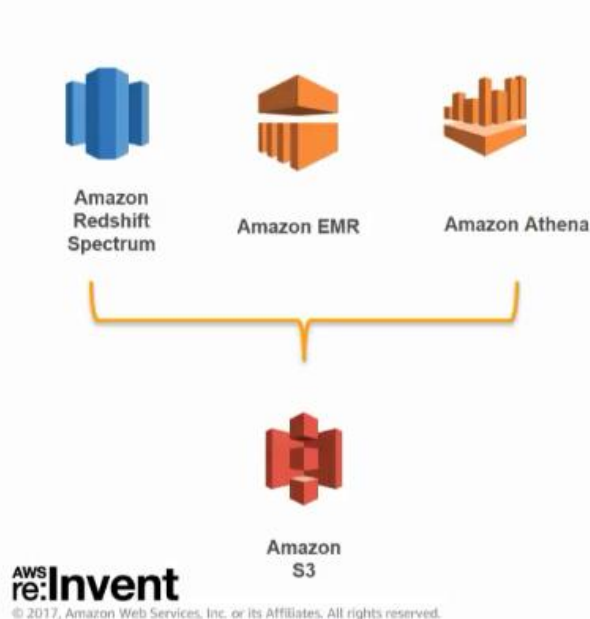


## Agenda

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



## Amazon Athena, Amazon Redshift Spectrum & EMR



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. Amazon Athena, Amazon Redshift Spectrum, and Amazon EMR
4. **Connecting with Amazon Athena**
5. Creating Tables
6. AWS Glue Data Catalog
7. Partitioning data
8. Running queries and performance optimizations
9. OLX case study

