

# Build Your First Big Data Application on AWS

Ben Willett

Sr. Solutions Architect

Pablo Redondo Sanchez

Sr. Solutions Architect

Mike George

Sr. Technical Account  
Manager

Graham Zulauf

Sr. Solutions Architect



# Workshop Agenda

- Download this presentation from Github
- Apply \$25 AWS usage credit to your account (5 mins)
- Create AWS resources via CloudFormation (10 mins)
- Review application and AWS services used (45 mins)
- Build Your First Big Data Application on AWS! (90 mins)
- <https://github.com/wrbaldwin/da-week>

# Launch AWS CloudFormation template

- Browse to the link below and select New VPC
- Select **N. Virginia** or **Oregon** or **Ireland** or **Frankfurt** for the region to launch your stack.
- <https://tinyurl.com/y6v2hjuz>

## AWS Account

In order to complete this workshop you'll need an AWS Account, and an AWS IAM user in that account with at least full permissions to the following AWS services:

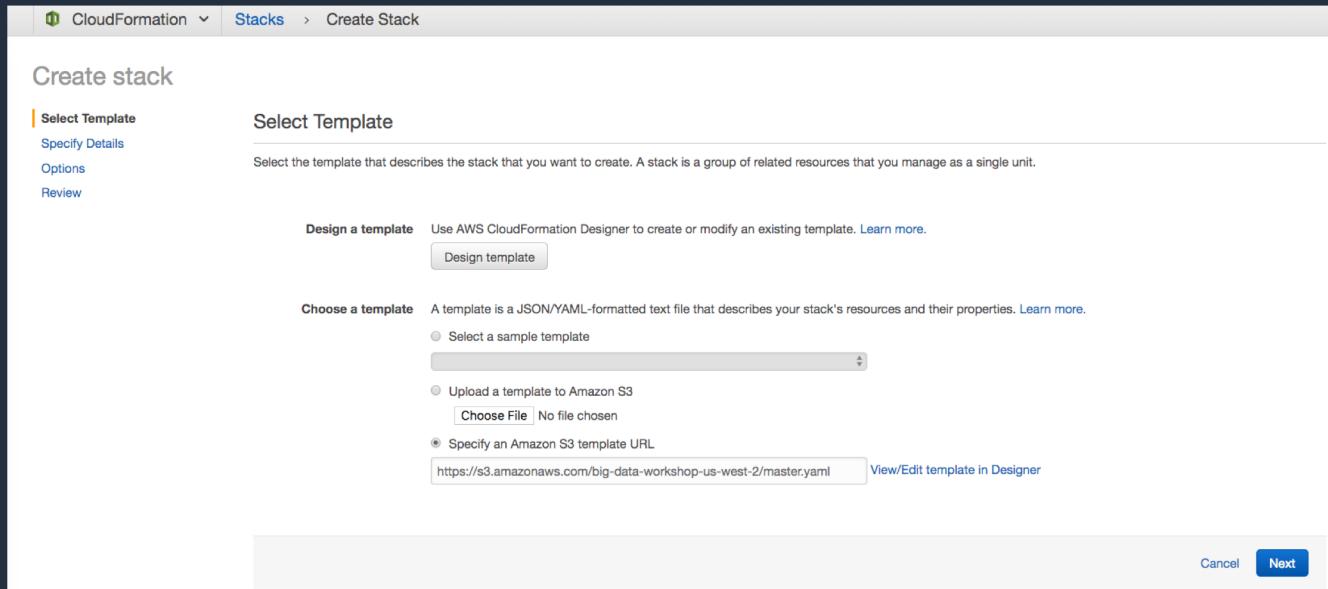
- AWS IAM
- Amazon S3
- Amazon Kinesis
- AWS Glue
- Amazon Redshift
- Amazon SageMaker

You can launch this CloudFormation stack in your account in one of the supported Regions below:

AWS Region	Short name	New VPC	Existing VPC
US East (N. Virginia)	us-east-1	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>
US West (Oregon)	us-west-2	<a href="#">Launch Stack</a>	<a href="#">Launch Stack</a>

# Launch AWS CloudFormation template - Continued

- After selecting New or Existing VPC, you will be launched into the AWS CloudFormation service
- Verify the Amazon Simple Storage Service (Amazon S3) template URL, and click “Next”



# Launch AWS CloudFormation template - Continued

- All defaults can be left as-is for this lab. Depending on if you selected new or existing, the only change is picking a CIDR range for the VPC or selecting an existing VPC in your account.
- If there is a Amazon Redshift password in there, leave as is.
- If there is no password, enter Abc12345 as the password. Make a note of this for later.
- Click “Next”

Create stack

Select Template  
**Specify Details**  
Options  
Review

Specify Details

Specify a stack name and parameter values. You can use or change the default parameter values, which are defined in the AWS CloudFormation template. [Learn more.](#)

Stack name

Parameters

Redshift Configuration

User Name  Default value is 'Abc12345'. Include 1 Upper, 1 Lower, 1 Number, min 8 characters

Password  Database Name

Node Type  The type of Redshift node to be provisioned

Network Configuration

Vpc CIDR  Please enter the IP range (CIDR notation) for the VPC to be created

Public Subnet CIDR  Please enter the IP range (CIDR notation) for the public subnet

Environment Configuration

Environment Name  An environment name that will be prefixed to resource names

Include Redshift  Specifies whether to include the Redshift section of the workshop.

Include EMR  Specifies whether to include the EMR section of the workshop.

Include SageMaker  Specifies whether to include the SageMaker section of the workshop.

[Cancel](#) [Previous](#) **Next**

# Launch AWS CloudFormation template - Continued

- Leave the Options at their defaults and click “Next”.

**Options**

**Tags**  
You can specify tags (key-value pairs) for resources in your stack. You can add up to 50 unique key-value pairs for each stack. [Learn more](#).

Key (127 characters maximum)	Value (255 characters maximum)
1	

**Permissions**  
You can choose an IAM role that CloudFormation uses to create, modify, or delete resources in the stack. If you don't choose a role, CloudFormation uses the permissions defined in your account. [Learn more](#).

IAM Role: Choose a role (optional)  
Enter role arn:

**Rollback Triggers**  
Rollback triggers enable you to have AWS CloudFormation monitor the state of your application during stack creation and updating, and to rollback that operation if the application breaches the threshold of any of the alarms you've specified. [Learn more](#)

Monitoring Time: 0-180 Minutes  
Minimum value of 0. Maximum value of 180.

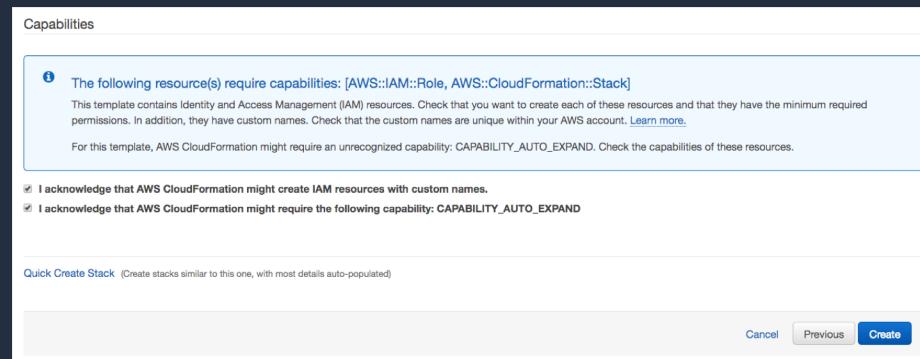
Type	ARN (Amazon Resource Name)	Available triggers remaining: 5
1 AWS::CloudWatch::Alarm		+ 5

**Advanced**  
You can set additional options for your stack, like notification options and a stack policy. [Learn more](#).

[Cancel](#) [Previous](#) **Next**

# Launch AWS CloudFormation template - Continued

- Acknowledge the AWS CloudFormation template will create AWS Identity and Access Management (IAM) resources in your account
- Acknowledge the CAPABILITY\_AUTO\_EXPAND as well.
- Click “Create”
- Verify the stacks are in “CREATE\_IN\_PROGRESS“ status.
- Wait for all CREATE\_COMPLETE (approx. 15 minutes)



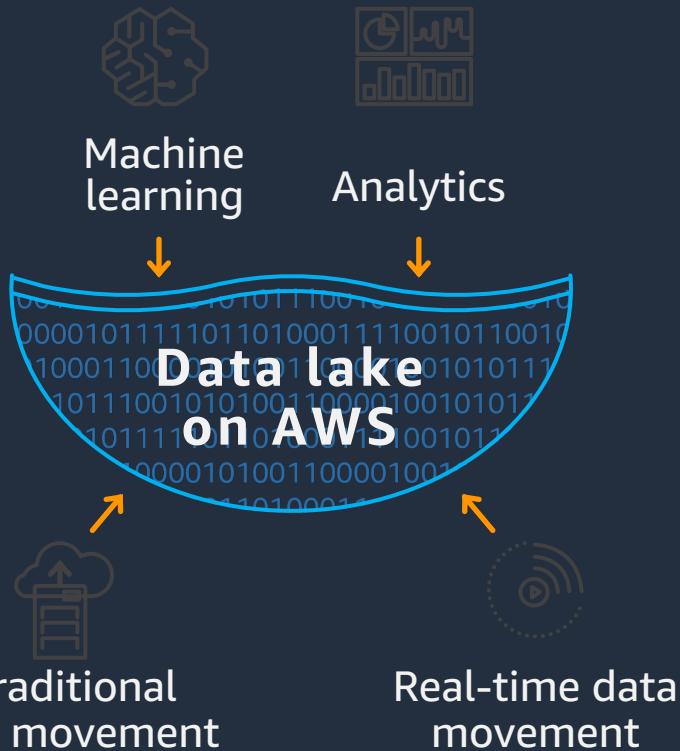
# Workshop objectives

- Build an end-to-end analytics application on AWS
- Implement batch and real-time layers in a single application
- Explore AWS analytics portfolio for other use cases

# Important! – Delete CloudFormation Stacks When Done

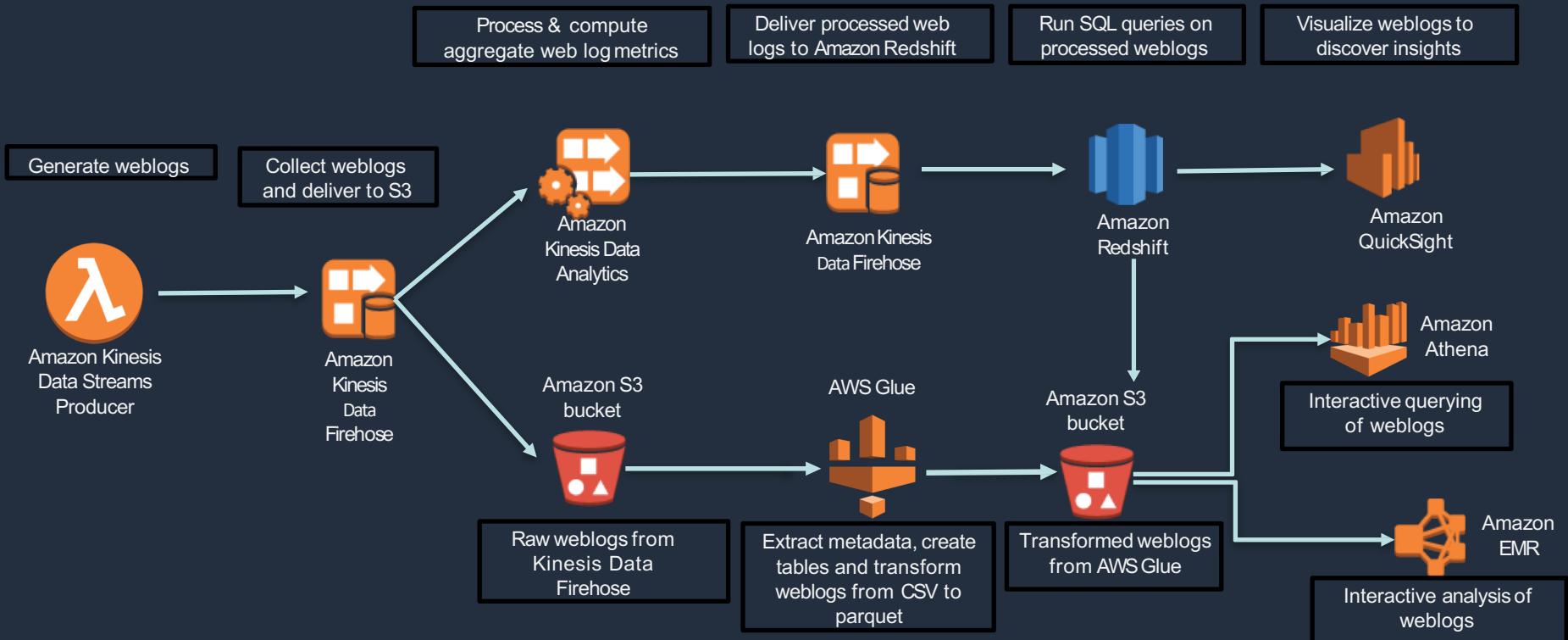
- **All AWS resources** created in this lab **should be removed** by deleting the CloudFormation Stacks in your AWS account so **you stop paying for them**
- **If you are not sure, ask for help!**

# Why analytics on AWS?



- Unmatched durability, and availability at EB scale
- Security, compliance and audit capabilities.
- Object-level controls for fine-grained access
- Fast performance by retrieving subsets of data
- Analyze with broad set of analytics & ML services
- Future proof your data lake architecture

# Your application architecture



# Streaming ingest with Amazon Kinesis

# Streaming with Amazon Kinesis

Easily collect, process, and analyze video and data streams in real time



## Kinesis Video Streams

Capture, process, and store video streams



## Kinesis Data Streams

Capture, process, and store data streams



## Kinesis Data Firehose

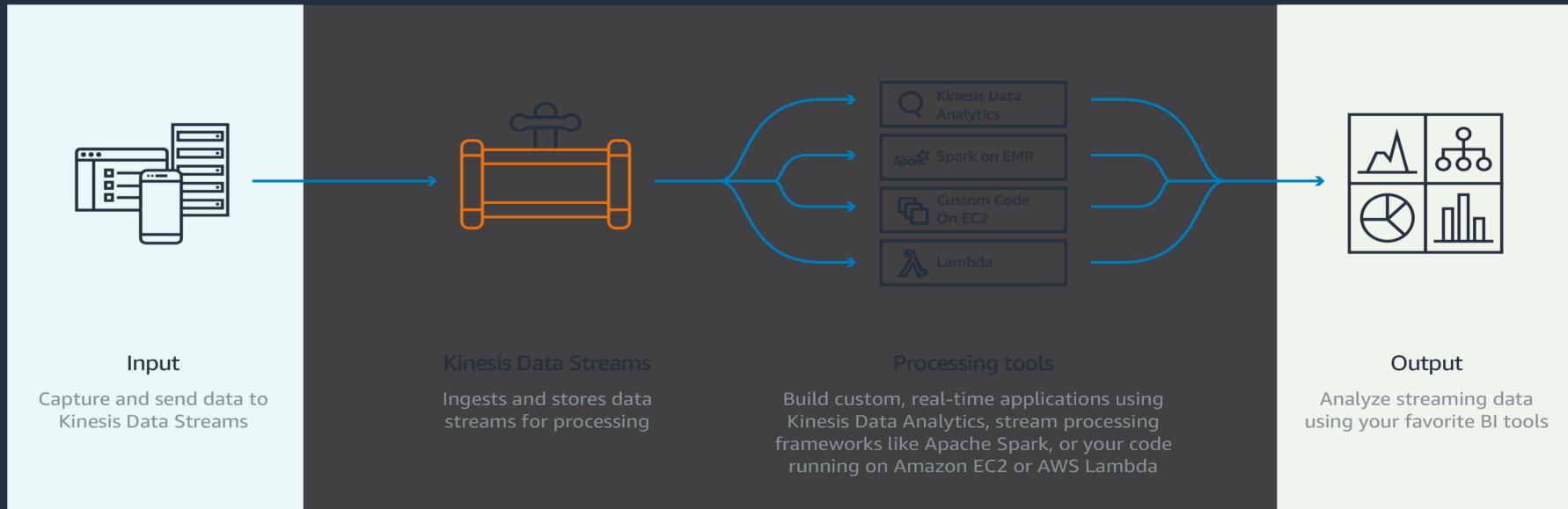
Load data streams into data stores



## Kinesis Data Analytics

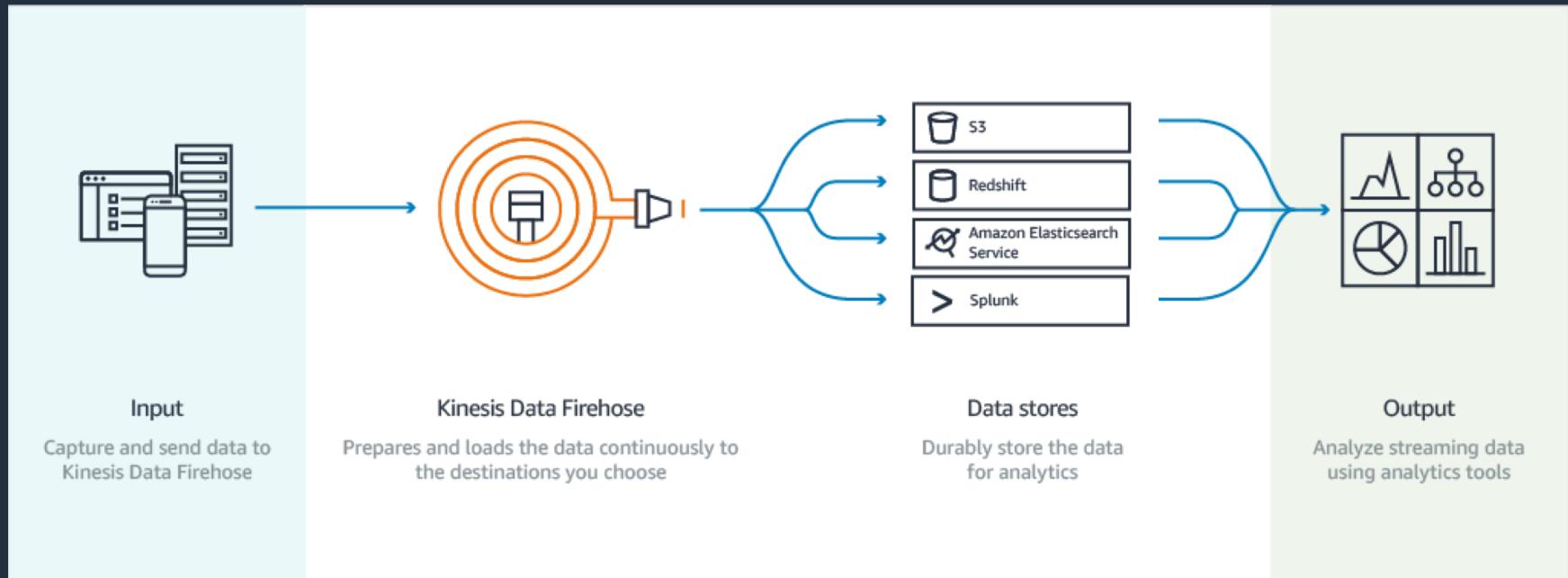
Analyze data streams with SQL

# Amazon Kinesis Data Streams



- Easy administration and low cost
- Build real-time applications with framework of choice
- Secure, durable storage