## Connecting with Amazon 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) => {})
```

The **Athena API is asynchronous**, this means that there are 2 key APIs. The **startQueryExecution API** where you get the **queryId** back, the **GetQueryResults API** which you pass the **queryId** to and get back the results. We have seen a lot of customers use this to build event driven pipeline where data in S3 is given to Athena to run queries on by using a Lambda function creates a table, loads the data into the table, starts the query and to also get the results and store in another S3 bucket or DynamoDB.

## Agenda

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

# Creating Databases and Tables

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

The Athena flow is basically about **creating tables**, **creating partitions**, and **running queries**. In the background, Athena use **Hive to run your DDL statements** and **Presto to run your SQL queries** but you get to interact with it through the Athena API. To create tables in Athena, you can use the console to directly write a Hive DDL statement, or you can directly use the **Athena API** and give it a Hive DDL statement to create a table, or you can use the JDBC and ODBC driver.

The **Glue Catalog is a shared metadata catalog** that can share metadata between **EMR cluster running Presto, Spark or Hive, Redshift Spectrum, and Athena**. Once we have created our table in any one of the above systems with the glue Catalog, all the 3 engines can see and use the table. This means that we can directly use the **Glue API** to create the tables that we want to use in Athena. **Glue Crawlers** are things that run on your data in **S3** and automatically infer table schema and create a table or partition for you.

## 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:

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



**AWS**  
**re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Athena only works on the concept of external tables, this means that the location of your stored metadata is different from the location where your data is stored.

# Schema on Read Versus Schema on Write

## Schema on write

- Create the schema
- Fit the data to the schema

Good for repetition and performance

## Schema on Read

- Store raw data
- Schema is applied to the data

Good for exploration, and experimentation

# 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

**SerDes means SerializerDeserializer**, it basically tells Athena what is the kind of underlying data that we have like Regex, **Grok** (specific Regex patterns), JSON etc. **Athena** is faster if you use a columnar format like **Parquet** and **ORC**. Athena also supports some compression formats, if you compress your data, Athena charges you by the amount of data scanned in S3, your cost goes down if you have compressed data in S3 into a format like **Parquet**.

## Agenda

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

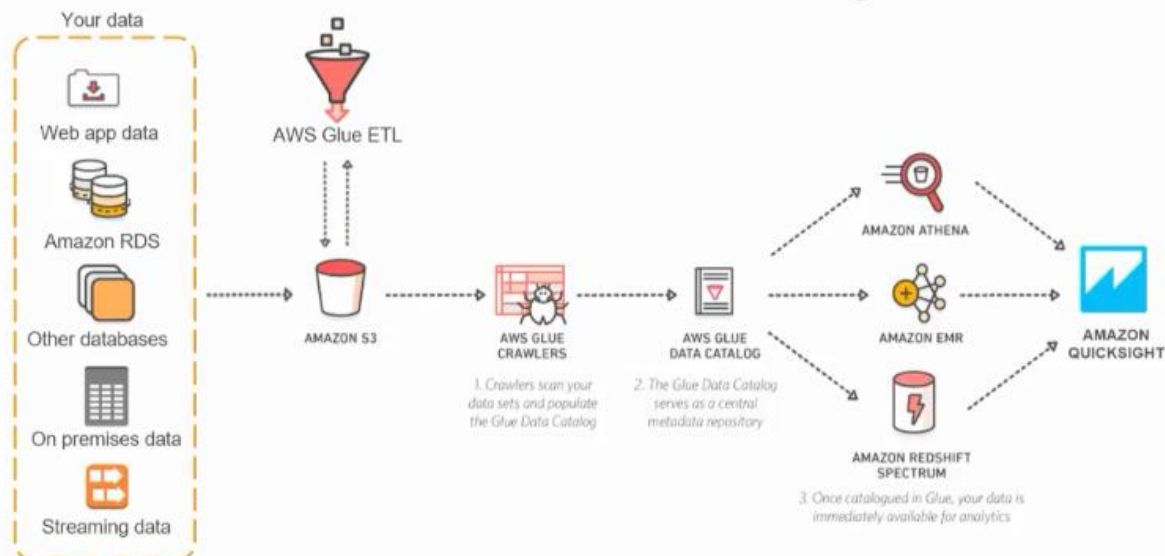


# What is the AWS Glue Data Catalog?

**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





# Automatically detected partitions

12	emrccode	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	year	month	day		
us-west-2	2016	02	16	<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>

Showing: 1 - 3

Partitioning helps to reduce the amount of data being scanned by Athena. Partition is like using a virtual column with a WHERE clause, this helps to restrict your query to a smaller dataset and scan less data, pay less, become faster.

## 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

# How is my data classified?

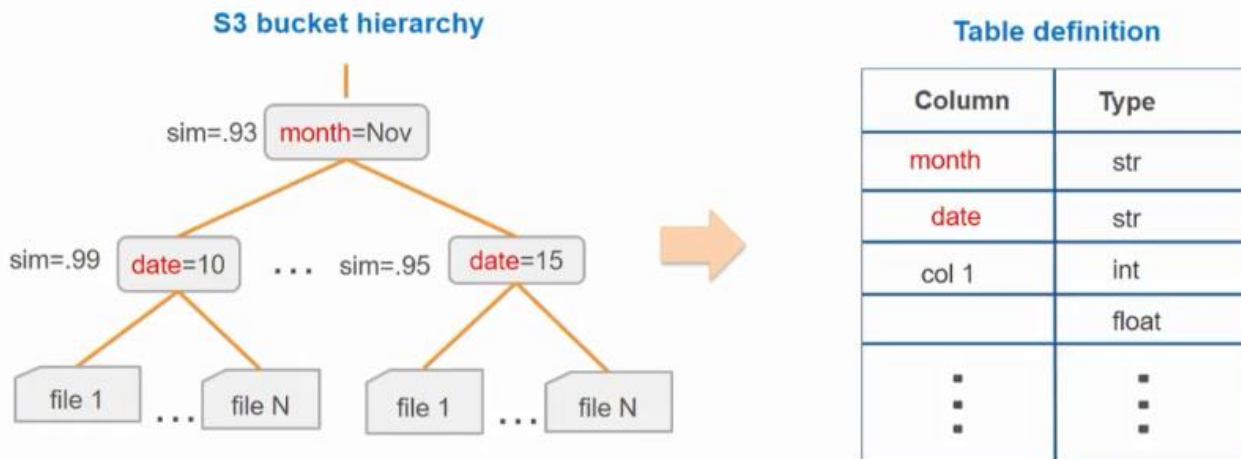
AWS Glue applies a set of classifiers to the data as it scans it and adds the metadata as Tables to the Data Catalog.