# Amazon Kinesis Data Firehose



- Zero administration and seamless elasticity
- Direct-to-data store integration
- Serverless, continuous data transformations

# Amazon Kinesis - Firehose vs. Streams



Kinesis Data  
Streams

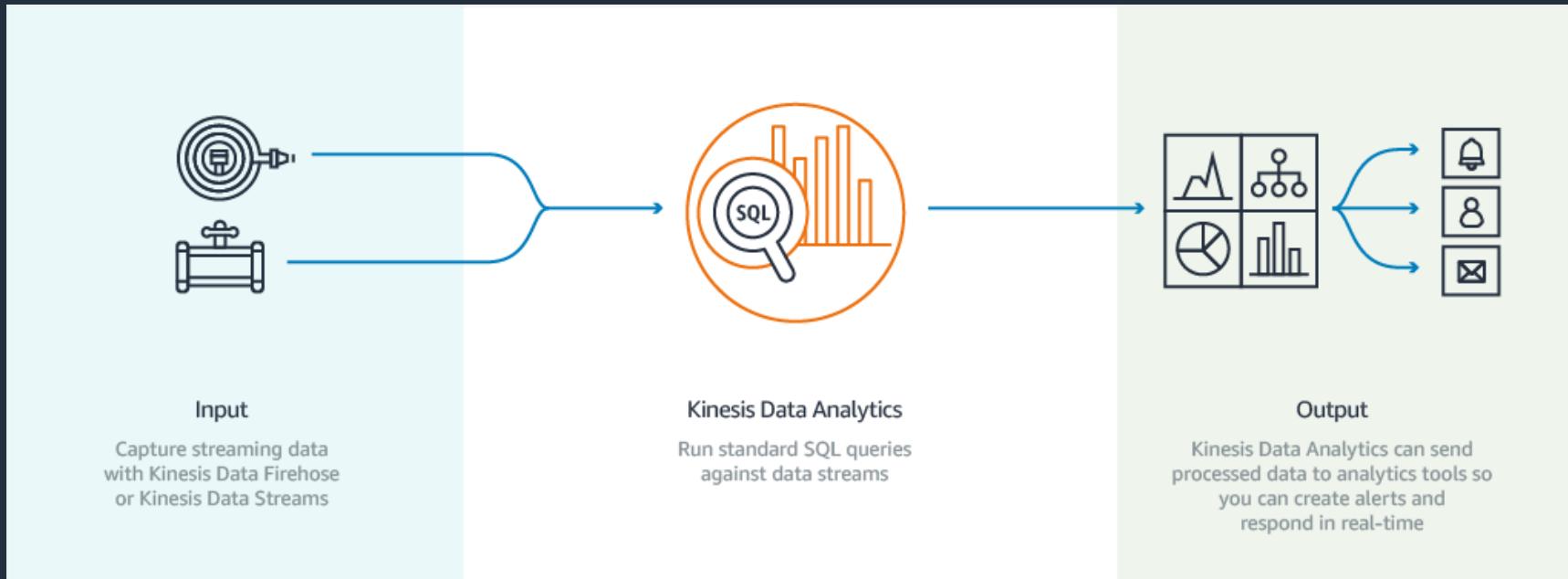
**Amazon Kinesis Data Streams** is for use cases that require custom processing, per incoming record, with sub-1 second processing latency, and a choice of stream processing frameworks



Kinesis Data  
Firehose

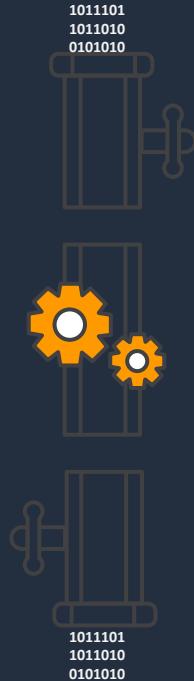
**Amazon Kinesis Data Firehose** is for use cases that require zero administration, ability to use existing analytics tools based on Amazon S3, Amazon Redshift, and Amazon ES, and a data latency of 60 seconds or higher

# Amazon Kinesis Data Analytics



- Powerful real-time applications
- Easy to use, fully managed
- Automatic elasticity
- Windowed aggregations

# Kinesis Data Analytics applications



Connect to streaming source

Easily write SQL code to process streaming data

Continuously deliver SQL results

# Kinesis Data Analytics application metadata

- Note that Amazon Kinesis adds metadata to each record being sent, which was shown in the formatted record sample:
- The **ROWTIME** represents the time when the Kinesis application inserts a row in the first in-application stream. It's a special column used for time series analytics. This is also known as the *processing time*.
- The **APPROXIMATE\_ARRIVAL\_TIME** is the time the record was added to the streaming source. This is also known as *ingest time* or *server-side time*.
- The **Event Time** is the timestamp when the event occurred. It's also called *client side time*. Its useful because it's the time when an event occurred at the client.

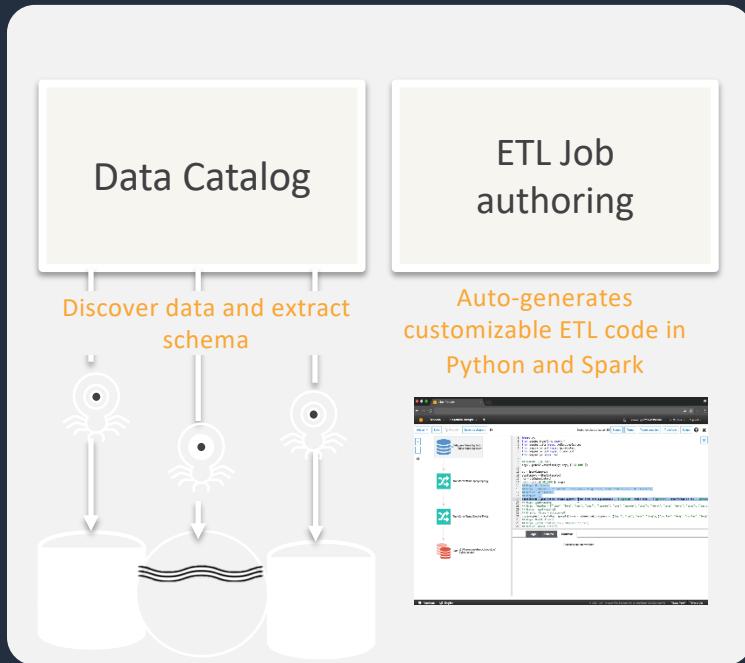
# Calculate an aggregate metric

- |          |   |
|----------|---|
| Tumbling | <ul style="list-style-type: none"><li>• Fixed size and non-overlapping</li><li>• Use FLOOR() or STEP() function in a GROUP BY statement</li></ul>   |
| Sliding  | <ul style="list-style-type: none"><li>• Fixed size and overlapping; row boundaries are determined when new rows enter window</li><li>• Use standard OVER and WINDOW clause</li></ul>  |
| Custom   | <ul style="list-style-type: none"><li>• Not fixed size and overlapping; row boundaries by conditions</li><li>• Implementations vary, but typically require two steps (Step 1—identify boundaries, Step 2—perform computation)</li></ul> |
| Stagger  | <ul style="list-style-type: none"><li>• Not fixed size and non-overlapping; windows open when the first event matching the partition key arrives</li><li>• Use WINDOWED BY STAGGER and PARTITION BY statements</li></ul>                |

# Extract, Transform, and Load (ETL) with AWS Glue

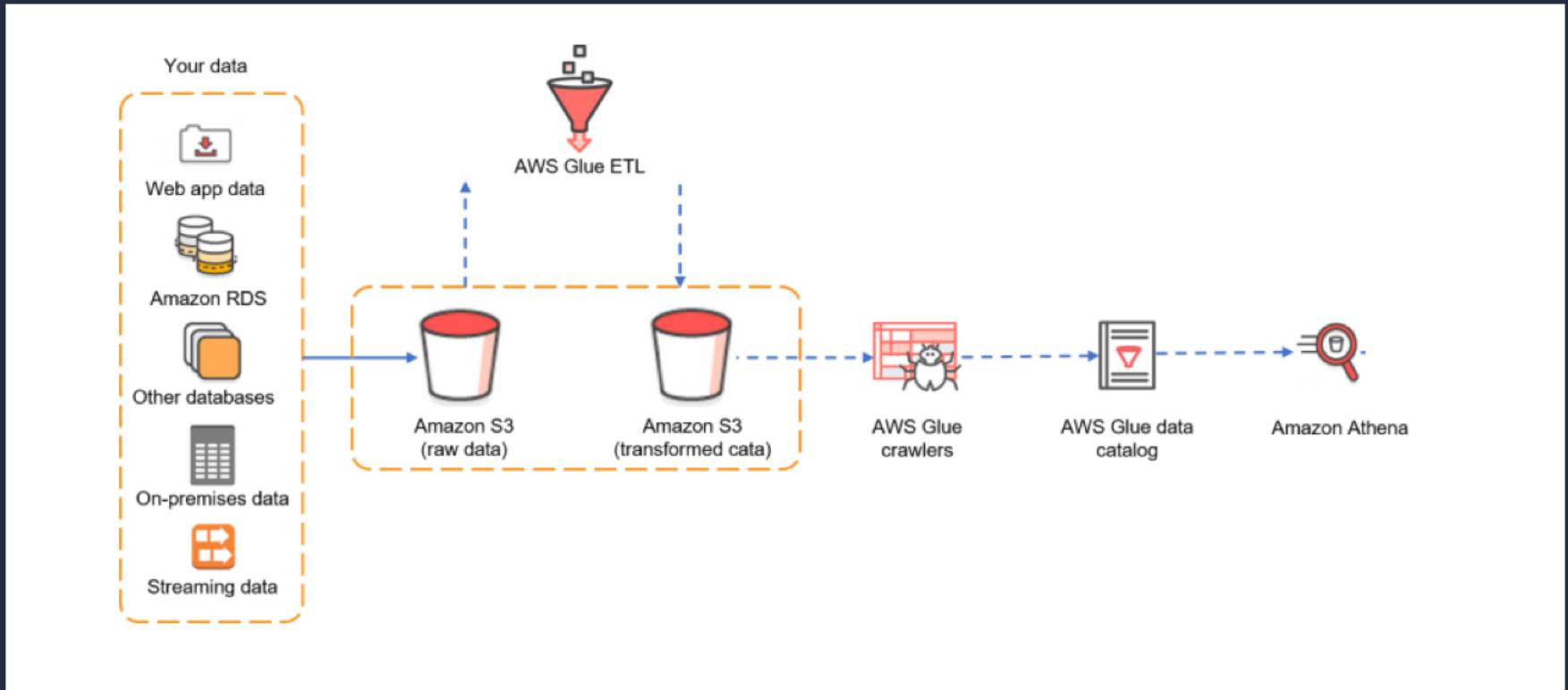
# AWS Glue - Components

Make data discoverable



- Automatically discovers data and stores schema
- Catalog makes data searchable and available for ETL
- Catalog contains table and job definitions
- Computes statistics to make queries efficient

# AWS Glue - How it works



# Querying data in Amazon S3 with Amazon Athena and Amazon Redshift Spectrum

# Interactive query service



Amazon  
Athena

- Query directly from Amazon S3
- Use ANSI SQL
- Serverless
- Multiple data formats
- Cost-effective

# 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

# Comparing performance and cost savings for compression and columnar format

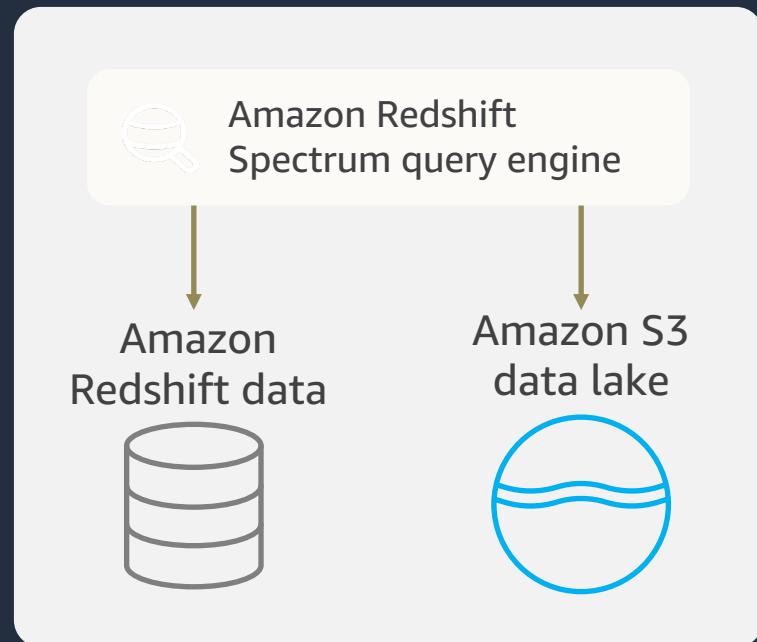
Dataset	Size on Amazon S3	Query run time	Data scanned	Cost
Data stored as text files	1 TB	236 seconds	1.15 TB	\$5.75
Data stored in Apache parquet format*	130 GB	6.78 seconds	2.51 GB	\$0.013
Savings / Speedup	87% less with parquet	34x faster	99% less data scanned	99.7% savings

(\*compressed using Snappy compression)

<https://aws.amazon.com/blogs/big-data/analyzing-data-in-s3-using-amazon-athena/>

# Amazon Redshift Spectrum

Extend the data warehouse to your Amazon S3 data lake



- Scale compute and storage separately
- Join data across Amazon Redshift and Amazon S3
- Amazon Redshift SQL queries against exabytes in Amazon S3
- Stable query performance and unlimited concurrency
- Parquet, ORC, Grok, Avro, & CSV data formats
- Pay only for the amount of data scanned

# Defining external schema and creating tables

- Define an external schema in Amazon Redshift using the Glue Data Catalog or your own Apache Hive Metastore

```
CREATE EXTERNAL SCHEMA <schema_name>
```

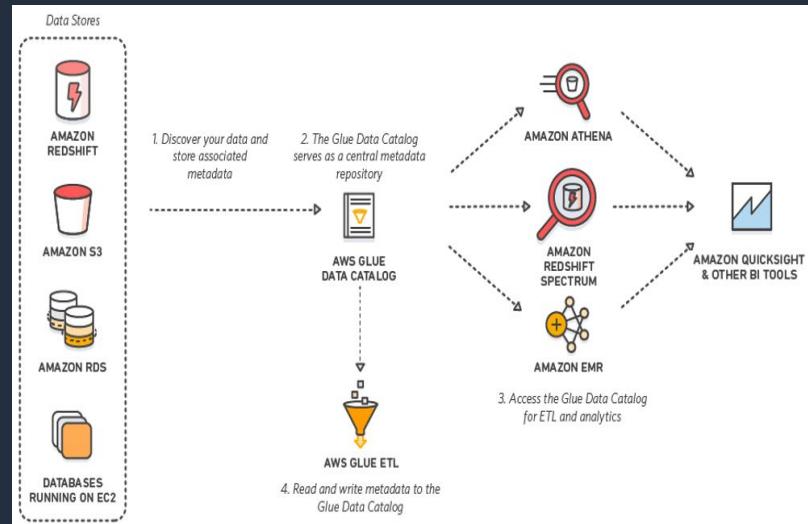
- Query external tables using <schema\_name>.<table\_name>

- Register external tables using AWS Glue Data Catalog, your Hive Metastore client, or from Amazon Redshift CREATE EXTERNAL TABLE syntax

```
CREATE EXTERNAL TABLE <table_name>
  [PARTITIONED BY <column_name, data_type, ...>]
  STORED AS file_format
  LOCATION s3_location
  [TABLE PROPERTIES property_name=property_value, ...];
```

# Recap: AWS Glue, Amazon Redshift Spectrum, and Athena

- Used the AWS Glue ETL job to convert and place the raw log data to parquet in S3
- Used the AWS Glue crawler to extract schema for parquet data in Amazon S3 and place in AWS Glue Data Catalog (weblogs\_dev.parquet)
- Used the Redshift Spectrum external table to query parquet data in Amazon S3 (spectrum.parquet)
- Used Athena to run queries on the same parquet data in Amazon S3 (weblogs\_dev.parquet)
- Both Athena and Redshift Spectrum use the same AWS Glue Data Catalog



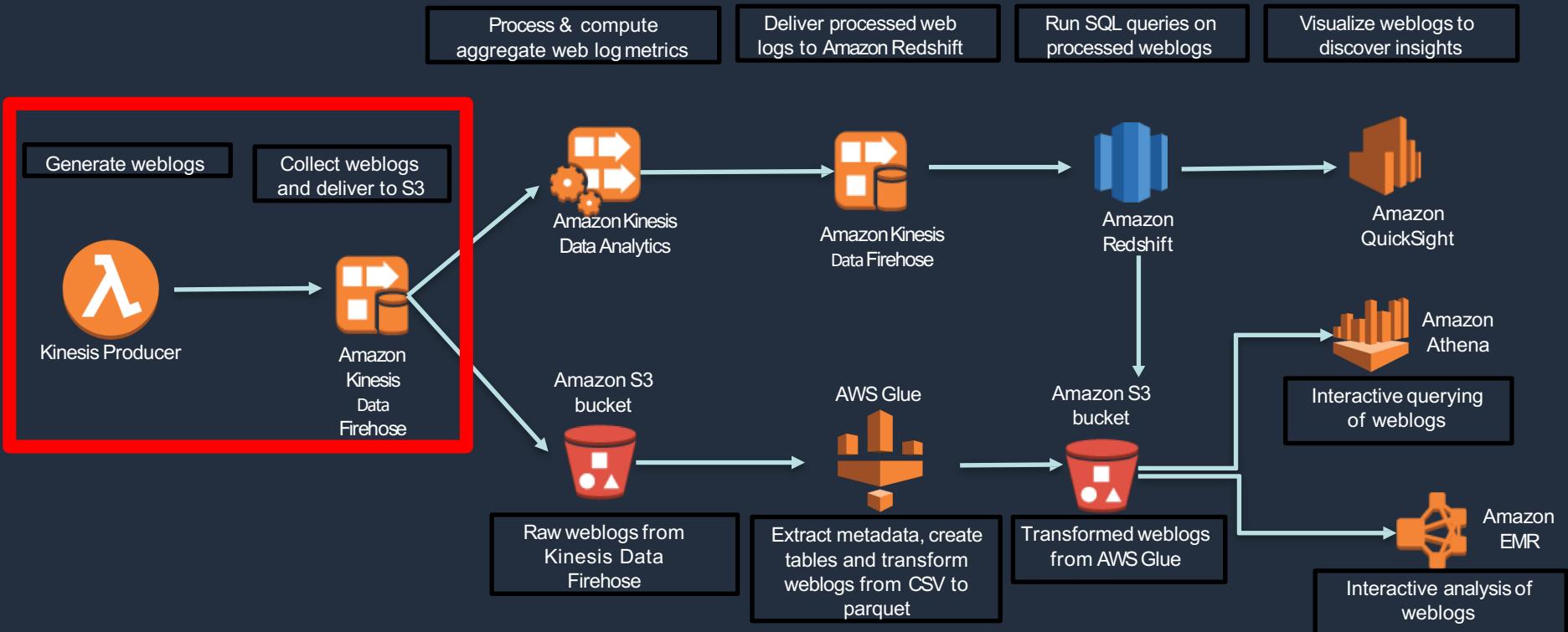
# Workshop activities schedule

- First big data app workshop activities:
  - 1: Collect logs with Kinesis Data Firehose delivery stream—5 mins
  - 2: Real-time streaming queries using Kinesis Data Analytics—20 mins
  - 3: Deliver streaming results to Amazon Redshift—5 mins
  - 4: Transform weblogs using AWS Glue—30 mins
  - 5: Query parquet data with Amazon Redshift Spectrum and Athena—30 mins
- Total hands-on lab time: 90 mins
- Activities have to be completed in sequence
- Pace yourself
- Helpers are available!