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 AWS Glue determine if there is a match

AWS 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?



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**

**Classification**

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

**Grok pattern**

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	

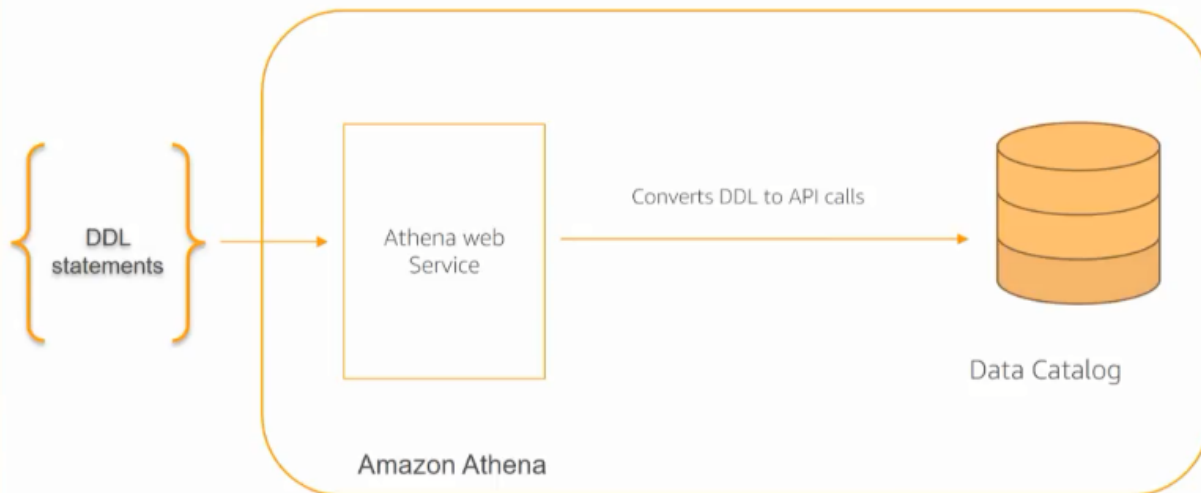
Optional custom building blocks for the grok pattern.

**AWS**  
**re:Invent**

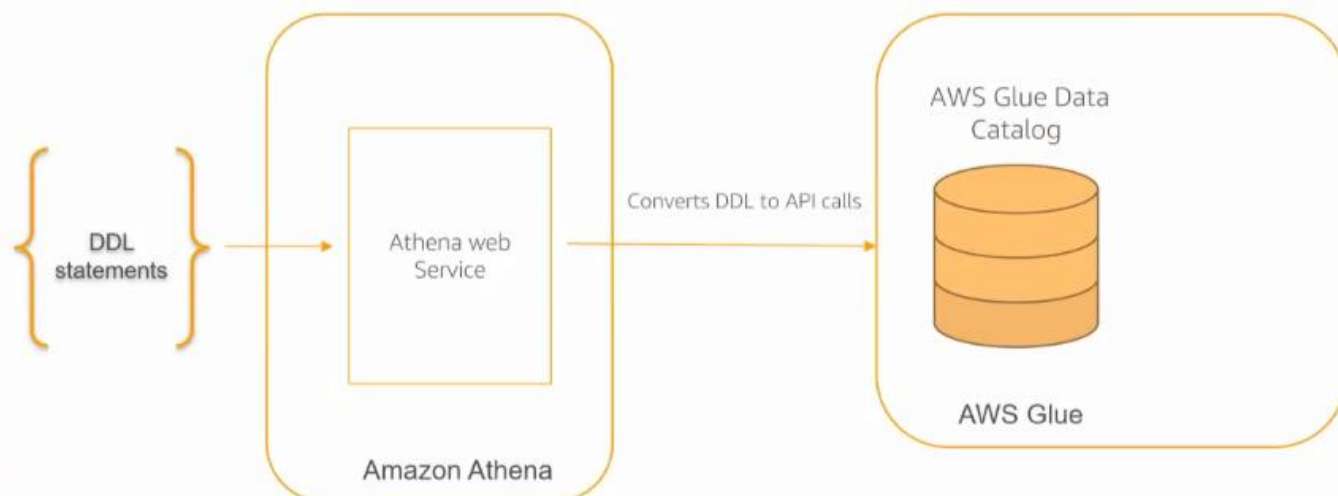
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## Amazon Athena Data Catalog or AWS Glue Data Catalog



## What happens when you upgrade to a AWS Glue Catalog



## Upgrade is a simple process

1. Add AWS Glue-specific policies to your existing Amazon 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), Amazon Athena and Amazon Redshift Spectrum
2. You can use the AWS 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 AWS Glue to ETL data and query in Amazon Athena

# Agenda

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

## Data Partitioning

- Separates data files by any column
- 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.
- Partitions are virtual columns that you can reference in your query
- Amazon Athena uses Hive-style partitioning
- Works on both text and columnar data

Partition allows you to read only the data that your query needs, partitioning is essentially a virtual column. You define partitioning at the time of defining a table and then in your WHERE clause, you will be able to use different filters and based on different filters, we might not read all the data and might read only a small chunk of this data.

If your data is text based, text is not really a good format for querying data because we have to read the entire dataset to be able to go to a specific column. This is not the same with columnar data formats like ORC and Parquet.

# 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

**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This query is about 74GB of data, the first query for a specific shipdate value takes about 9.71 secs to run on a table that is not partitioned, this is because it has to scan the entire table when using text data. **When we partition the entire table, it takes 2.16 secs and costs almost nothing.** This is because we simply look at the exact data we need and don't have to scan the entire dataset. The 2<sup>nd</sup> example is getting data between a range of 3 partitions.

## But it also has an overhead

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

**Partitioning your data means how you lay out your data in S3**, but partition might not be good for you if you are always going to scan your entire table because all the partitions will have to be fetched always.



# 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/<PARTITION\_COLUMN\_NAME1>=<VALUE>/<PARTITION\_COLUMN\_NAME2>=<VALUE>/

AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Partitioning is defined at the time of defining a table, assume ***we have a table that we want to partition by 2 things, the year and the month*** as above.

# 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/<PARTITION\_COLUMN\_NAME1>=<VALUE>/<PARTITION\_COLUMN\_NAME2>=<VALUE>/

AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



The way we need to lay out our data in S3 should use the partition keys year and month in the path to the data as above, this is how the data should be laid out in S3. In your SQL query, you can now specify a year=2015 and month=11, this will be a unique S3 location associated with that query tuple and only queries the files in this S3 location. ***Customers typically use a lambda function to sort incoming data into the right partition format in S3 so that they query less.***

# 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/'
```



AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Before you can use a partition, you have to tell the Data Catalog where the partition is. *The name of our table is **access\_logs***. You have to define that you want a partition in your CREATE TABLE statement, you have to lay out the data properly in S3, you have to also use the DDL statement to tell **Athena** where the partition is located in the S3 bucket.

# Layout of data for partitioned tables

```
CREATE EXTERNAL TABLE access_logs
```

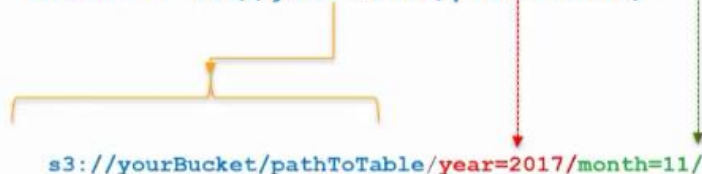
```
(  
  ip_address String,  
  request_time Timestamp,
```

```
)  
PARTITIONED by
```

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

```
STORED AS PARQUET
```

```
LOCATION 's3://yourBucket/pathToTable/'
```



AWS  
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

```
Alter Table access_logs
```

```
Add Partition
```

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

```
location
```

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

```
MSCK Repair Table
```

```
GLUE API
```



We can do the steps using an Alter Table command and pass the table name, Add Partition and then give each partition a value, then associate the value with an S3 path as above.

# 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/
```



**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Alter Table access\_logs

Add Partition

year='2017',

month='11')

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

~~MSCK Repair Table~~

GLUE API



What if we have data located as above, it does not have 'year=2017/month=11' as the previous case. It does not matter because we can still specify the key for the value to use.

## Partition loading and S3 prefixes

1

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

This MSCK REPAIR TABLE command is an expensive command to load all your partitions because it will scan all your data.

## 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/'
```

If we have data in the format having no year and month in the querystring as in #2, we can use the Alter Table command and give the values as above