# Activity 1

## Collect logs using a Kinesis Data Firehose delivery stream

# Your application architecture



# Collect logs w/a Kinesis Data Firehose delivery stream

Time: 5 minutes

We are going to:

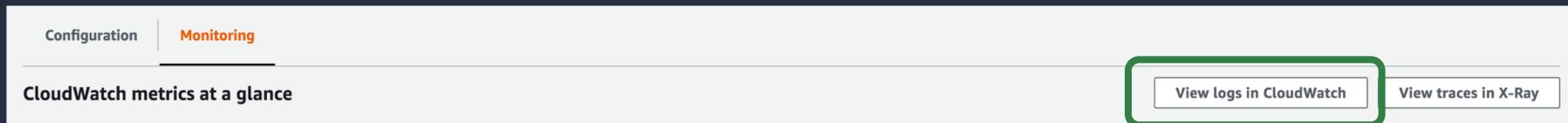
- Write to a Data Firehose delivery stream—Simulate writing transformed Apache weblogs to a Data Firehose delivery stream that is configured to deliver data into an S3 bucket.
- There are **many** different tools and libraries that can be used to write data to a Data Firehose delivery stream. One popular option is called the Amazon Kinesis Agent.

# Collect logs w/a Kinesis Data Firehose delivery stream

- So that we don't have to install or set up software on your machine, we are going to use a Lambda function to simulate using the Amazon Kinesis Agent. The Lambda function can populate a Data Firehose delivery stream using a template and is simple to setup.
- **Let's get started!**

# Activity 1A: Verify Lambda function delivering logs

- Go to the Lambda console and find a function named like:
  - <StackName>-KinesisStack-xxx-GenerateLogsLambdaFunc-xxxxxx
- Go to the Monitoring tab and click the “View logs in CloudWatch” button

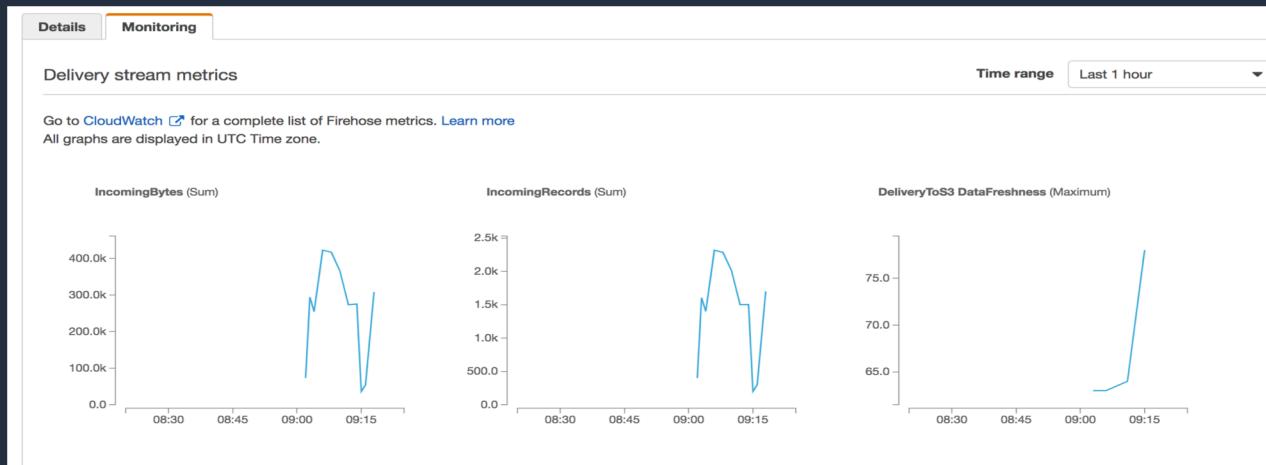


- View the top log stream and you should see Apache log entries like below:

Filter events		all 2018-09-24 (16:10:42)
Time (UTC +00:00)	Message	
2018-09-25		
▶ 16:02:43	START RequestId: 6f0337e3-c0dc-11e8-9f7a-71a603fe81fd Version: \$LATEST	
▶ 16:02:43	169.243.191.25 - - [25/Sep/2018:16:02:43 +0000] "POST /search/tag/list HTTP/1.0" 404 4989 "-" "Mozilla/5.0 (X11; Linux i686) AppleWebKit/536.2 (KHTML, like Gecko) Chrome/15.0.826.0 Safari/536.2"	
▶ 16:02:43	192.52.244.34 - - [25/Sep/2018:16:02:43 +0000] "GET /wp-admin HTTP/1.0" 200 5026 "-" "Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_12_0 rv:3.0; tt-RU) AppleWebKit/534.15.5 (KHTML, like Gecko) Version/5.1.7 Safari/534.15.5"	
▶ 16:02:43	198.51.98.170 - - [25/Sep/2018:16:02:43 +0000] "POST /wp-content HTTP/1.0" 200 5004 "-" "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 4.0; Trident/5.1)"	
▶ 16:02:43	192.175.63.43 - - [25/Sep/2018:16:02:43 +0000] "DELETE /list HTTP/1.0" 301 4960 "-" "Opera/8.33 (X11; Linux x86_64; hu-HU) Presto/2.9.181 Version/10.00"	
▶ 16:02:43	192.0.50.1 - - [25/Sep/2018:16:02:43 +0000] "POST /apps/cart.jsp?appId=8491 HTTP/1.0" 500 5023 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.2) AppleWebKit/532.28.1 (KHTML, like Gecko) Version/5.1.1"	
▶ 16:02:43	198.42.197.105 - - [25/Sep/2018:16:02:43 +0000] "POST /explore HTTP/1.0" 200 4968 "-" "Mozilla/5.0 (compatible; MSIE 8.0; Windows NT 5.2; Trident/3.0)"	
▶ 16:02:43	169.168.111.2 - - [25/Sep/2018:16:02:43 +0000] "POST /search/tag/list HTTP/1.0" 200 4957 "-" "Mozilla/5.0 (Windows NT 6.1; ms-MY; rv:1.9.2.20) Gecko/2010-03-09 23:35:22 Firefox/3.8"	

# Review: Monitoring Kinesis Data Firehose delivery to Amazon S3

- Go to the Kinesis console, click on “Data Firehose” and find data stream named like:
  - <StackName>-KinesisStack-xxx-FirehoseDeliveryStream-xxxxxxx
- Go to the Monitoring tab and there should be metrics for delivery to Amazon S3. This might take some time to show up.



# Review: Monitoring Kinesis Data Firehose Delivery to Amazon S3

- Go to the Kinesis console and click on “Data Firehose” and find data stream named like:
  - <StackName>-KinesisStack-xxx-FirehoseDeliveryStream-xxxxxx
- On the details tab you can see the S3 bucket the Apache logs are being streamed to:

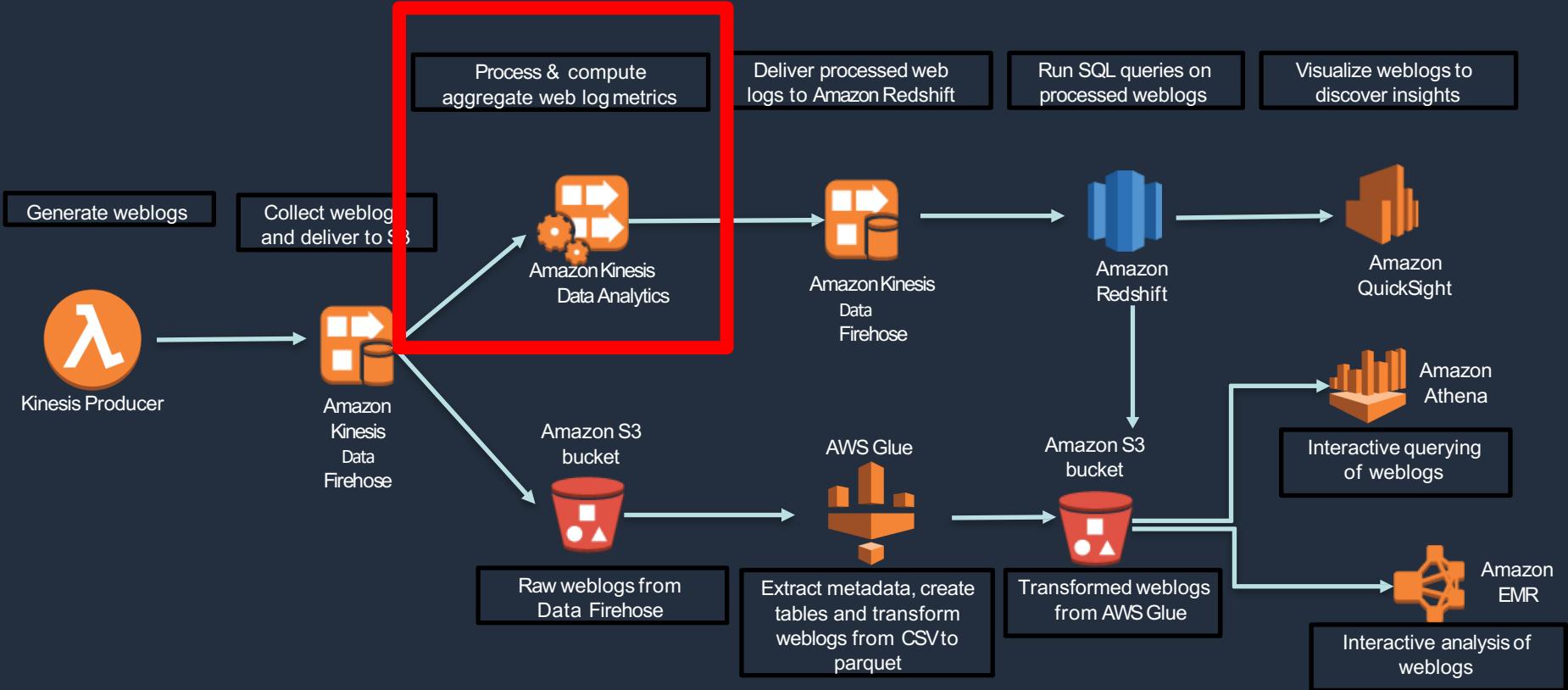
The screenshot shows the configuration for an Amazon S3 destination. A green box highlights the 'S3 bucket' field, which contains the value 'bdw-reinvent-2018-us-west-2-workshop'. Below it, the 'Prefix' is set to 'weblogs/raw/'. Other settings shown include 'Buffer conditions' (50 MB or 60 seconds), 'Compression' (GZIP), and a table below showing four log files. The table has columns for Name, Last modified, Size, and Storage class. All four log files are in the Standard storage class.

Name	Last modified	Size	Storage class
bd-reinvent-KinesisStack-1A-FirehoseDeliveryStream-QABJEPWPF7QX-1-2018-09-25...	Sep 25, 2018 9:03:51 AM GMT-0700	33.5 KB	Standard
bd-reinvent-KinesisStack-1A-FirehoseDeliveryStream-QABJEPWPF7QX-1-2018-09-25...	Sep 25, 2018 9:04:58 AM GMT-0700	28.3 KB	Standard
bd-reinvent-KinesisStack-1A-FirehoseDeliveryStream-QABJEPWPF7QX-1-2018-09-25...	Sep 25, 2018 9:06:01 AM GMT-0700	33.3 KB	Standard
bd-reinvent-KinesisStack-1A-FirehoseDeliveryStream-QABJEPWPF7QX-1-2018-09-25...	Sep 25, 2018 9:07:09 AM GMT-0700	28.2 KB	Standard

# Activity 2

## Real-time data processing using Amazon Kinesis Data Analytics

# Your application architecture



# Process data using Kinesis Data Analytics

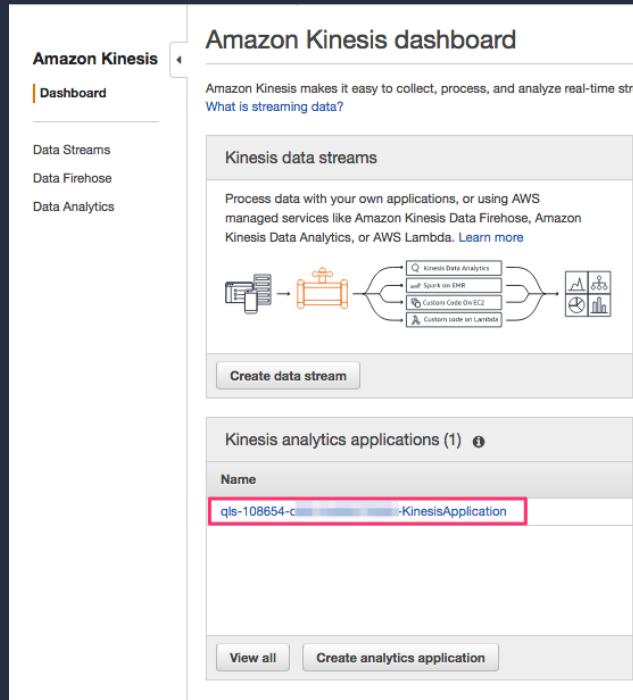
Time: 20 minutes

We are going to:

- Write a SQL query to compute an aggregate metric for an interesting statistic on the incoming data
- Write a SQL query using an anomaly detection function

# Activity 2A: Start Amazon Kinesis Data Analytics app

- Navigate to the Kinesis dashboard
- Click on the Kinesis Data Analytics application



# Activity 2A: Start Kinesis app

- Click on “Go to SQL editor”
- On the next screen, click on “Yes, start application”

Description: Kinesis Analytics Application Example  
Application ARN: arn:aws:kinesisanalytics:us-west-2:174689215959:application/qls-108654-d5fb1fe390279284-KinesisApplication  
Application version ID: 2 ⓘ

**Source**  
Connect to an existing Kinesis stream or Firehose delivery stream, or easily create and connect to a new demo Kinesis stream. The limit is one streaming source for each application. [Learn more](#).

Source	In-application stream name	ID ⓘ	Record pre-processing ⓘ
Firehose delivery stream <a href="#">qls-108654-d5fb1fe390279284-FirehoseDeliveryStream-1HTZRR63L1MRD</a> ⓘ	SOURCE_SQL_STREAM_001	1.1	apache_log_preprocess ⓘ

**Real time analytics**  
Continuously analyze your source data with SQL. [Learn more](#)

[Go to SQL editor](#)

Would you like to start running "qls-108654-d5fb1fe390279284-KinesisApplication" ⓘ ✖

The SQL editor is much more powerful when your application is running.

- See samples from your source data stream
- Get feedback on any errors in your configuration or SQL
- Watch as your data is processed in real-time by your SQL code

[No, I'll do this later](#) [Yes, start application](#)

# View sample records in Kinesis app

- Review sample records delivered to the source stream (SOURCE\_SQL\_STREAM\_001)

Filter by column name							Edit schema
ROWTIME TIMESTAMP	request_method VARCHAR(8)	request_time TIMESTAMP	response_size INTEGER	user_agent VARCHAR(256)	response_code INTEGER	host_address VARCHAR(16)	▲
2017-11-16 20:39:18.26	PUT	2017-11-16 10:39:17.0	4522	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.3; Trident/6.0; .NET CLR 2.6.78464.8)	200	17.199.76.178	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	7604	Mozilla/5.0 (Windows NT 6.2; Trident/7.0; Touch; rv:11.0) like Gecko	200	163.35.42.123	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	1757	Mozilla/5.0 (Windows; U; Windows NT 5.2) AppleWebKit/531.1.2 (KHTML, like Gecko) Chrome/21.0.856.0 Safari/533.1.2	200	176.101.197.231	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	3313	Mozilla/5.0 (Windows; U; Windows NT 5.1) AppleWebKit/531.1.0 (KHTML, like Gecko) Chrome/33.0.838.0 Safari/531.1.0	200	215.252.135.41	▼
2017-11-16 20:39:18.26	PUT	2017-11-16 10:39:17.0	715	Mozilla/5.0 (Windows; U; Windows NT 5.3) AppleWebKit/532.2.1 (KHTML, like Gecko) Chrome/25.0.896.0 Safari/532.2.1	200	243.24.77.108	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	4125	Mozilla/5.0 (Windows; U; Windows NT 6.1) AppleWebKit/536.2.0 (KHTML, like Gecko) Chrome/19.0.884.0 Safari/536.2.0	200	139.184.6.39	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	4538	Mozilla/5.0 (Windows; U; Windows NT 5.0) AppleWebKit/532.2.2 (KHTML, like Gecko) Chrome/29.0.846.0 Safari/532.2.2	200	224.35.169.60	▼
2017-11-16 20:39:18.26	DELETE	2017-11-16 10:39:17.0	2803	Mozilla/5.0 (X11; Linux i686 AppleWebKit/533.1.2 (KHTML, like Gecko) Chrome/39.0.834.0 Safari/533.1.2	200	154.116.241.193	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	7252	Mozilla/5.0 (Windows; U; Windows NT 5.3) AppleWebKit/532.1.1 (KHTML, like Gecko) Chrome/15.0.872.0 Safari/532.1.1	200	200.247.104.14	▼
2017-11-16 20:39:18.26	GET	2017-11-16 10:39:17.0	5539	Mozilla/5.0 (Windows NT 5.1; WOW64; rv:6.9) Gecko/20100101 Firefox/6.9.8	200	96.175.42.171	▼
2017-11-16 20:39:18.26	DFI FTF	2017-11-16 10:39:17.0	2649	Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 5.1; Trident/6.0; .NET CLR 1.1.18429.1)	301	116.42.71.19	▼

# Activity 2B: Calculate an aggregate metric

- Open the **KinesisAnalyticsSQL** file located in the **Scripts** section (Workshop Scripts)
- Copy and paste the entire SQL in the SQL editor of your Kinesis Data Analytics application, **OVERWRITING** the existing text
- Click “Save and run SQL”

The screenshot shows the AWS Kinesis Data Analytics SQL editor interface. The left sidebar navigation bar includes 'Amazon Kinesis' (selected), 'Dashboard', 'Data Streams', 'Data Firehose', 'Data Analytics' (selected), 'Video Streams', 'External resources', and 'What's new'. The main content area has a header 'Kinesis Analytics applications > bidataapp-reinvent-gs-KinesisStack-LHEV8E1OHN8R-KinesisApplication > SQL editor'. Below the header are buttons for 'Save and run SQL', 'Add SQL from templates', 'Download SQL', 'SQL reference guide', and 'Kinesis data generator tool'. The 'Real-time analytics' tab is selected. A code editor window contains the following SQL script:

```
22    "request_count" INTEGER,  
23    "avg_response_size" INTEGER;  
24  
25 /* Step 2 - Query for aggregating log data by response code and request time using stagger windows for late arriving events */  
26 CREATE OR REPLACE PUMP aggregate_pump AS INSERT INTO aggregate_stream  
27 SELECT STREAM "response_code", FLOOR("request_time" TO MINUTE), COUNT(*) as request_count, AVG("response_size")  
28 FROM DESTINATION_SQL_STREAM  
29 WINDOWDOW BY STAGGER (  
30     | PARTITION BY FLOOR("request_time" TO MINUTE), "response_code" RANGE INTERVAL '1' MINUTE);  
31  
32 /* Activity 2C: Anomaly detection */  
33 <
```