# 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/'
```

**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We can even use the #3 where we introduce entropy in the prefix names of S3 objects. We can do the same thing as above

## Loading Partitions

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

**AWS re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



There are many ways to do partition as listed above



# Choose partitions to get performance even on text data

1. Columns that can be used as filter (e.g. date)
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

Try to choose your partitions so that the partitioning is balanced and each partition about 128MB of data in it. This will help optimize your query patterns

## Agenda

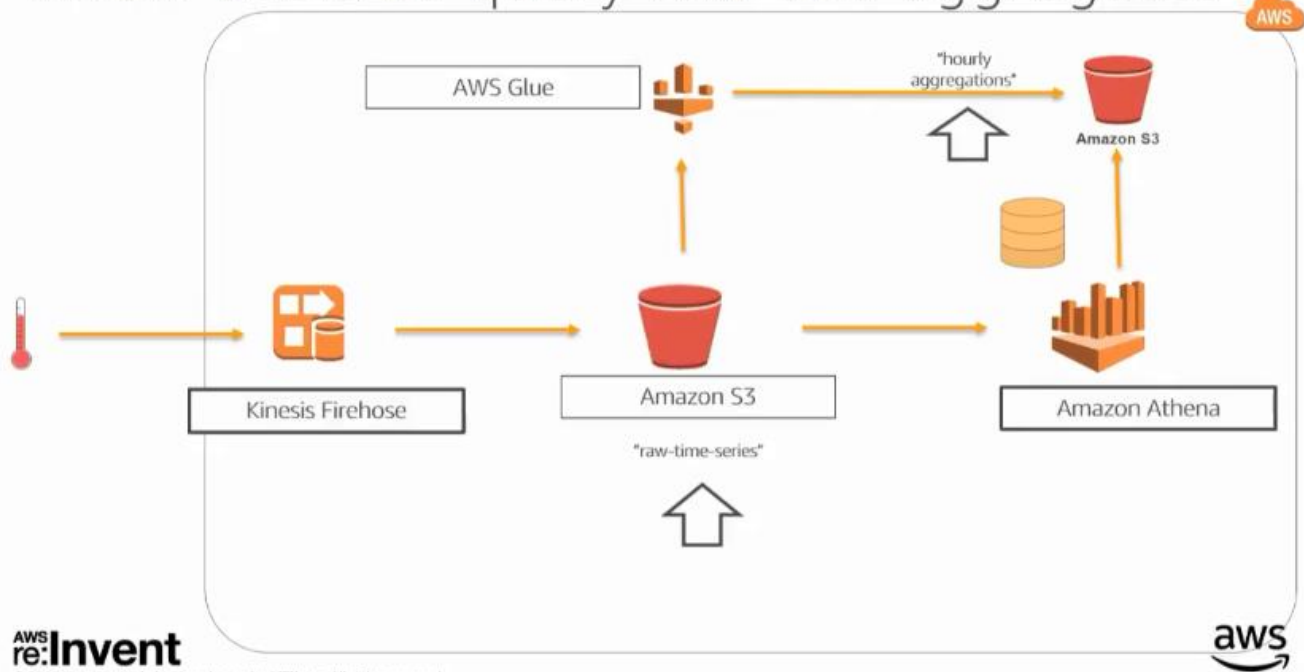
1. Review of the year
2. Use cases seen since launch
3. Amazon Athena, Amazon Redshift Spectrum, and Amazon EMR
4. Connecting with Amazon Athena
5. Creating Tables
6. AWS Glue Data Catalog
7. Partitioning data
8. Running queries and performance optimizations
9. OLX case study

## Query Performance – bigger files help

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

Athena does not like small files. You need to concatenate smaller files into larger files before storing them in S3.

## Allow users to query raw and aggregates



Customers often use AWS Glue to aggregate the data and create hourly file aggregations for small files.

## Query performance- ORC and Parquet

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

When you convert your data into ORC and Parquet columnar formats, we run faster because the data is compressed. They also allow the query to select only the referenced columns and also get needed data faster.

# Query Performance – Order by

## Example:

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