Below the code editor is a table titled 'Real-time analytics' with tabs for 'Source data', 'Real-time analytics' (selected), and 'Destination'. The 'Application status: RUNNING' is shown. The 'In-application streams:' section lists 'AGGREGATE\_STREAM' (selected), 'ANOMALY\_STREAM', 'DESTINATION\_SQL\_STREAM', and 'error\_stream'. The 'Real-time analytics' section displays a sampled table with columns: ROWTIME, response\_code, request\_time, request\_count, and avg\_re. The table data is as follows:

ROWTIME	response_code	request_time	request_count	avg_re
2018-10-09 22:18:12.256	404	2018-10-09 22:16:00.0	22	5003
2018-10-09 22:19:02.777	301	2018-10-09 22:18:00.0	223	4995
2018-10-09 22:19:02.777	500	2018-10-09 22:18:00.0	243	5000
2018-10-09 22:19:02.777	404	2018-10-09 22:18:00.0	219	5003
2018-10-09 22:19:02.777	200	2018-10-09 22:18:00.0	218	5002
2018-10-09 22:20:01.83	200	2018-10-09 22:19:00.0	248	4998

# Activity 2C: Anomaly detection

Take a look at the anomaly detection section in the SQL script:

- It creates an `anomaly_stream` with the attribute `anomaly_score`
- It calculates the anomaly score for each record in the stream by using the built-in random cut forest function

Real-time analytics

Save and run SQL Add SQL from templates Download SQL SQL reference guide Kinesis data generator tool

```
47 /* Compute an anomaly score for each record in the input stream */
48 /* using Random Cut Forest */
49 /* Step 2 - Compute anomaly score */
50 CREATE OR REPLACE PUMP anomaly_pump AS INSERT INTO anomaly_stream
51 SELECT STREAM ROWTIME,
52     "host_address", "request_time", "request_method", "request_path", "request_protocol",
53     "response_code", "response_size", "referrer_host", "user_agent",
54     anomaly_score
55 FROM TABLE(RANDOM_CUT_FOREST(
56     CURSOR(SELECT STREAM * FROM source_sql_stream_001)));
57
```

Source data Real-time analytics Destination Application status: RUNNING

In-application streams: AGGREGATE\_STREAM ANOMALY\_STREAM DESTINATION\_SQL\_STREAM error\_stream

Pause results New results are added every 2-10 seconds. The results below are sampled.

Scroll to bottom when new results arrive.

Filter by column name

	ANOMALY_SCORE
Firefox/3.6.8	0.73
	1.01
VебKit/535.25.5 (KHTML, like Gecko) Version/4.0.5 Safari/535.25.5	0.83
Chrome/36.0.869.0 Safari/5322	0.77
Chrome/34.0.846.0 Safari/5340	0.73

# Kinesis Data Analytics in-application streams

- In-application streams:
  - 1) Aggregate stream
  - 2) Anomaly stream
  - 3) Destination SQL stream
  - 4) Error stream

The screenshot shows the AWS Kinesis Data Analytics SQL editor. At the top, there's a navigation bar with services like EC2, Kinesis, Amazon Redshift, RDS, S3, EMR, and Database Migration Service. Below that is a breadcrumb trail: Kinesis Analytics applications > bigdataapp-techsummit-gs-KinesisApplication > SQL editor. The main area has tabs for 'Save and run SQL', 'Add SQL from templates', 'Download SQL', 'SQL reference guide', and 'Kinesis data generator tool'. A code editor window contains the following SQL:

```
1 CREATE OR REPLACE STREAM "DESTINATION_SQL_STREAM" (
2   process_time TIMESTAMP,
3   host_address VARCHAR(512),
4   request_time TIMESTAMP,
5   request_method VARCHAR(10),
6   request_path VARCHAR(1024),
7   request_protocol VARCHAR(10),
8   response_code INTEGER,
9   response_size INTEGER,
10 referrer_host VARCHAR(1024),
11  user_agent VARCHAR(512));
12
```

Below the code editor is a results panel titled 'Real-time analytics' with tabs for 'Source data', 'Real-time analytics' (which is selected), and 'Destination'. It shows application status as 'RUNNING'. Under 'In-application streams', there are four streams listed: 'AGGREGATE\_STREAM', 'ANOMALY\_STREAM', 'DESTINATION\_SQL\_STREAM', and 'error\_stream'. A message says 'New results are added every 2-10 seconds. The results below are sampled.' There's a checkbox for 'Scroll to bottom when new results arrive.' A search bar labeled 'Filter by column name' is present. A table displays sampled results with columns: ROWTIME, PROCESS\_TIME, HOST\_ADDRESS, REQUEST\_TIME, and F. One row of data is shown:

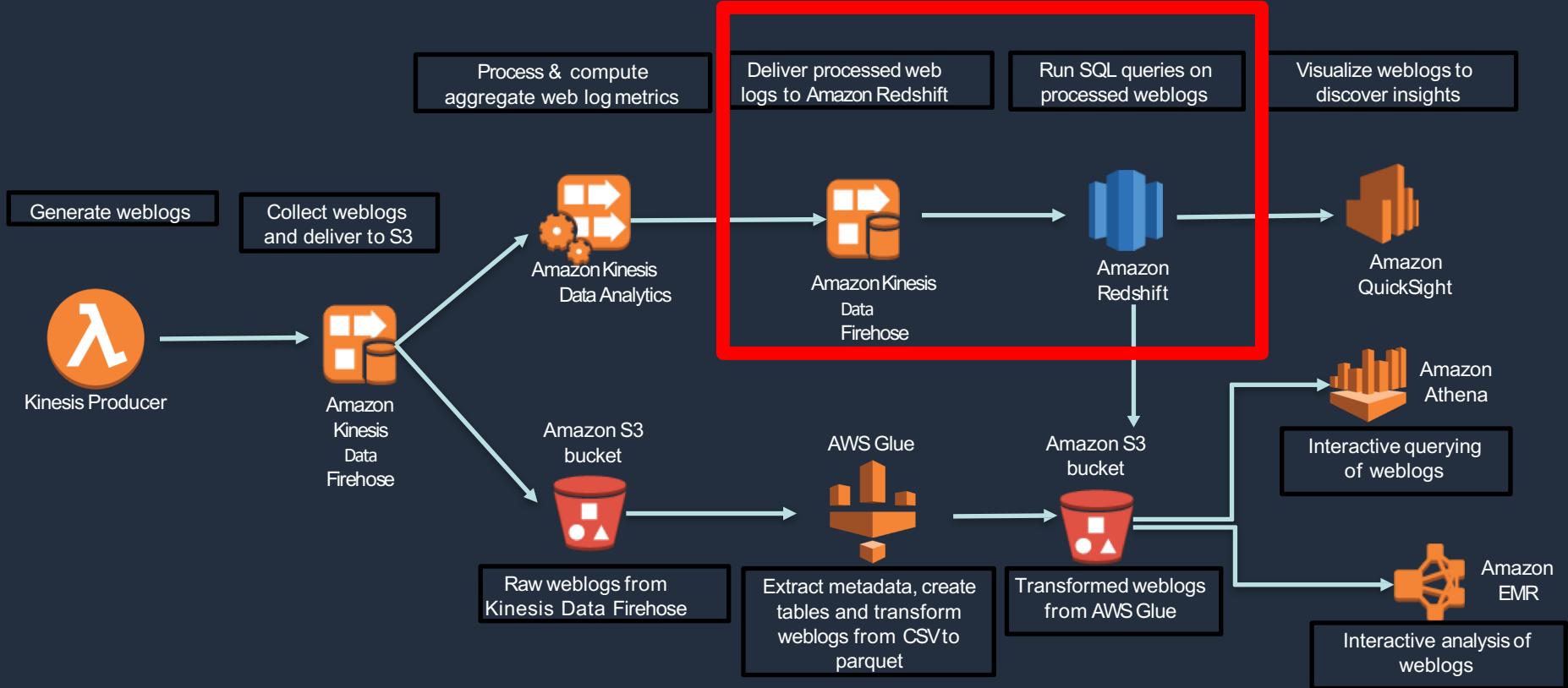
ROWTIME	PROCESS_TIME	HOST_ADDRESS	REQUEST_TIME	F
2018-07-03 22:14:10.675	2018-07-03 22:14:10.675	64.244.155.103	2018-07-03 08:14:09.0	C

A 'Close' button is at the bottom right of the results panel.

# Activity 3

## Deliver streaming results to Amazon Redshift

# Your application architecture



# Activity 3: Deliver data to Amazon Redshift using Kinesis Data Firehose

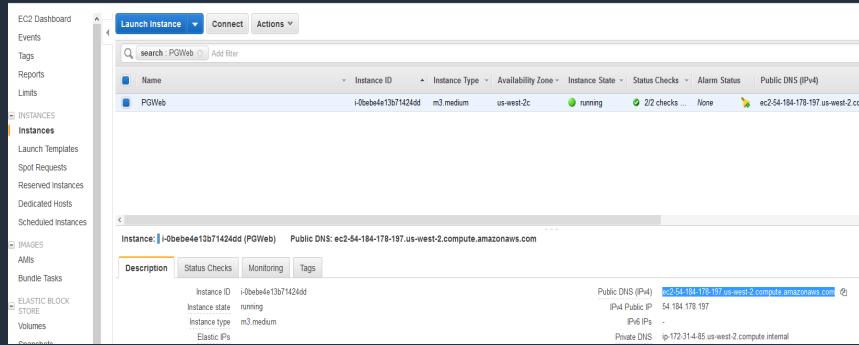
Time: 5 minutes

We are going to:

- Connect to Amazon Redshift cluster and create a table to hold web logs data
- Update Kinesis Data Analytics application to send data to Amazon Redshift, via the Data Firehose delivery stream

# Activity 3A: Connect to Amazon Redshift

- You can connect with [pgweb](#)
- Already installed for the Amazon Redshift cluster
- Just navigate to pgweb: Services> EC2> Select the pgweb instance> Copy the Public DNS (IPv4)> Paste in your browser> Start interacting with pgweb



- Or, use any JDBC/ODBC/libpq client
  - [Aginity Workbench for Amazon Redshift](#)
  - [SQL Workbench/J](#)
  - [DBeaver](#)
  - [Datagrip](#)
- If you use the above SQL clients, the username/password is in AWS CloudFormation
- Select the big data CFN template and go to the Outputs tab
- You will find the Amazon Redshift cluster end point, username, and password

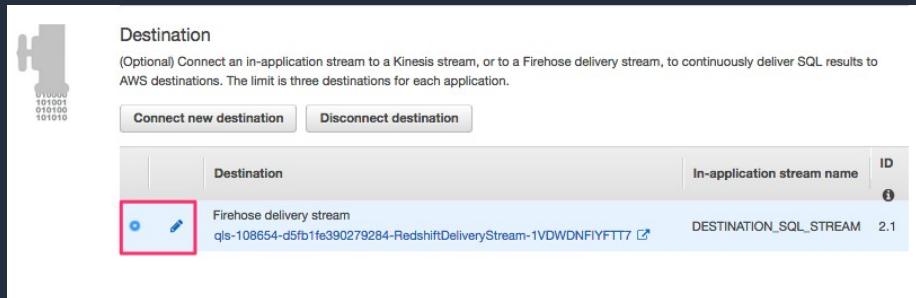
# Activity 3B: Create table in Amazon Redshift

- Make sure you are in the logs database in pgweb
- Create table weblogs to capture incoming data from Kinesis Data Firehose delivery stream
- Use RedshiftSQL.txt from Workshop Scripts for the CREATE TABLE statement
- Paste in the CREATE TABLE SQL statement and click “Run Query” to create the Redshift table

```
1 --DROP TABLE weblogs;
2 CREATE TABLE weblogs
3 ▼ (
4     row_time timestamp encode raw,
5     host_address varchar(512) encode lzo,
6     request_time timestamp encode raw,
7     request_method varchar(5) encode lzo,
8     request_path varchar(1024) encode lzo,
9     request_protocol varchar(10) encode lzo,
10    response_code int encode delta,
11    response_size int encode delta,
12    referrer_host varchar(1024) encode lzo,
13    user_agent varchar(512) encode lzo
14 ) DISTSTYLE EVEN
15 SORTKEY (request_time);|
```

# Activity 3C: Deliver data to Amazon Redshift using Data Firehose

- Update Kinesis Data Analytics application to send data to Kinesis Data Firehose delivery stream. Data Firehose delivers the streaming data to Amazon Redshift.
  1. Go to the **Kinesis Data Analytics console**. Go to Application Details.
  2. Choose the Amazon Redshift delivery stream as destination and click on the edit button (see the pencil icon in the figure below).



# Activity 3C: Deliver data to Amazon Redshift using Data Firehose

- Validate your destination
  - Validate that the Kinesis Data Firehose stream is “<stackName>RedshiftStack-xxx-**RedshiftDeliveryStream**-xxxxxxxx”
  - Keep the default for “Choose an existing in-application stream”. DESTINATION\_SQL\_STREAM
  - Make sure CSV is the “Output format”
  - Validate that “Choose from IAM roles that Kinesis Data Analytics can assume”
  - Click “Save and continue”
- It will take about 1–2 minutes for everything to be updated and for data to start appearing in Amazon Redshift

# Activity 3C: Deliver data to Amazon Redshift using Data Firehose

The screenshot shows the configuration page for connecting an in-application stream to a Kinesis Analytics application. The stream selected is a 'Firehose delivery stream' named 'qls-108752-f84000a865bde39d-RedshiftDeliveryStream-134WPEAWMAT75'. The 'In-application stream' section is active, showing options to 'Choose an existing in-application stream' or 'Specify a new in-application stream name'. A note explains that using the existing stream ensures data persistence. The 'In-application stream name' is set to 'DESTINATION\_SQL\_STREAM'. The 'Output format' is set to 'CSV'. In the 'Access to chosen resources' section, the 'Choose from IAM roles that Kinesis Analytics can assume' option is selected. The 'IAM role' dropdown shows 'qls-108752-f84000a865bde39d-KinesisAnalyti...'.

Filter by stream name or stream type

Stream name	Stream type
qls-108752-f84000a865bde39d-RedshiftDeliveryStream-134WPEAWMAT75	Firehose delivery stream

In-application stream

In-application streams are continuous flows of data records. You create in-application streams in SQL to contain the data you want to persist to the specified destination. [Learn more](#).

Connect in-application stream

Choose an existing in-application stream  
 Specify a new in-application stream name

Use this option for in-application streams that you haven't created yet, but plan to create at a later time. Specifying a stream name ensures that you don't lose output data.

In-application stream name\* DESTINATION\_SQL\_STREAM

Output format

JSON  
 CSV

Access to chosen resources

Create or choose IAM role with the required permissions. [Learn more](#)

Access to chosen resources\*

Create / update IAM role kinesis-analytics-qls-108752-f84000a865bde39d-KinesisA-us-west-2  
 Choose from IAM roles that Kinesis Analytics can assume

IAM role\* qls-108752-f84000a865bde39d-KinesisAnalyti... [View role in the IAM console](#)

# Review: Amazon Redshift test queries

- Find distribution of response codes over days (copy SQL from RedshiftSQL file)

```
-- find distribution of response codes over days
SELECT TRUNC(request_time), response_code, COUNT(1)
FROM weblogs
GROUP BY 1,2
ORDER BY 1,3 DESC;
```

- Count the number of 404 response codes

```
-- find distribution of response codes over days
SELECT TRUNC(request_time), response_code, COUNT(1)
FROM weblogs
GROUP BY 1,2
ORDER BY 1,3 DESC;
```

# Review: Amazon Redshift test queries

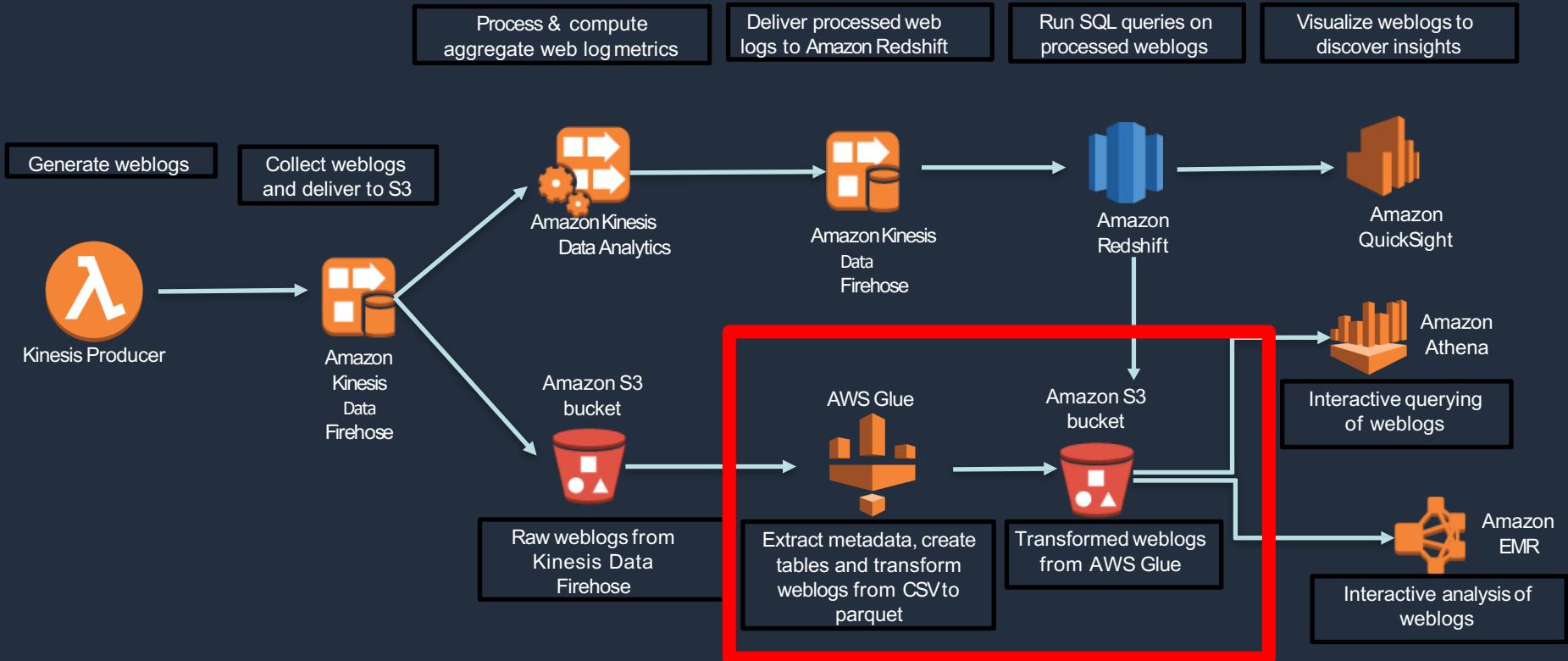
- Show all requests paths with status “PAGE NOT FOUND”

```
-- show all requests for status as PAGE NOT FOUND
SELECT TOP 1 request_path, COUNT(1)
FROM weblogs
WHERE response_code = 404
GROUP BY 1
ORDER BY 2 DESC;
```

# Activity 4

## Transform weblogs to parquet format using AWS Glue

# Your application architecture



# Activity: Catalog and perform ETL on weblogs

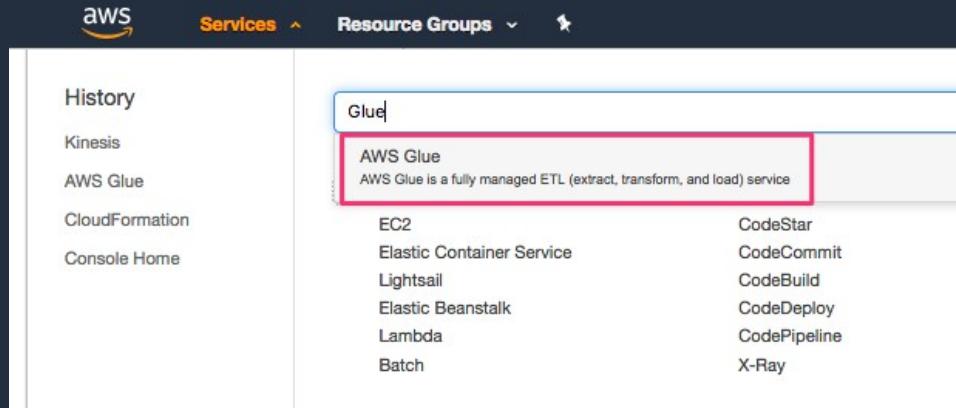
Time: 30 minutes

We are going to:

- A. Discover and catalog the weblog data deposited into the S3 bucket using AWS Glue crawler
- B. Transform weblogs to parquet format using the AWS Glue ETL job authoring tool

# Activity 4A: Discover dataset with AWS Glue

- We use AWS GLUE's crawler to extract data and metadata. From the AWS Management Console, select AWS Glue. Click on “**Get Started**” on the next screen.



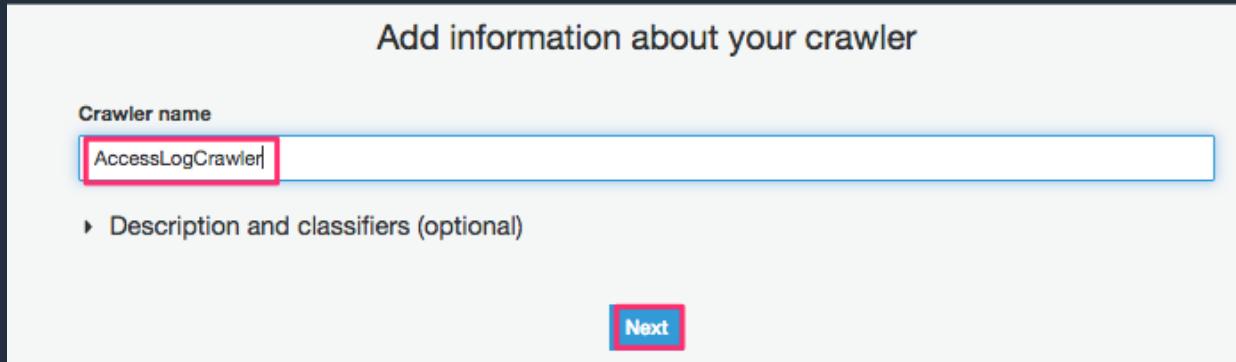
# Activity 4A: Add crawler using AWS Glue

- Select "Crawlers" section on the left and click on "Add crawler"

The screenshot shows the AWS Glue Data Catalog interface. On the left, there is a sidebar with options: Databases, Tables, Connections, **Crawlers** (which is selected and highlighted with a red box), and Classifiers. The main content area has a title "Crawlers" with a subtitle: "A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog." Below this are three buttons: "Add crawler" (highlighted with a red box), "Run crawler", and "Action". A table header row includes columns for Name, Schedule, Status, Logs, Last runtime, Median runtime, Tables updated, and Tables added. A message in the center states "You don't have any crawlers yet." with a "Add crawler" button below it.

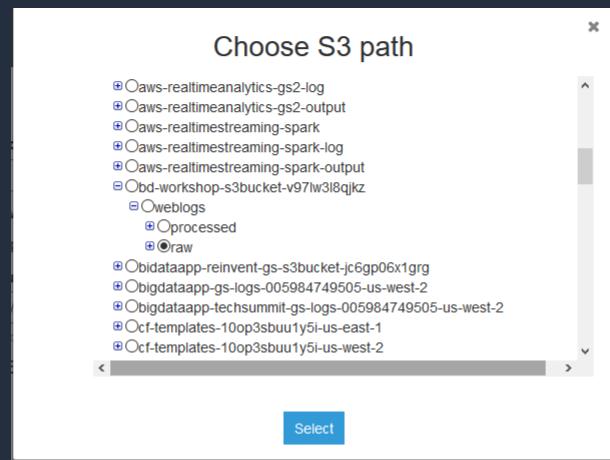
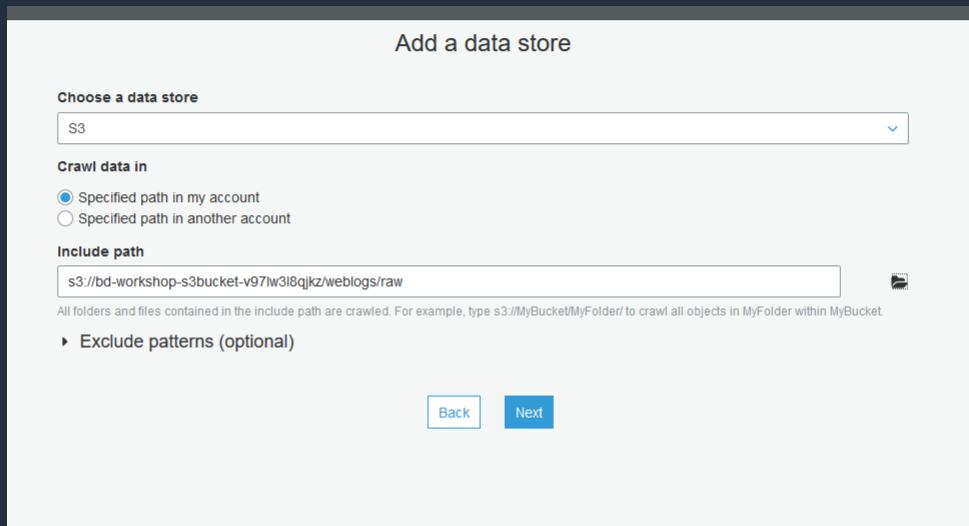
# Activity 4A: ETL with AWS Glue

- Specify a name for the crawler. Click "Next"



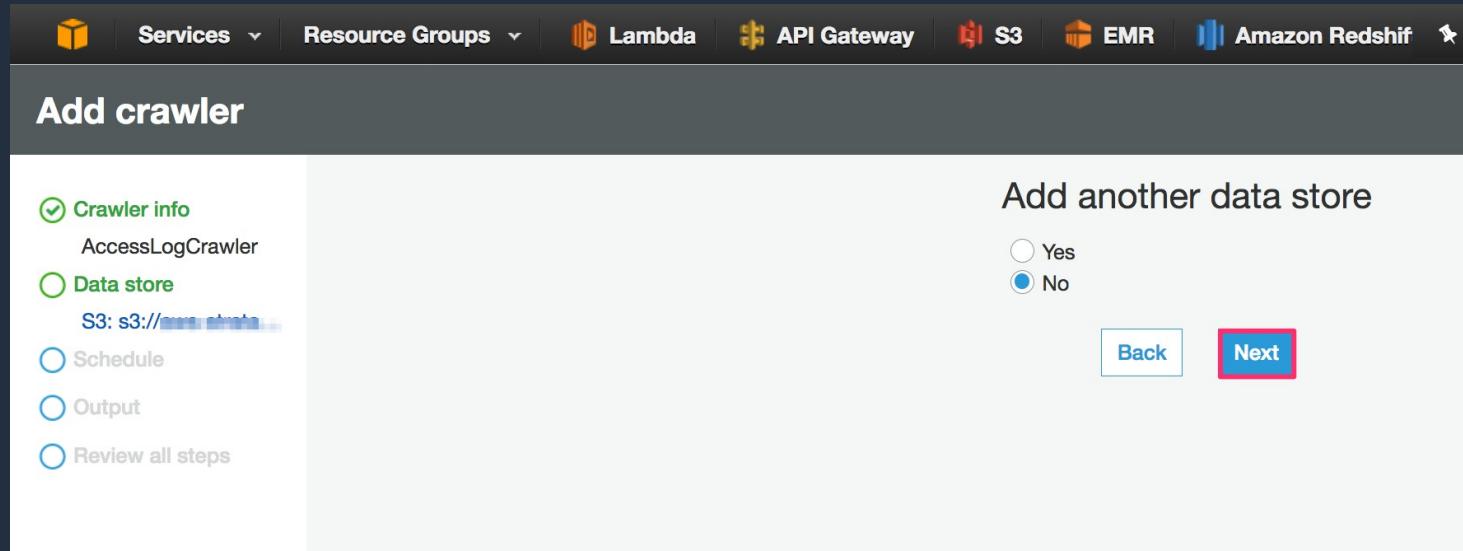
# Activity 4A: ETL with AWS Glue

- Provide S3 path location where the raw weblogs were placed (navigate to S3 path: s3://<S3 bucket from template>/weblogs/raw)
- Click "Next"



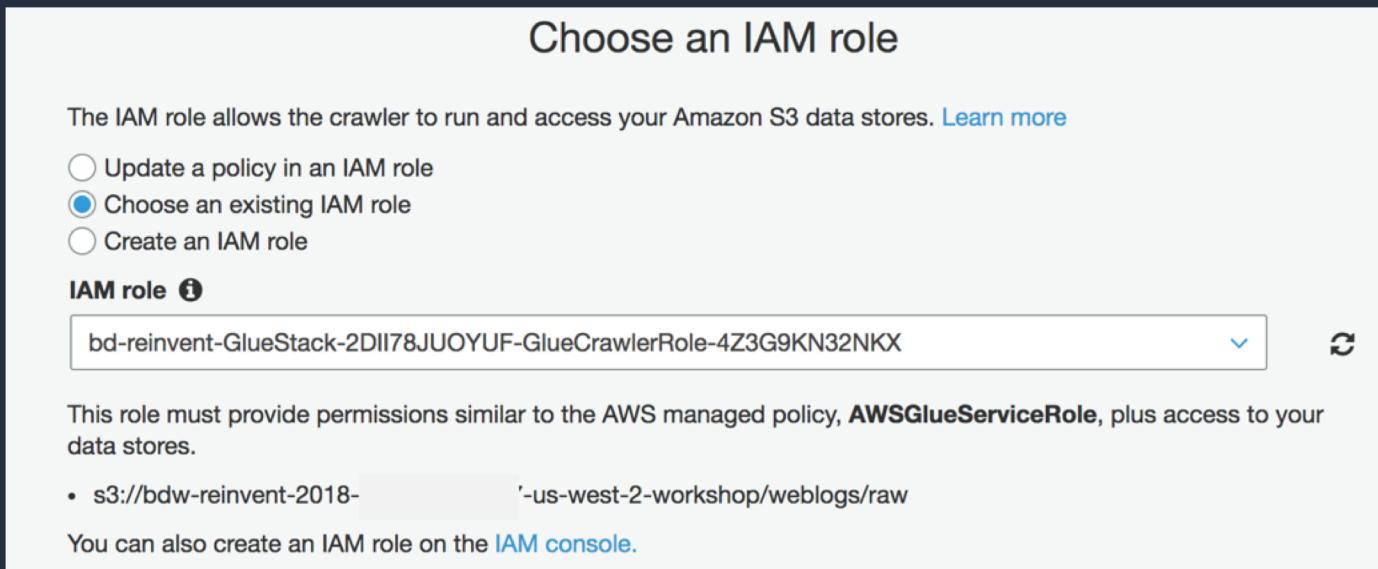
# Activity 4A: ETL with AWS Glue

- Click "Next" on the next screen to not add another data store



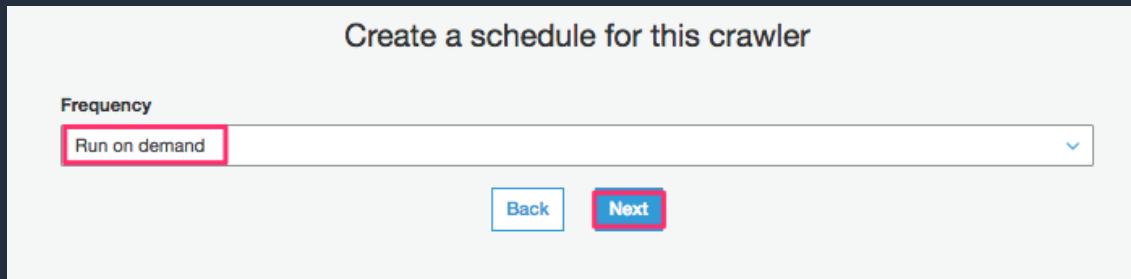
# Activity 4A: ETL with AWS Glue

- In the IAM role section, select “Choose an existing IAM role”
- Select role <StackName>-GlueStack-xxx-GlueCrawlerRole-xxxxxxx as the IAM role and click “Next”



# Activity 4A: Add crawler with AWS Glue

- Choose "Run on demand" to run the crawler now, and click "Next"



# Activity 4A: Add crawler with AWS Glue

- On the next screen, drop down Database and select “weblogs\_dev”
- Click “Next”

Configure the crawler's output

**Database** ⓘ

weblogs\_dev

Add database

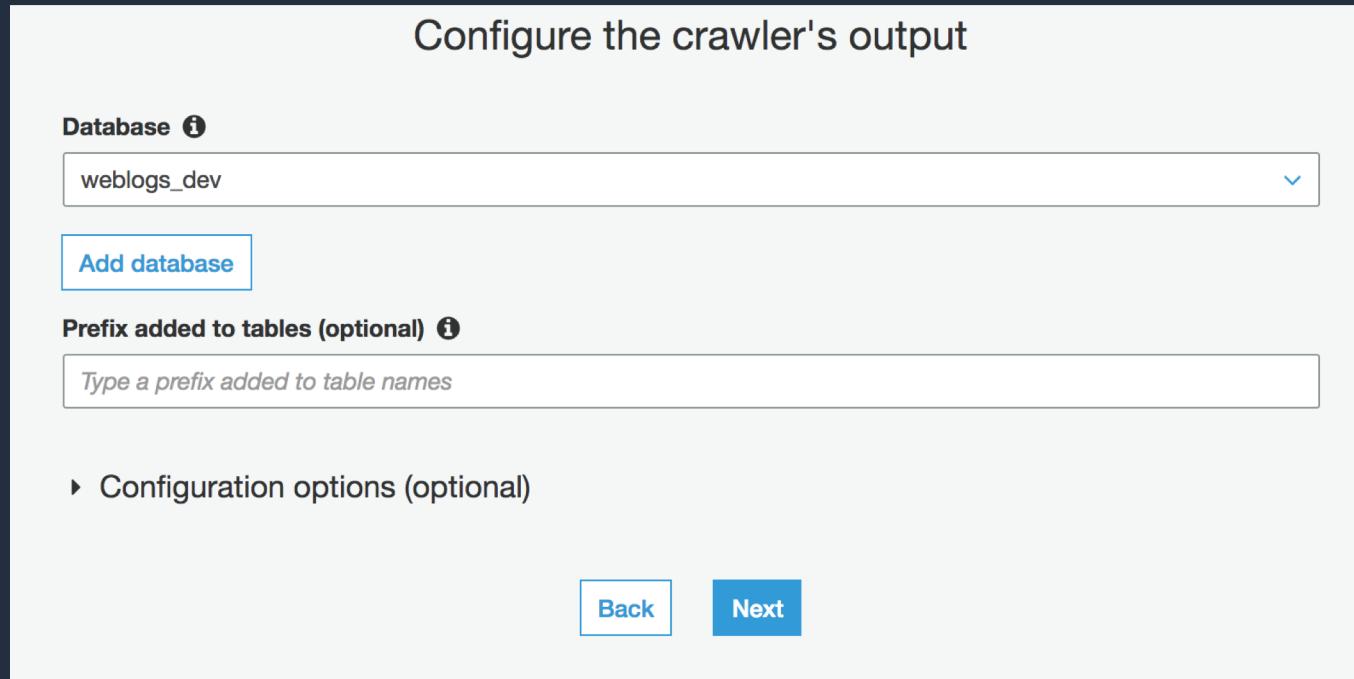
**Prefix added to tables (optional)** ⓘ

Type a prefix added to table names

▶ Configuration options (optional)

Back

Next



# Activity 4A: Add crawler with AWS Glue

- Review and click "Finish" to create a crawler

The screenshot shows the AWS Glue Crawler configuration wizard with the following details:

- Crawler info:** Name: AccessLogCrawler
- Data stores:** Data store: S3, Include path: s3://qls-...-logs-...-us-west-2, Exclude patterns: None
- IAM role:** IAM role: arn:aws:iam::...:role/service-role/AWSGlueServiceRole-default
- Schedule:** Schedule: Run on demand
- Output:** Database: weblogdb, Prefix added to tables (optional): None, Schema change policy: None

At the bottom right, there are "Back" and "Finish" buttons, with "Finish" being highlighted.

# Activity 4A: Add crawler with AWS Glue

- Click on "Run it now?" link to run the crawler

The screenshot shows the AWS Glue interface with the 'Crawlers' section selected in the sidebar. The main area displays a message about a newly created crawler, followed by a table listing existing crawlers.

**Crawlers**  
A crawler connects to a data store, progresses through a prioritized list of classifiers to determine the schema for your data, and then creates metadata tables in your data catalog.

Crawler **AccessLogCrawler** was created to run on demand. [Run it now?](#) X

	Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
<input type="checkbox"/>	AccessLogCrawler		Ready		0 secs	0 secs	0	0

# Activity 4A: Add crawler with AWS Glue

- Crawler shows a **Ready** status when it is finished running

The screenshot shows the AWS Glue interface with the 'Crawlers' section selected. A success message indicates that the 'AccessLogCrawler' completed its run. The crawler's status is listed as 'Ready'.

Name	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
AccessLogCrawler	Ready	Logs	33 secs	33 secs	0	1

# Activity 4A: Table creation in AWS Glue

- Observe that the crawler has created a table for your dataset
- The crawler automatically classified the dataset as **combinedapache log** format
- Click the table to take a look at the properties

The screenshot shows the AWS Glue Data Catalog Tables page. On the left, there's a sidebar with navigation links: AWS Glue (selected), Data catalog, Databases, Tables (highlighted with a red box), Connections, Crawlers, and Classifiers. The main area has a title 'Tables' with 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 is a search bar with 'Add tables' and 'Action' buttons, a filter bar with a search icon and 'Save view' button, and a table header with columns: Name, Database, Location, Classification, Last updated, and Deprecated. A single row is listed: Name is 'raw', Database is ' weblogdb', Location is 's3://qls-108752-f84000a865bde39d-logs-408270358...', Classification is 'combinedapache' (highlighted with a red box), Last updated is '23 November 2017 10:0...', and Deprecated is 'false'.