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

Using LIMIT clauses as above within your ORDER BY clauses will make the query run much faster and cheaper.

# Query Performance - Joins

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

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

Keep the larger Table on the Left Side of the join

If you keep your larger table on the left side of the join is better. The **part** table is larger than the **lineitem** table and should be placed on the left in the query as in the 2<sup>nd</sup> example above.

# 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

Regex based operators are much better than LIKE based operators as shown above

## Coming soon !

1. Improved support for large aggregations with the ability to Spill-to-disk
2. A new optimized PARQUET reader with better support for complex data types
3. Better error messaging
4. Upgraded to a version based on Presto 0.172 (support for Lambda expressions)

## Agenda

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

CLICK TO ADD TEXT

# Athena @ OLX Brazil

## Data Engineering & Service Team

Raúl Rentería

**AWS**  
**re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



## OLX Brazil – Biggest Online Marketplace



Located in Rio de Janeiro, Brazil since 2010

Part of Naspers and Schibsted,  
world leaders in Online marketplaces



**+26B USD**  
**circulated** during  
2016 – **1.4%** of  
Brazilian GDP

700K Brazilians  
publish for the first  
time every month

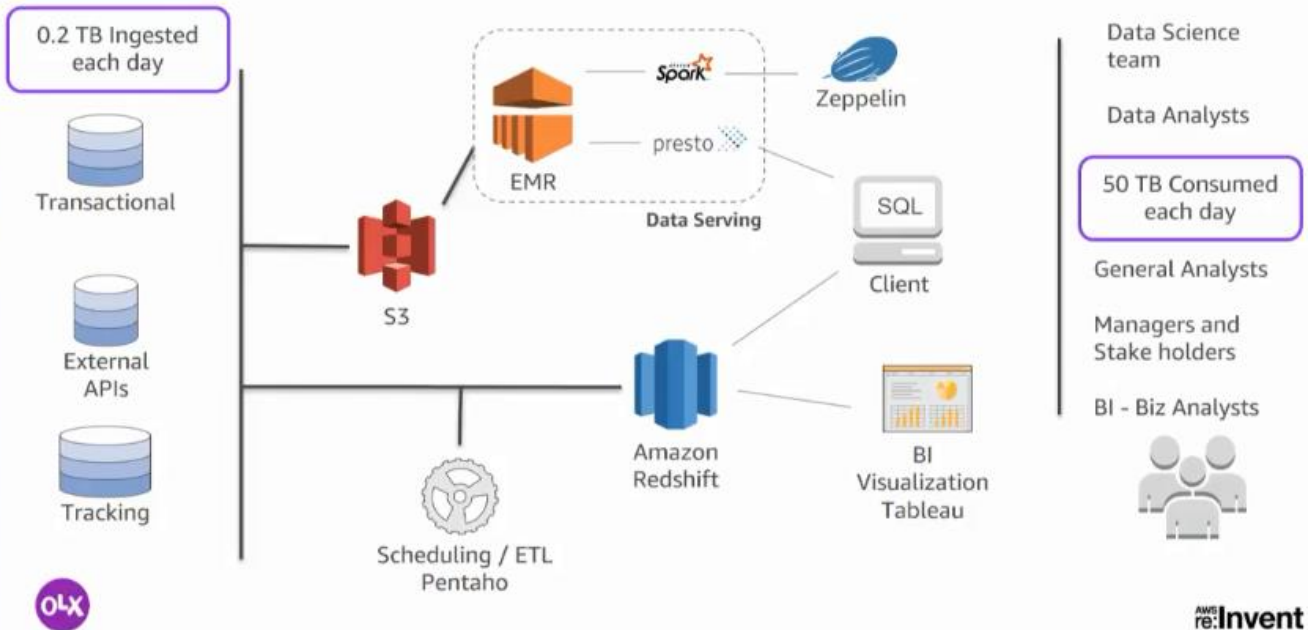
**25M sold items**  
during 2016 – 90%  
growth YOY

**2.1M cars sold, 1/5** of  
Brazilian sales volume  
2016

**AWS**  
**re:Invent**



# OLX Data Platform – December 2016



## Data Platform – High Demand Scenario

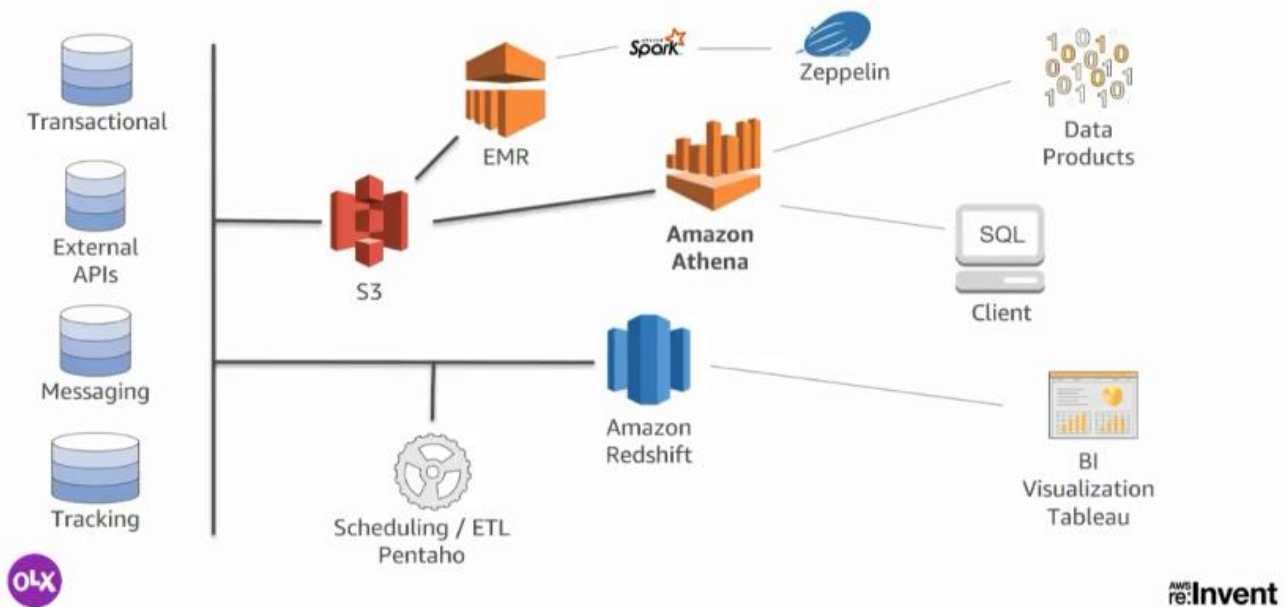