Name	Database	Location	Classification	Last updated	Deprecated
raw	weblogdb	s3://qls-108752-f84000a865bde39d-logs-408270358...	combinedapache	23 November 2017 10:0...	false

# Activity 4A: Table creation in AWS Glue

- AWS Glue used the **GrokSerDe** (Serializer/Deserializer) to correctly interpret the weblogs
- You can click on the "**View partitions**" link to look at the partitions in the dataset

The screenshot shows the AWS Glue Table configuration interface. At the top, there are buttons for "Edit table" and "Delete table". On the right, there are buttons for "View partitions" (which is highlighted with a red box), "Compare versions", and "Edit schema".

**Table Details:**

- Name: raw
- Description: weblogdb
- Database: weblogdb
- Classification: combinedapache
- Location: s3://qls-108752-...-logs-408270358361-us-west-2/weblogs/raw/
- Connection: No
- DDeprecated: No
- Last updated: Thu Nov 23 10:04:56 GMT-500 2017
- Input format: org.apache.hadoop.mapred.TextInputFormat
- Output format: org.apache.hadoop.hive.io.HiveIgnoreKeyTextOutputFormat
- Serde serialization lib: com.amazonaws.glue.serde.GrokSerDe
- Serde parameters: input.format: %{COMBINEDAPACHELOG}

**Table Properties:**

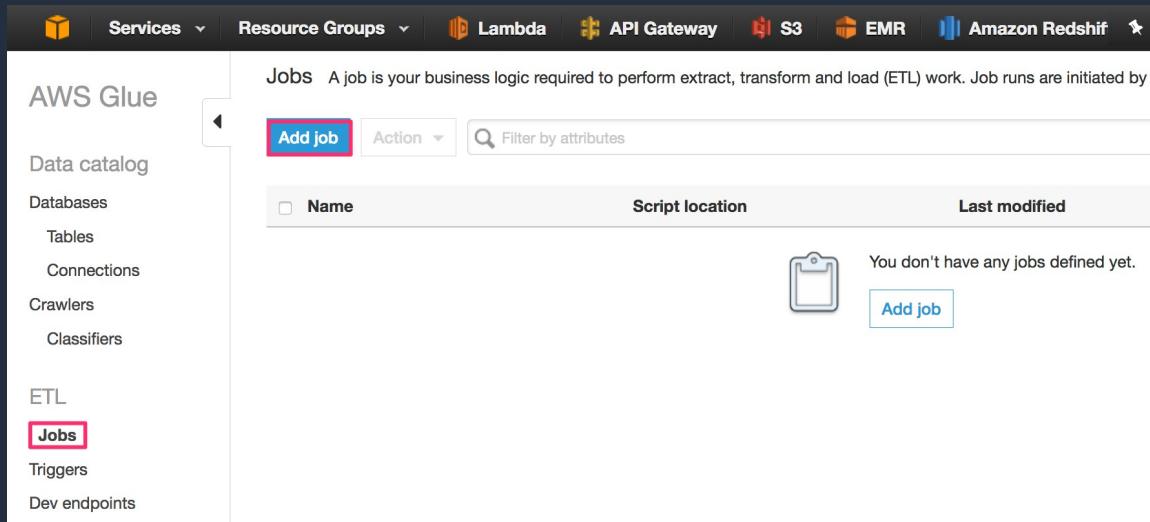
- sizeKey: 3319664
- objectCount: 1
- UPDATED\_BY\_CRAWLER: AccessLogCrawler
- CrawlerSchemaVersion: 1.0
- recordCount: 897
- averageRecordSize: 184
- grokPattern: %{COMBINEDAPACHELOG}
- CrawlerSchemaDeserializerVersion: 1.0
- compressionType: gzip
- typeOfData: file

**Schema:**

Column name	Data type	Key
1	clientip	string
2	ident	string

# Activity 4A: Create ETL job in AWS Glue

- With the dataset cataloged and table created, we are now ready to convert the weblogs-based Apache combined log format to a more optimal parquet format for querying.
- Click on "Add job" to begin creating the ETL job



# Activity 4B: ETL job in AWS Glue

- Job name: **accesslogetl1**
- Select the IAM role <StackName>-GlueStack-xxx-GlueJobRole-xxxxxxxx
- Select Type as Spark
- Select “a new script to be authored by you”  
Script file name: **glue-workshop-etl.py**
- For the S3 path where the script will be stored, use path **s3://<S3 bucket from template>/script**
- For the Temporary Directory, use path **s3://<S3 bucket from template>/temp**
- **DO NOT CLICK NEXT JUST YET**

Configure the job properties

Name

IAM role ?

Ensure that this role has permission to your Amazon S3 sources, targets, temporary directory, scripts, and any libraries used by the job. [Create IAM role](#).

Type  
 Spark

This job runs

A proposed script generated by AWS Glue ?  
 An existing script that you provide  
 A new script to be authored by you

ETL language  
 Python  Scala

Script file name

S3 path where the script is stored

Temporary directory ?

# Activity 4B: ETL job in AWS Glue

- Expand **Script libraries and job parameters** section, and set the **Maximum capacity (DPUs)** to 10.
- Let's pass a job parameter to send the S3 path where parquet files will be deposited.
- Specify the following values for Key and Value
- **Key:** `--parquet_path` (notice the 2 hyphens at the beginning and underscore between “parquet” and “path”)
- **Value:** `s3://<S3 bucket from template>/weblogs/parquet`
- **Note:** Value is the S3 path we stored from the previous slide

The screenshot shows the AWS Glue Job Editor interface. It includes fields for Maximum capacity (0.0625), Max concurrency (1), Job timeout (2880 minutes), Delay notification threshold (empty), Number of retries (0), and a Job parameters section. The Job parameters section contains a key-value pair: --parquet\_path with value s3://bd-workshop-s3bucket-. Below this, there are input fields for Type key... and Type value... with an 'x' button to remove the entry. At the bottom, there is a Catalog options (optional) section and a blue Next button.

Maximum capacity ⓘ  
0.0625

Max concurrency ⓘ  
1

Job timeout (minutes) ⓘ  
2880

Delay notification threshold (minutes) ⓘ

Number of retries  
0

Job parameters

Key	Value
--parquet_path	s3://bd-workshop-s3bucket-

Type key... Type value... x

▶ Catalog options (optional)

**Next**

# Activity 4B: ETL job in AWS Glue

- Click "**Next**" in the following screen
- Review and click "**Save Job and Edit Script**" to create the job

The screenshot shows the AWS Glue job configuration interface. The top navigation bar has tabs: History, Details (which is selected), Script, and Metrics. The main content area displays various job settings:

Name	accesslogetl1	Python lib path	-
IAM role	bd-workshop-GlueStack-9XZW7Z0CQ9N1-GlueJobRole-L5ZR3J8JA4OX	Jar lib path	-
ETL language	python	Other lib path	-
Script location	s3://bd-workshop-s3bucket-v97lw3l8qjkz/script/glue-workshop-etl.py	Parameters	--parquet_path s3://bd-workshop-s3bucket-v97lw3l8qjkz/weblogs.parquet
Temporary directory	s3://bd-workshop-s3bucket-v97lw3l8qjkz/temp	Connections	-
Job bookmark	Disable	DPU	20
Job metrics	Disable	Job timeout (minutes)	2880
Server-side encryption	Disabled	Delay notification threshold (minutes)	-

Below the settings, there is a section titled "Automatically run this job if any of the following triggers fire:" which contains a table:

Trigger name	Trigger type	Trigger status	Trigger parameters	Jobs to trigger
No triggers start this job				

# Activity 4B: ETL job in AWS Glue

```
1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7 from pyspark.sql.functions import *
8 from awsglue.dynamicframe import DynamicFrame
9
10
11 ## @params: [JOB_NAME]
12 args = getResolvedOptions(sys.argv, ['JOB_NAME', 'parquet_path'])
13
14 # Create Glue Context and spark session
15 sc = SparkContext()
16 glueContext = GlueContext(sc)
17 spark = glueContext.spark_session
18 job = Job(glueContext)
19 job.init(args['JOB_NAME'], args)
20
21 # Input: Database and table name from Catalog
22 db_name = " weblog_dev"
23 table_name = " raw"
24
25 # Output: S3 and temp directories
26 parquet_output_path = args['parquet_path']
27
28 # Create dynamic frame from catalog
29 datasource0 = glueContext.create_dynamic_frame.from_catalog(database = db_name, table_name = table_name, transformation_ctx = "datasource0")
30
31 # Convert to Spark DataFrame
32 df = datasource0.toDF()
33
34 new_df = df.select(df.clientip, df.ident, df.auth, df.timestamp, df.verb, df.request, df.httpversion, df.response, df.bytes, df.referrer, df.agent)
35
36 # Cast the datetime column to timestamp
37 ts_df = new_df.withColumn("dt", unix_timestamp(df.timestamp, "yyyy-MM-dd HH:mm:ss").cast("timestamp").cast("double").cast("timestamp")).drop(df.timestamp)
38 ts_df = new_df.withColumn("dt", unix_timestamp(df.timestamp, "yyyy-MM-dd HH:mm:ss").cast("timestamp")).drop(df.timestamp)
39
```

- Close script editor tips window (if it appears).
- In the AWS Glue script editor, copy the ETL code in glue-workshop-etl.py from BigDataWorkshop.zip and paste. Overwrite; don't append.
- Ensure that the db\_name and table\_name statements reflect the database and table name created by the AWS Glue crawler.

# Activity 4B: ETL job in AWS Glue

- Click "Save" and then "Run job" button to execute your ETL

Job: accesslogetl Action ▾ Save Run job Generate diagram ? Insert template at cursor ? Source

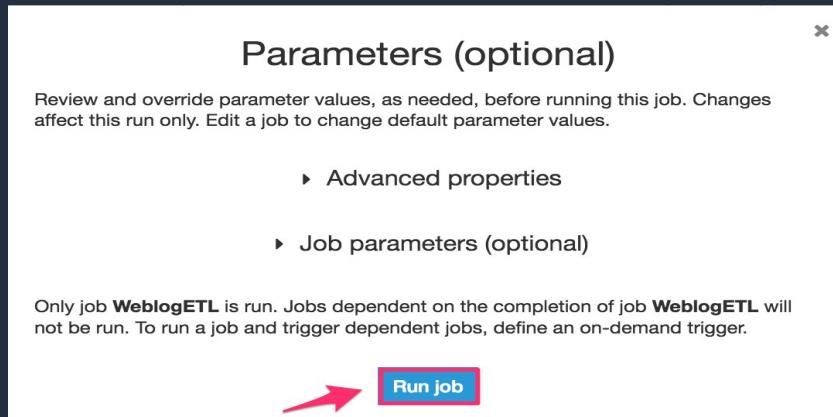
The diagram could not be generated. Check the annotations in your script.

[+] [-] ⓘ

```
1 import sys
2 from awsglue.transforms import *
3 from awsglue.utils import getResolvedOptions
4 from pyspark.context import SparkContext
5 from awsglue.context import GlueContext
6 from awsglue.job import Job
7 from pyspark.sql.functions import *
8 from awsglue.dynamicframe import DynamicFrame
9
10
11 ## @params: [JOB_NAME]
12 args = getResolvedOptions(sys.argv, ['JOB_NAME', 'parquet_path'])
13
14 # Create Glue Context and spark session
15 sc = SparkContext()
16 glueContext = GlueContext(sc)
17 spark = glueContext.spark_session
18 job = Job(glueContext)
19 job.init(args['JOB_NAME'], args)
20
```

# Activity 4B: ETL job in AWS Glue

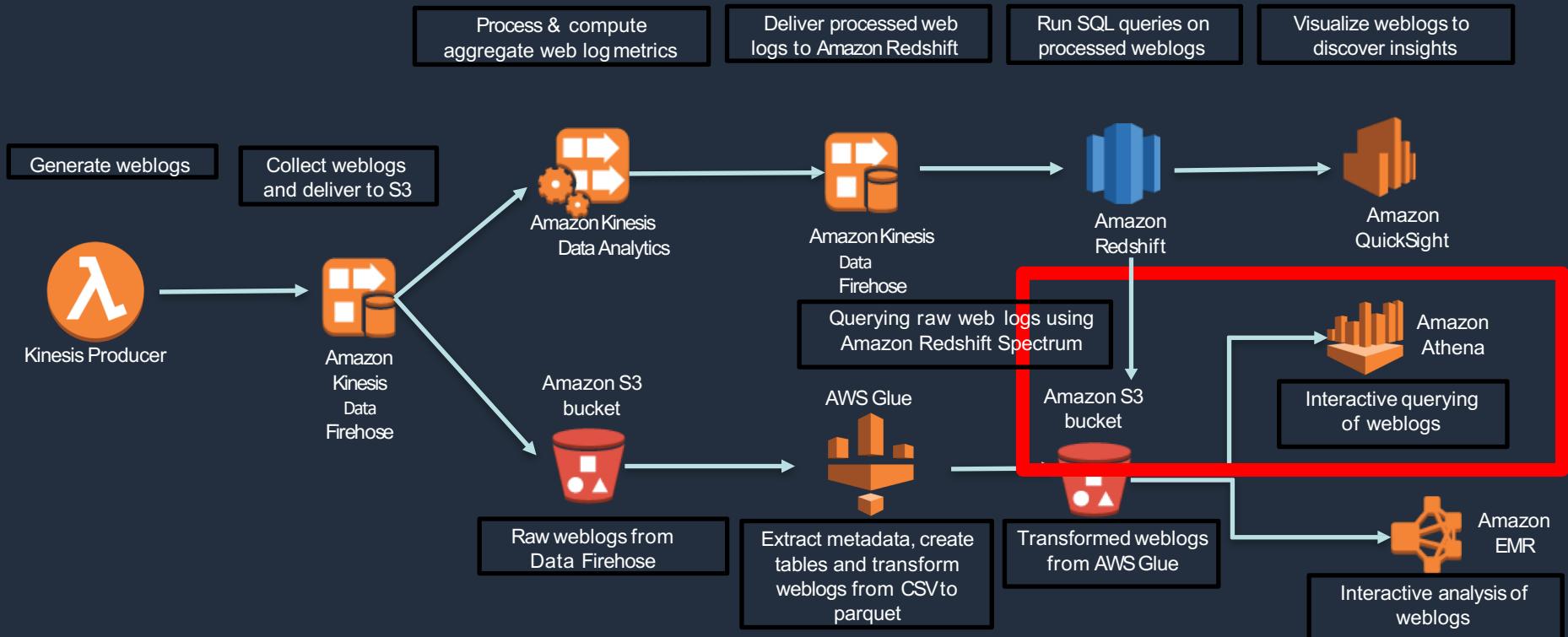
- Check that the --parquet\_path is correctly set in the job parameters.
- Check that the script name is glue-workshop-etl.py
- Click "Run job" to continue. This might take a few minutes. When the job finishes, weblogs will be transformed to parquet format.
- Go to s3://bd-workshop-s3bucket-xxxxxxxxxx/weblogs/parquet and verify that you see parquet.snappy files in there.



# Activity 5

## Amazon Redshift Spectrum and interactive querying with Amazon Athena

# Your application architecture



# Activity: Querying data in Amazon S3 using Redshift Spectrum

Time: 30 minutes

We are going to:

- A. Create a table over the processed weblogs in Amazon S3 using an AWS Glue crawler. These are the parquet files created by AWS Glue ETL job in the previous section.
- B. Run queries from Amazon Redshift on the parquet weblogs in Amazon S3 using Amazon Redshift Spectrum.
- C. Run interactive queries from Athena on parquet weblogs in S3.

# Activity 5A: Set up AWS Glue crawler for processed parquet data

- Go to Crawlers in AWS Glue and search for ParquetLogs.
- Select the ParquetLogsCrawler-xxxxxxx. Go to details tab and check that it's pointing to the right parquet location in Amazon S3.
- Run crawler. Should take about 30 secs for the crawler to finish running.

The screenshot shows the AWS Glue Crawler list page. A success message at the top states: "Crawler 'ParquetLogsCrawler-CejAbrANPNW4' completed and made the following changes: 1 tables created, 0 tables updated. See the tables created in database [weblogs\\_dev](#)". Below this, there are buttons for "Add crawler", "Run crawler", and "Action". A search bar is labeled "Name : ParquetLogs". The main table lists one crawler:

<input checked="" type="checkbox"/>	Name	Schedule	Status	Logs	Last runtime	Median runtime	Tables updated	Tables added
<input checked="" type="checkbox"/>	ParquetLogsCrawler-CejAbrANPN...		Ready	<a href="#">Logs</a>	29 secs	29 secs	0	1

At the bottom right of the table, there are links for "Showing: 1 - 1" and icons for "User preferences", "Edit", and "Delete".

# Activity 5A: Set up AWS Glue crawler for processed parquet data

- Navigate to Databases-Tables in AWS Glue. Search weblogs\_<environment>.
- Make sure you see both the “raw” and the newly created “parquet” table.

Tables 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.

The screenshot shows the AWS Glue 'Tables' list interface. At the top, there are buttons for 'Add tables', 'Action', a search bar containing 'search : weblogs\_dev', and a 'Save view' button. To the right, it says 'Showing: 1 - 2'. Below the header is a table with the following data:

Name	Database	Location	Classification	Last updated	Deprecated
parquet	weblogs_dev	s3://bd-workshop-s3bucket-v97lw3l8qjzk/weblogs/parquet/	parquet	26 October 2018 4:58 PM UTC-7	
raw	weblogs_dev	s3://bd-workshop-s3bucket-v97lw3l8qjzk/weblogs/raw/	combinedapache	26 October 2018 3:25 PM UTC-7	

# Activity 5A: Set up AWS Glue crawler for parquet data

- Take a look at the schema of the parquet table
- Should have eight columns
- Classification should be parquet

The screenshot shows the AWS Glue Table configuration interface for a table named "parquet".