Increasing number of users all over the company – 20% employee growth YOY

- BI for Business and General Analysts
- Data Scientists, PMs, Developers and Stakeholders
- Clear push from the company to be data driven in every department

Increasing number of data / machine learning based products processes

- New messaging service
- Increased tracking coverage
- User profile for content understanding
- Collaborative filtering for Content Recommendation

# OLX Data Platform – January 2017



Athena is now our query engine that now serves query results for all the Data Products projects that we have.

## Data Platform – Amazon Athena Spot

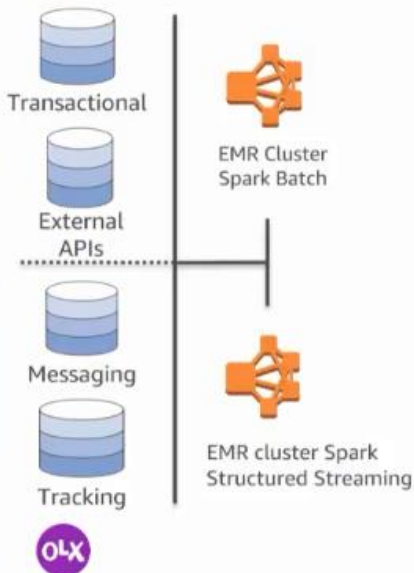
Amazon Athena launched late 2016

- Same time OLX Brazil faced increased data demand
- Demand mostly from users but also processes
- Pushing existing infrastructure would not be effective – quality and cost

Amazon Athena Early Adopter – January 2017

- Implemented in 2 days
- Straightforward on top of existing S3 Data formatting

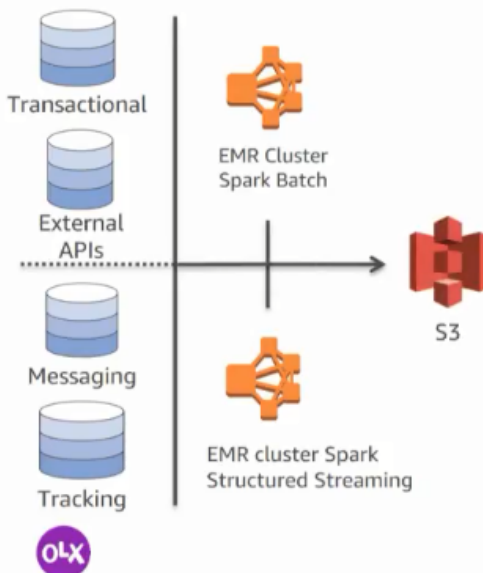
# How we Implemented Amazon Athena



AWSS re:Invent

Our data architecture consists of 2 clusters, a batch-oriented data cluster using Spark Batch that loads data from the transactional systems and external APIs. We also have a 2<sup>nd</sup> cluster that handles all the streaming based jobs.

# How we Implemented Amazon Athena



Data ingested by Spark job in columnar format

Data Partitioned by [Year, Month, Day]

Compressed ORC format used

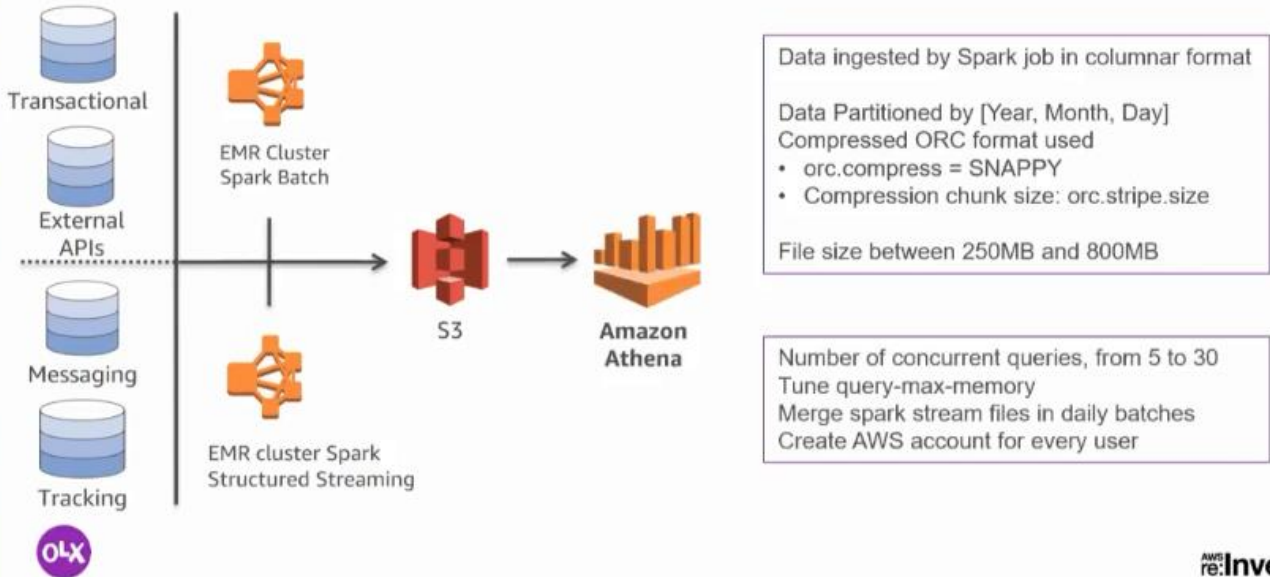
- orc.compress = SNAPPY
- Compression chunk size: orc.stripe.size

File size between 250MB and 800MB

AWSS re:Invent