**Table Details:**

- Name: parquet
- Description:
- Database: weblogs\_dev
- Classification: parquet
- Location: s3://bd-workshop-s3bucket-v97lw3l8qjz/weblogs/parquet/
- Connection:
- Deprecated: No
- Last updated: Fri Oct 26 16:58:48 GMT-700 2018
- Input format: org.apache.hadoop.hive.io.parquet.MapredParquetInputFormat
- Output format: org.apache.hadoop.hive.io.parquet.MapredParquetOutputFormat
- Serde serialization lib: org.apache.hadoop.hive.io.parquet.serde.ParquetHiveSerDe
- Serde parameters: serialization.format: 1

**Table Properties:**

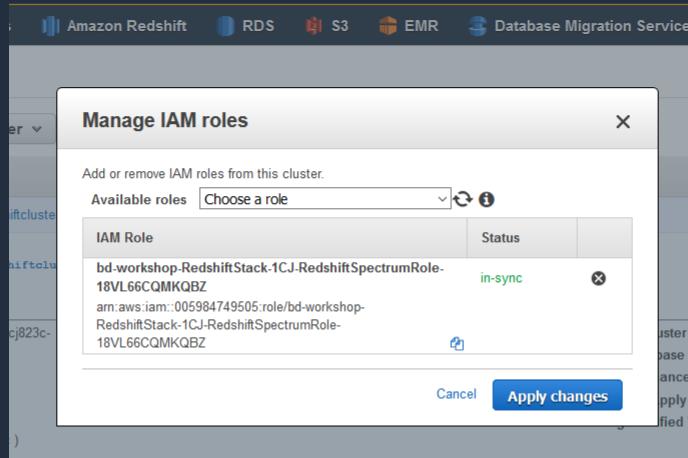
sizeKey	5960173	objectCount	132	UPDATED_BY_CRAWLER	ParquetLogsCrawler-CejAbrANPNW4	CrawlerSchemaSerializerVersion	1.0	recordCount	147821	averageRecordSize	112
CrawlerSchemaDeserializerVersion	1.0	compressionType	none	typeOfData	file						

**Schema:**

	Column name	Data type	Partition key	Comment
1	clientip	string		
2	verb	string		
3	request	string		
4	httpversion	string		
5	agent	string		
6	dt	timestamp		

# Activity 5B: Query using Amazon Redshift Spectrum

- Navigate to Amazon Redshift-Clusters. Select the cluster with the <stack-name>
- Click on “Manage IAM roles”.
- Validate that IAM role shows bd-workshop-RedshiftStack-xxx-RedshiftSpectrumRole-xxxxxxxxxxxx.



# Activity 5B: Query using Amazon Redshift Spectrum

- Navigate to pgWeb. If you need the URL, check public IP of the pgweb EC2.
- Get the RedshiftSpectrumSetup.sql from the BigDataWorkshop.zip.
  - Make sure you replace the role ARN with your SpectrumRole ARN.
  - Make sure you are pointing to the weblogs\_dev database.
  - This will create an external schema in Amazon Redshift called “Spectrum”.

The screenshot shows the pgWeb interface with the following details:

- Logs** tab is selected.
- SQL Query** tab is also visible.
- public** schema is expanded, showing:
  - Tables (1)**: A single table named `spectrum`.
  - weblogs**: A folder containing:
    - Views (0)**
    - Materialized Views (0)**
    - Sequences (0)**
- SQL Query** pane displays the following code:

```
1 create external schema spectrum
2   from data catalog
3   database 'weblogs_dev'
4   iam_role 'arn:aws:iam::005984749505:role/bd-workshop-RedshiftStack-1CJ-RedshiftSpectrumRole-18VL66CQWKBZ'
5   create external database if not exists;
```
- Buttons**: `Run Query` and `Explain Query`.
- Message**: `No records found`.

# Activity 5B: Query using Amazon Redshift Spectrum

- Let's run a couple of simple queries against Amazon S3 from Amazon Redshift using Amazon Redshift Spectrum
  - Count of all records in the parquet location in Amazon S3
  - Select count(\*) from spectrum.parquet
  - Count of all records in the parquet location in Amazon S3 where the response is 404
  - Select count(1) from spectrum.parquet where response=404;

The screenshot shows the SQL Query interface with the following details:

- SQL Query tab is selected.
- Query text:

```
1 select count(*) from spectrum.parquet
2
```
- Result pane:

count
157400
- Buttons: Run Query, Explain Query.

The screenshot shows the SQL Query interface with the following details:

- SQL Query tab is selected.
- Query text:

```
1 SELECT COUNT(1)
2 FROM spectrum.parquet
3 WHERE response = 404;
4
```
- Result pane:

count
6400
- Buttons: Run Query, Explain Query.

# Activity 5C: Querying using Amazon Athena

- Go to the Athena console. Close the tutorial that comes up.
- In the Athena console, choose the "**weblogs\_<environment>**" on the database dropdown.
- Select "**parquet**" from the tables section and click on the **three stacked dots** icon to preview/sample a few rows of the S3 data.

The screenshot shows the Amazon Athena Query Editor interface. The top navigation bar includes links for Athena, Query Editor, Saved Queries, History, AWS Glue Data Catalog, Settings, Tutorial, Help, and What's new. The main area is divided into several sections:

- Database:** A dropdown menu set to "weblogs\_dev".
- Tables:** Shows two tables: "parquet" and "raw (Partitioned)".
- Views:** Shows 0 views.
- Query Editor:** Displays a query in the "New query 1" tab:

```
1 SELECT * FROM "weblogs_dev"."parquet" limit 10;
```

A context menu is open over this query, listing options: Preview table, Show properties, Delete table, and Generate Create Table DDL.
- Run Query:** Buttons for Run query, Save as, and Create. Below these, it says "(Run time: 2.06 seconds, Data scanned: 204.59KB)" and provides keyboard shortcuts: Use Ctrl + Enter to run query, Ctrl + Space to autocomplete.
- Results:** A table showing the first 10 rows of the "parquet" table. The columns are clientip, verb, request, httpversion, agent, dt, response, and by. The data includes:clientip verb request httpversion agent dt response by  
1 192.0.4.19 DELETE /wp-content 1.0 Opera/9.72.(Windows NT 6.2; nds-NL)Presto/2.9.174 Version/10.00 2018-10-26 00:00:00.000 200 50  
2 181.212.240.57 PUT /search/tag/list 1.0 Mozilla/5.0 (Windows; U; Windows CE) AppleWebKit/535.43.6 (KHTML, like Gecko) Version/5.0.5 Safari/535.43.6 2018-10-26 00:00:00.000 500 50  
3 192.31.220.149 GET /posts/posts/explore 1.0 Opera/8.84.(X11; Linux x86\_64; ts-ZA)Presto/2.9.184 Version/10.00 2018-10-26 00:00:00.000 301 49

# Activity 5C: Querying using Amazon Athena

- Athena allows you to run interactive queries against parquet data in Amazon S3
- Run the queries from the AthenaSQL from the BigDataWorkshop.zip
  - Count of each clientIP
  - Count of each response code

The screenshot shows the AWS Athena Query Editor. The top navigation bar includes 'Athena', 'Query Editor' (which is selected), 'Saved Queries', 'History', and 'AWS Glue Data Catalog'. The left sidebar shows the 'Database' set to 'weblogs\_dev' and lists 'Tables (2)' including 'parquet' and 'raw (Partitioned)'. It also shows 'Views (0)'. The main area displays a 'New query 1' window with the following SQL code:

```
1 SELECT response, COUNT(1) FROM weblogs_dev.parquet GROUP BY response;
```

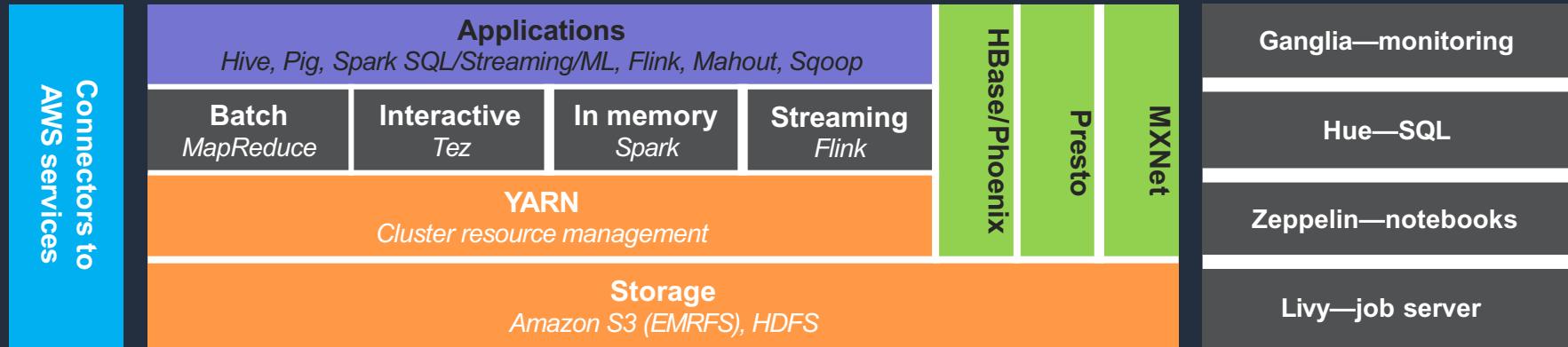
Below the code are buttons for 'Run query', 'Save as', and 'Create'. A status message indicates '(Run time: 5 seconds, Data scanned: 46.8KB)'. The bottom section is titled 'Results' and shows a table with four rows:

response	_col1
1	500
2	200
3	301
4	404

Congratulations!  
and  
Thank you for attending !

# Optional exercise: Data processing with Amazon EMR

# Amazon EMR service



# On-cluster UIs



SQL editor, workflow designer,  
metastore browser



Notebooks

**Design and execute  
queries and workloads**



**Manage applications**



And more using  
bootstrap actions!

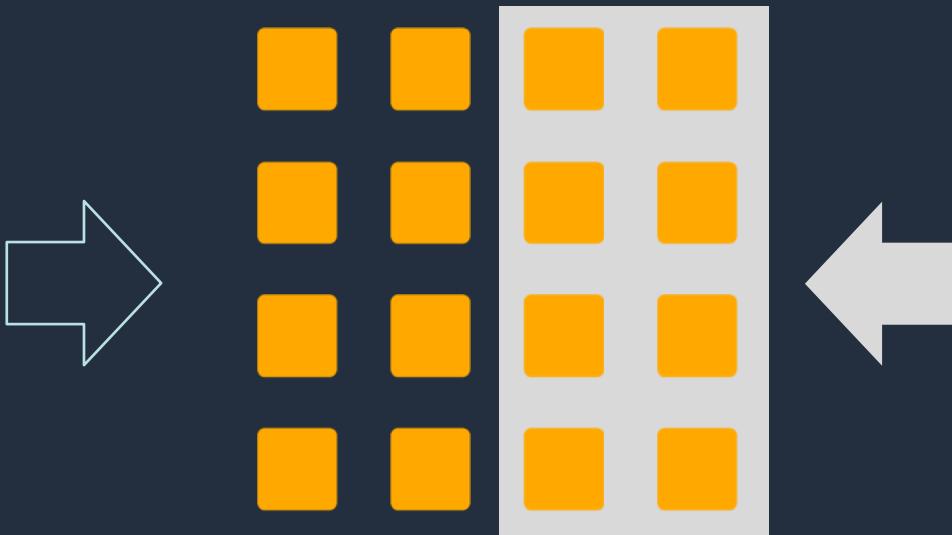
# The Hadoop ecosystem can run in Amazon EMR



# Easy-to-use Amazon EC2 Spot Instances

- Meet SLA at predictable cost
  - On-demand for core nodes
    - Standard Amazon Elastic Compute Cloud (Amazon EC2) pricing for on-demand capacity

Exceed SLA at lower cost

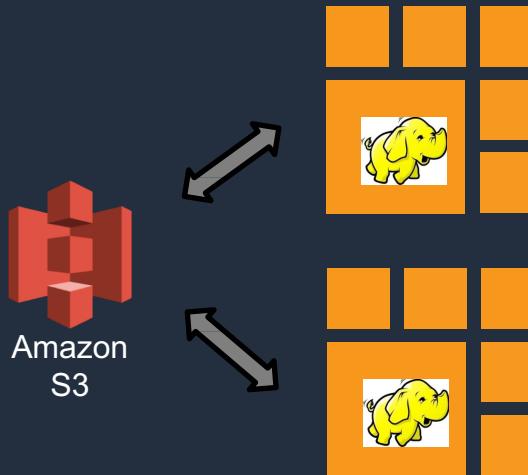


Spot Instances  
for task nodes

Up to 90%  
off Amazon  
EC2  
on-demand  
pricing

# Amazon S3 as your persistent data store

- Separate compute and storage
- Resize and shut down Amazon EMR clusters with no data loss
- Point multiple Amazon EMR clusters at same data in Amazon S3

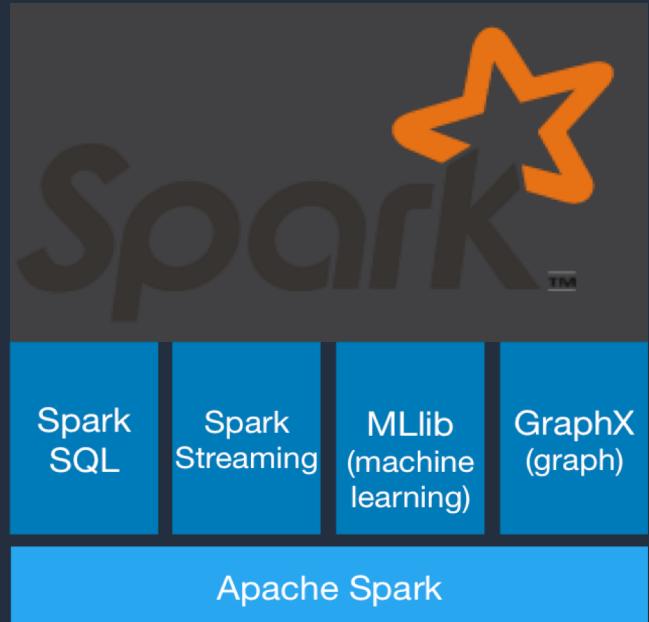


# EMRFS makes it easier to leverage Amazon S3

- Transparent to applications—Use “s3://”
- Support for Amazon S3 server-side and client-side encryption
- Faster listing using EMRFS metadata
- Makes it easier to secure your clusters (fine-grained access control, Kerberos, security configurations)
  - New feature! EMR 5.14.0+ supports the ability to audit users who ran queries that accessed data in Amazon S3 through EMRFS and pushes user/group information to audit logs like AWS CloudTrail.

# Apache Spark

- Fast, general-purpose engine for large-scale data processing
- Write applications quickly in Java, Scala, or Python
- Combine SQL, streaming, and complex analytics



# Apache Zeppelin

- Web-based notebook for interactive analytics
  - Multiple language back end
  - Apache Spark integration
  - Data visualization
  - Collaboration
- 
- <https://zeppelin.apache.org/>

```
val s = "Scala with built-in Apache Spark Integration"  
s: String = Scala with built-in Apache Spark Integration  
Took 0 seconds
```

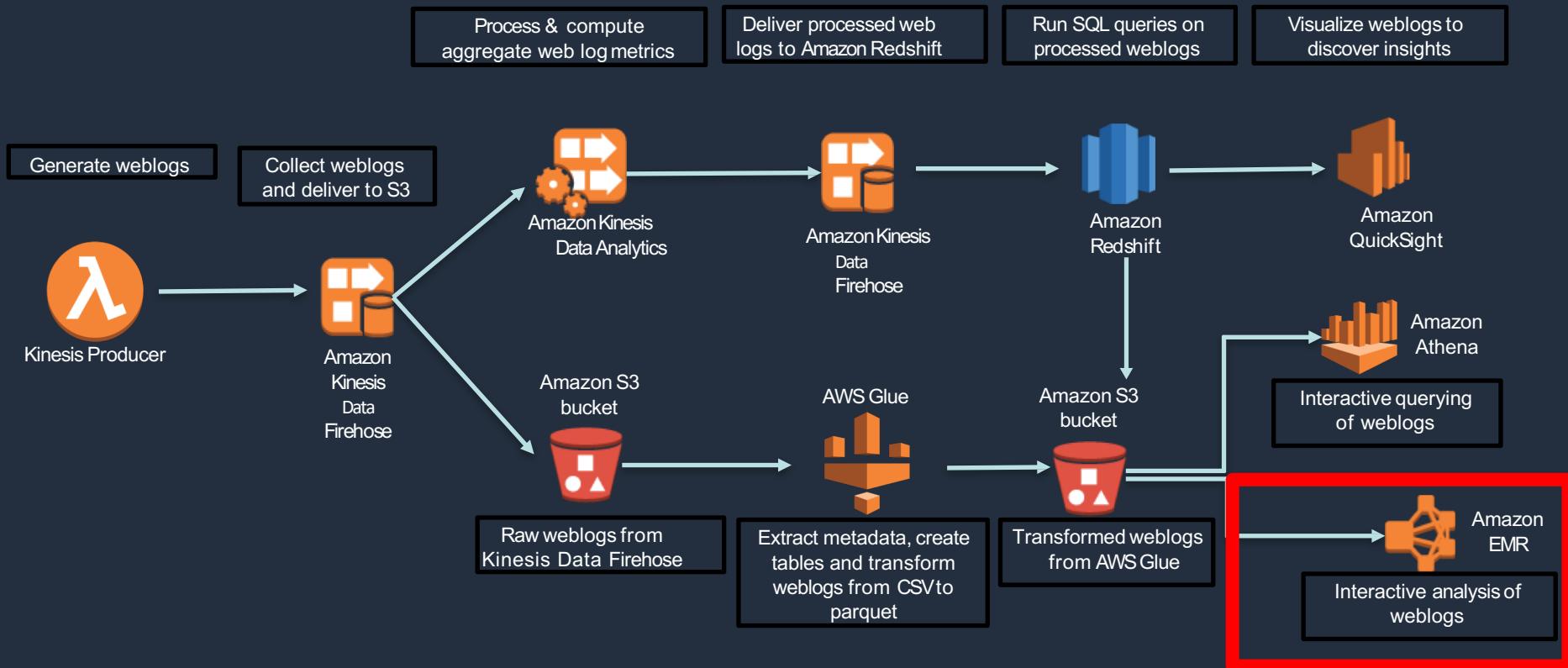
```
%pyspark  
print "Python with built-in Apache Spark Integration"  
Python with built-in Apache Spark Integration  
Took 0 seconds
```

```
%sql -- built-in SparkSQL Support  
select * from RDD
```

# Activity 6

## Interactive analysis using Amazon EMR

# Your application architecture



# Activity 6: Process and query data with Amazon EMR

Time: 20 minutes

We are going to:

- Use a Zeppelin notebook to interact with Amazon EMR cluster
- Process the data in Amazon S3 using Apache Spark
- Query the data processed in the earlier stage and create simple charts

# Activity 6A: Open the Zeppelin interface

- Copy the Zeppelin end point in the **AWS CloudFormation** output section
- Open the Zeppelin link in a new browser window
- Download the First Big Data Application json file from the **BigDataWorkshop.zip**
- Import the notebook using the "Import Note" link on Zeppelin interface
- **Note: Disconnect from VPN or the page will not load**

The screenshot shows two parts of the AWS CloudFormation interface. The top part is the 'Outputs' tab, which lists a single output named 'Zeppelin' with the value 'http://ec2-34-221-80-155.us-west-2.compute.amazonaws.com:8890'. The bottom part is the Zeppelin web interface, featuring a blue header with the Zeppelin logo and 'Notebook'. The main content area says 'Welcome to Zeppelin!' and describes it as a web-based notebook for interactive data analytics. It includes links for 'Import note' (which is highlighted with a red box), 'Create new note', 'Filter', and 'Zeppelin Tutorial'. To the right, there are sections for 'Help' (link to documentation), 'Community' (contribution welcome), and 'Mailing list', 'Issues tracking', 'Github' links.

# Activity 6A: Open the Zeppelin interface

- Step 1: Run the first paragraph
- Enter <stackname>-logs-<account>-us-west-2/processed/parquet in the Bucket field as input
- This is where the processed parquet files were stored earlier in your S3 bucket

The screenshot shows the Zeppelin Notebook interface. The top navigation bar includes the Zeppelin logo, a 'Notebook' dropdown, a search bar, and a user status. The main workspace is titled 'First Big Data Application'. It contains a code cell with the following content:

```
Step 1 : Get the Bucket name
val bucketName = z.input("Bucket")
```

Below the code cell, the output section is labeled 'Bucket' and displays the value: `qls-108881-f75177905b7b5b0e-logs-[REDACTED]-us-west-2/weblogs/processed/parquet`. To the right of the output, there is a status indicator 'FINISHED' with a green checkmark, followed by a red arrow pointing to the 'Run Paragraph' button.