The 2 cluster data pipelines will acquire their data from sources and ingest it into S3 as our data lake with the data partitioned in Year, Month, and Day partitions. we also used ORC data compression and we also used a job to aggregate files to keep them between 250MB and 800MB each for better query performance.

# How we Implemented Amazon Athena



The then added Athena with some further settings shown above

## Data Platform Achievements

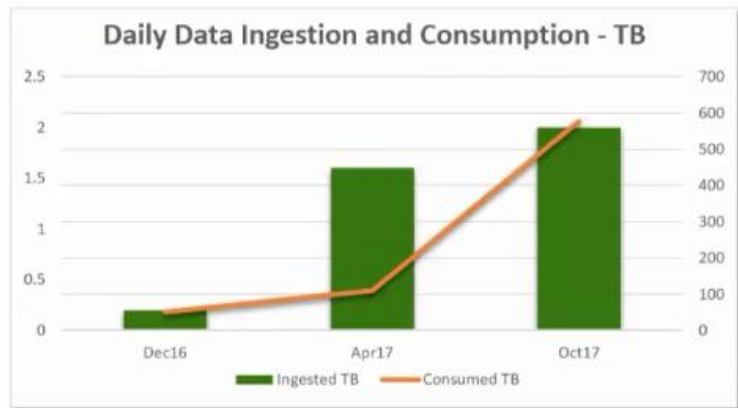
Fast deploy time – 2 days with existing S3 data structure

Scalable solution providing quality query time

- Messaging, Recommendation, ad hoc queries...

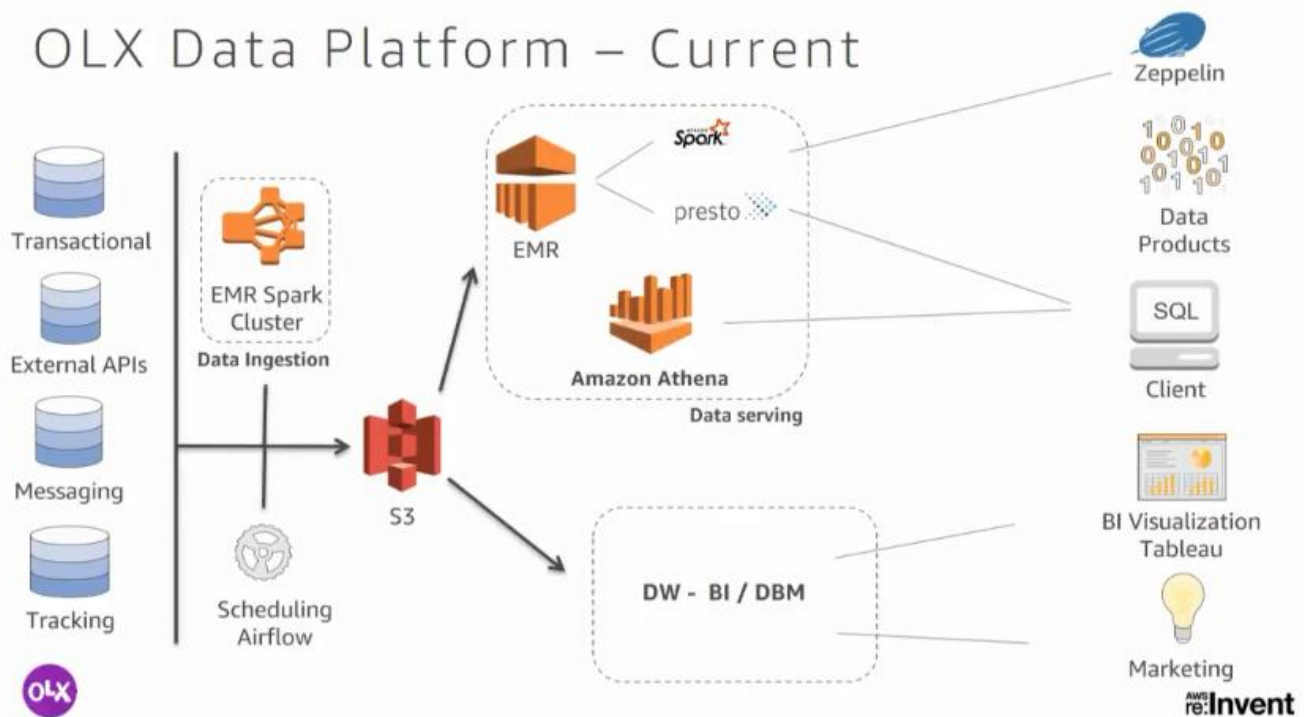
Other alternatives would require

- Non available team resources
- Stressing existing architecture in a non-effective way  
+100K USD / month



We are now seeing a lot of users within our company as seen in the data ingestion data above

# OLX Data Platform – Current



We have some users that need to use Presto as another query engine alongside Athena.

## Take away

### Challenges

Widespread use: every department should run their own analyses

Not everyone is an SQL expert 😊

### To consider

Think about your data format on S3 earlier

Evaluate your Data scale and Team size

Evaluate constrained and unconstrained querying environments (EMR Presto/Amazon Athena)

How much you need resource control through data dictionary



# About Us

Raúl Rentería

Head of Big Data – OLX Brazil

Email: [raul.renteria@olxbr.com](mailto:raul.renteria@olxbr.com)

We're hiring - Learn more about OLX Brazil at  
<http://www.linkedin.com/company/olx-brasil>

Contributed to this presentation: Marcello Fonseca and Hudson Santos – thanks!

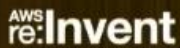


Athena @ OLX Brasil

# AWS re:Invent

## Thank you!

Raúl Rentería – OLX Brazil  
Data Engineering & Services Team



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