At the bottom of the workspace, the message 'Took 1 sec. Last updated by anonymous at November 23 2017, 9:51:06 PM.' is visible.

# Activity 6B: Run the notebook

- Execute Step 2
  - Create a data frame with the parquet files from the AWS Glue ETL job
- Execute Step 3
  - Sample a few rows

The screenshot shows a Zeppelin Notebook interface with two steps:

**Step 2 : Read data from S3 and create an in-memory Dataframe**

```
import org.apache.spark.sql.SQLContext
val sqlContext = new SQLContext(sc)
val weblogsPath = "s3://<bucketName>"

val webLogDF = sqlContext.read.parquet(weblogsPath)

webLogDF.show()
```

**Step 3 : Sample a few rows**

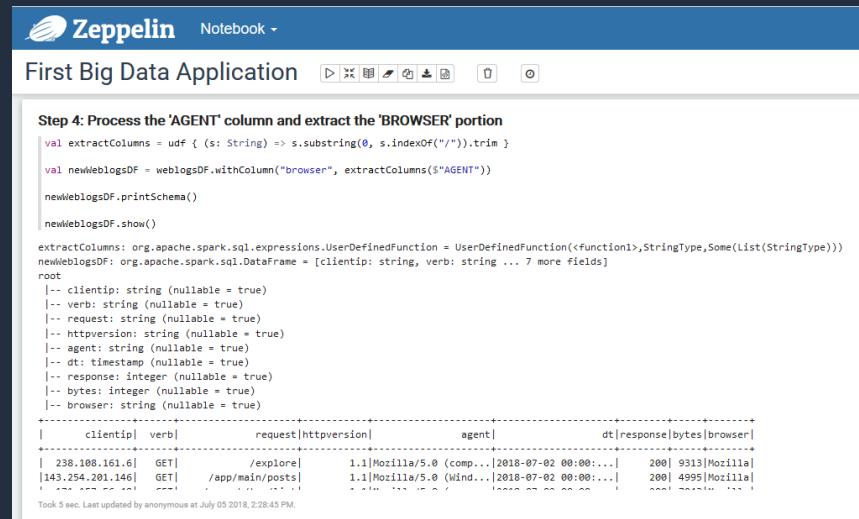
```
webLogDF.show()
webLogDF.printSchema()
```

Both steps show the same data frame output:

clientip	verb	request[httpversion]	agent	dt [responsebytes]
238.108.161.6	GET	/explore	1.1 Mozilla/5.0 (comp... 2018-07-02 00:00:...  200  9313	
[42.224.43.56]	GET	/app/main/welcome	1.1 Mozilla/5.0 (Wind... 2018-07-02 00:00:...  200  4995	
[213.199.42.142]	PUT	/search/tag/list	1.1 Mozilla/5.0 (comp... 2018-07-02 00:00:...  200  4995	
[142.224.43.56]	GET	/wp-admin	1.1 Mozilla/5.0 (comp... 2018-07-02 00:00:...  200  9225	
2,148,133,123	PUT	/search/tag/list	1.1 Mozilla/5.0 (Mac... 2018-07-02 00:00:...  200  287	
126,32,233,51	GET	/explore	1.1 Mozilla/5.0 (comp... 2018-07-02 00:00:...  200  3211	
[38.0,72,36]	GET	/wp-admin	1.1 Mozilla/5.0 (comp... 2018-07-02 00:00:...  200  3535	
229.94.119.172	GET /posts/posts/explore		1.1 Mozilla/5.0 (comp... 2018-07-02 00:00:...  200  8629	
112,193,177,232	GET	/explore	1.1 Mozilla/5.0 (Wind... 2018-07-02 00:00:...  200  8557	

# Activity 6B: Run the notebook

- Execute Step 4 to process the data
  - Notice how the “AGENT” field consists of the “BROWSER” at the beginning of the column value. Let’s extract the browser from the agent field.
  - Create a UDF to extract the browser and add to the data frame
  - Print the new data frame



The screenshot shows a Zeppelin Notebook interface titled "First Big Data Application". The code cell contains Scala code for processing a DataFrame:

```
Step 4: Process the 'AGENT' column and extract the 'BROWSER' portion
val extractColumns = udf { (s: String) => s.substring(0, s.indexOf("/")).trim }

val newWeblogsDF = weblogsDF.withColumn("browser", extractColumns($"AGENT"))

newWeblogsDF.printSchema()
newWeblogsDF.show()
```

The output section shows the schema and the first few rows of the modified DataFrame:

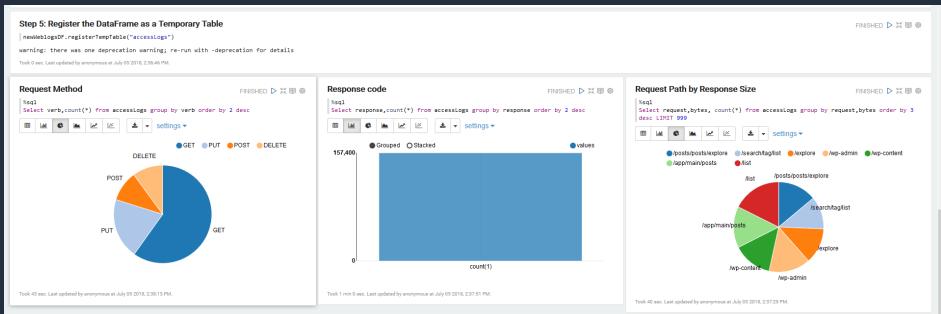
```
root
|-- clientip: string (nullable = true)
|-- verb: string (nullable = true)
|-- request: string (nullable = true)
|-- httpversion: string (nullable = true)
|-- agent: string (nullable = true)
|-- dt: timestamp (nullable = true)
|-- response: integer (nullable = true)
|-- bytes: integer (nullable = true)
|-- browser: string (nullable = true)

+-----+-----+-----+-----+-----+-----+
| clientip| verb| request|httpversion| agent| dt|response|bytes|browser|
+-----+-----+-----+-----+-----+-----+
| 238.108.161.6| GET| /explore| 1.1|Mozilla/5.0 (comp...|2018-07-02 00:00:...| 200| 9313|Mozilla|
|143.254.201.146| GET| /app/main/posts| 1.1|Mozilla/5.0 (Wind...|2018-07-02 00:00:...| 200| 4995|Mozilla|
+-----+-----+-----+-----+-----+-----+
```

At the bottom, it says "Took 5 sec. Last updated by anonymous at July 05 2018, 2:28:45 PM."

# Activity 6B: Run the notebook

- Execute Step 6
  - Register the data frame as a temporary table
  - Now you can run SQL queries on the temporary tables
- Execute the next 3 steps and observe the charts created
  - What did you learn about the data set?



# Review: Interactive analysis using Amazon EMR

- You just learned on how to process and query data using Amazon EMR with Apache Spark
- Amazon EMR has many other frameworks available for you to use
  - Hive, Presto, Flink, Pig, MapReduce
  - Hue, Oozie, HBase

# Optional exercise: Data visualization with Amazon QuickSight

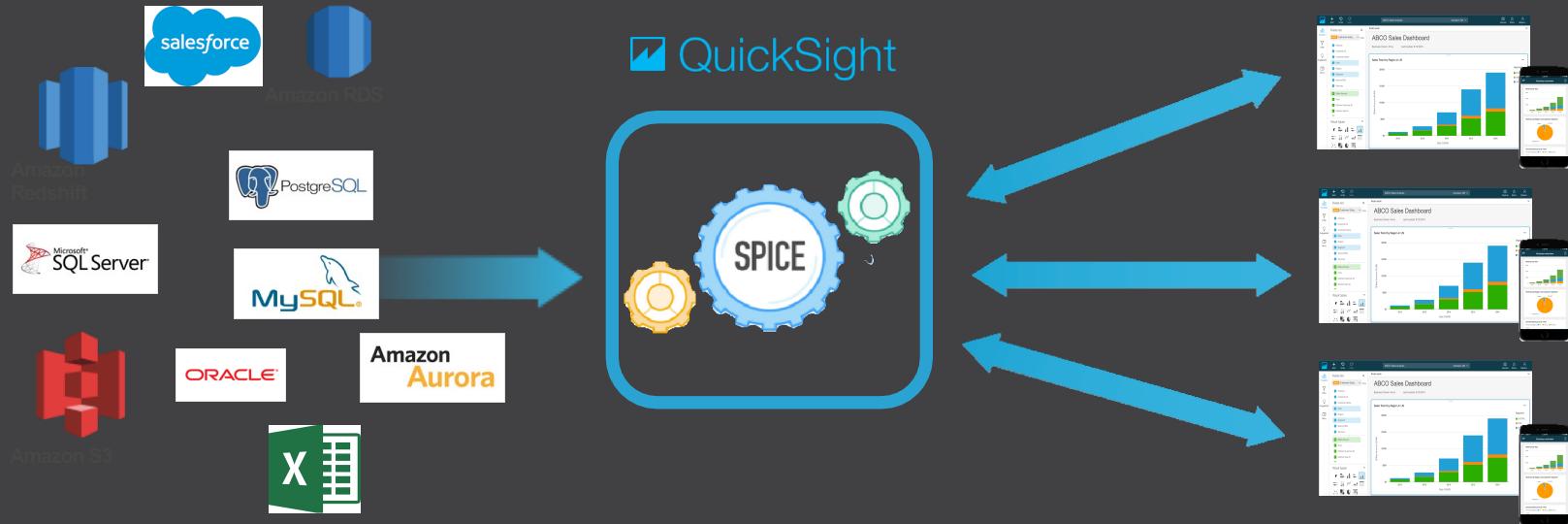
# Amazon QuickSight

- **Fast, easy interactive analytics for anyone, everywhere**
- Ease of use targeted at business users
- Blazing-fast performance powered by SPICE
- Broad connectivity with AWS data services, on-premises data, files, and business applications
- Cloud-native solution that scales automatically
- 1/10th the cost of traditional BI solutions
- Create, share, and collaborate with anyone in your organization, on the web, or on mobile

The screenshot displays the Amazon QuickSight interface. On the left, a smartphone shows a mobile version of the 'Business overview' dashboard. It features three charts: 'Revenue by Year' (stacked bar chart from 2012 to 2016), 'Revenue by Region and Customer Segment' (donut chart with segments for SMB, Enterprise, Unicorn, and others), and 'Service Revenues over Time' (line chart for HR, Billing, and Marketing categories). On the right, a desktop browser window shows the full dashboard for 'ABCO S'. The top navigation bar includes 'Add', 'Undo', 'Redo', and a search icon. The main area is titled 'Field wells' and lists various fields: Channel, Customer ID, Customer Name, Date (highlighted in blue), Region, Segment, Service SKU, Services, Billed Amount, Cost, and Distinct Customer ID. A large vertical bar on the right contains a chart titled 'Sales Trend by Re' with Y-axis labels '\$200K', '\$150K', and '\$100K', and an X-axis labeled 'Amount (SUM)'.

# Connect, SPICE, analyze

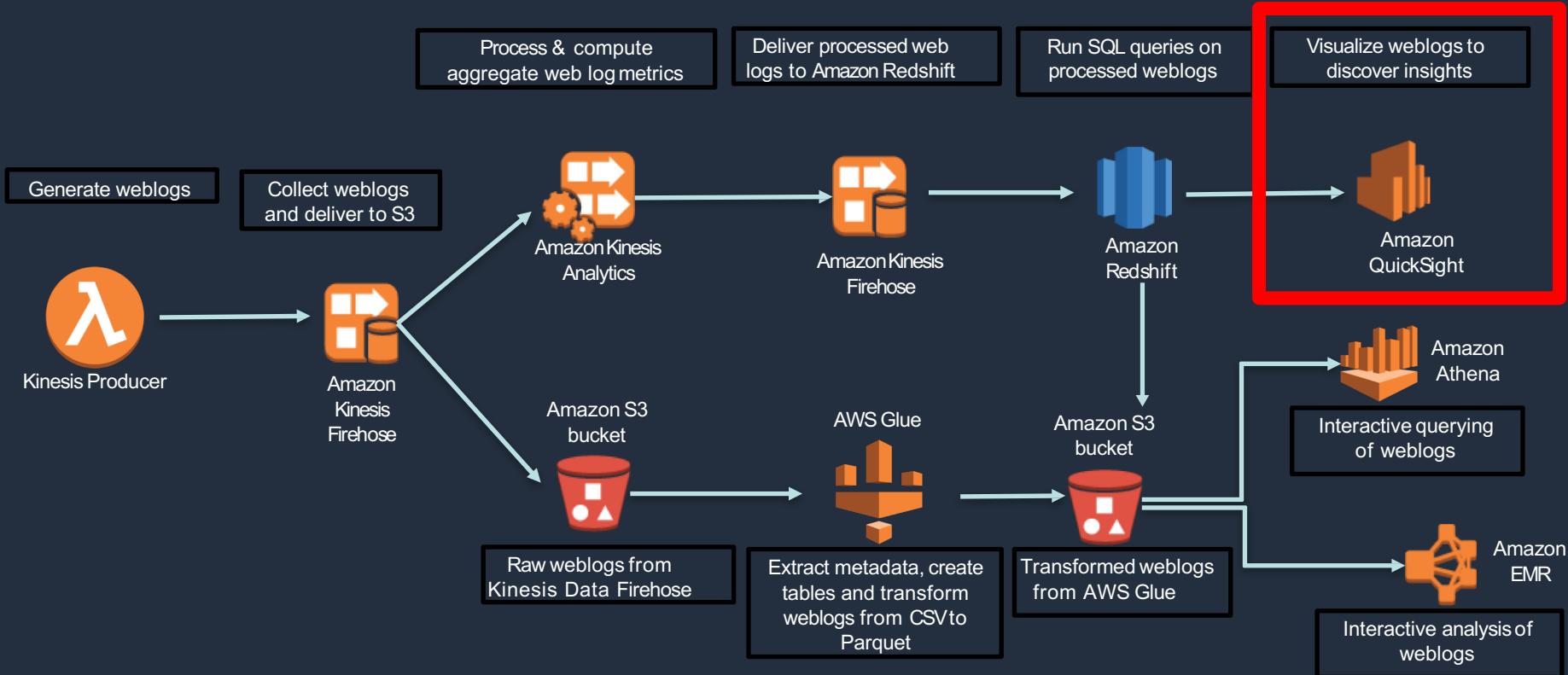
- Amazon QuickSight allows you to connect to data from a wide variety of AWS, third-party, and on-premises sources and import it to SPICE or query directly
- Users can then easily explore, analyze, and share their insights with anyone



# Activity 7

## Visualize results in Amazon QuickSight

# Your application architecture



# Activity 7: Visualization with Amazon QuickSight

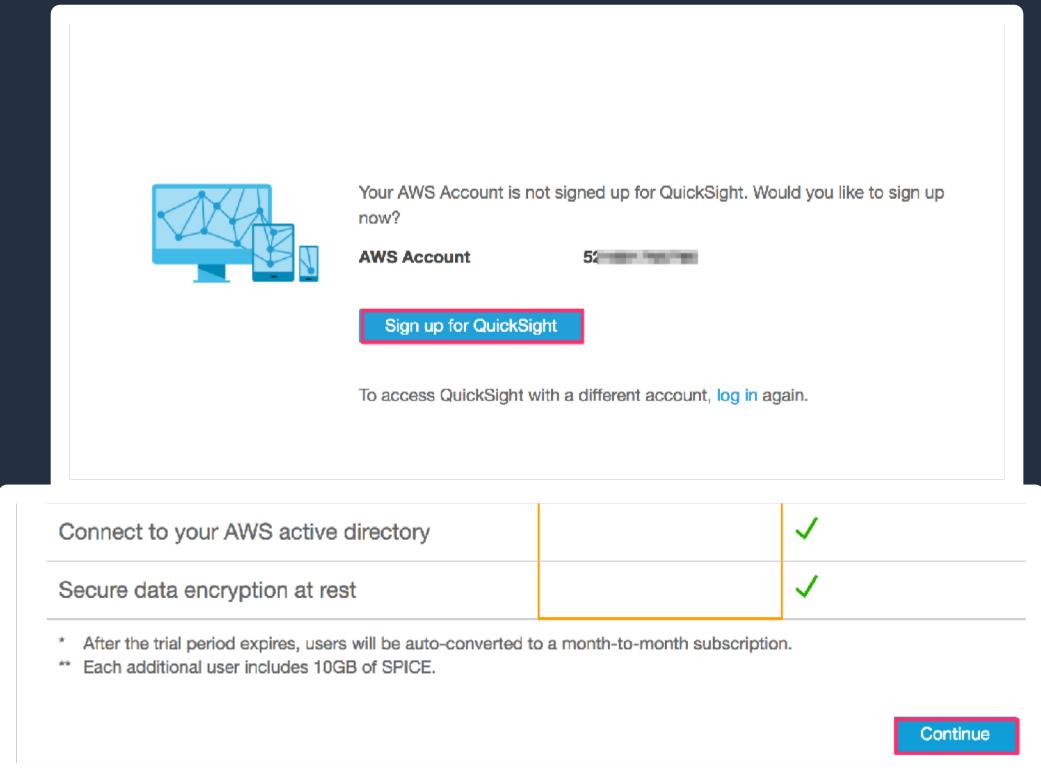
Time: 10 mins

We are going to:

- A. Register for an Amazon QuickSight account
- B. Connect to the Amazon Redshift cluster
- C. Create visualizations for analysis to answer questions like:
  - A. What are the most common http requests and how successful (response code of 200) are they?
  - B. Which are the most requested URIs?

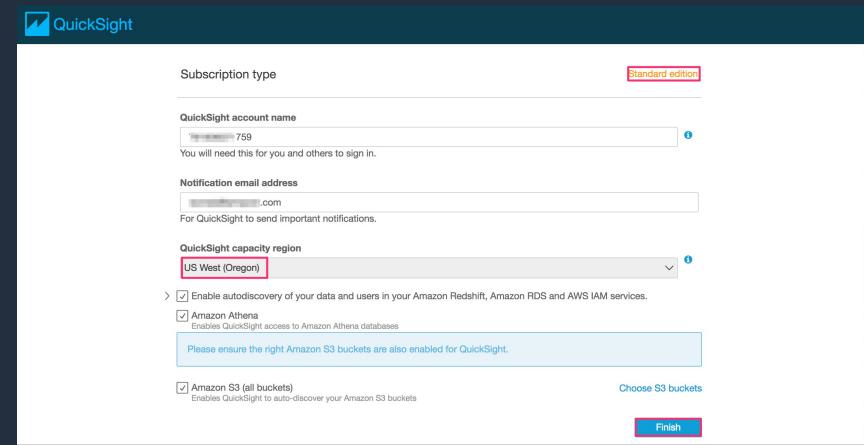
# Activity 7A: Amazon QuickSight registration

- Go to AWS console, click on Amazon QuickSight from the analytics section
- Click on "Sign up" in the next window
- Make sure the subscription type is standard and click "Continue" on the next screen



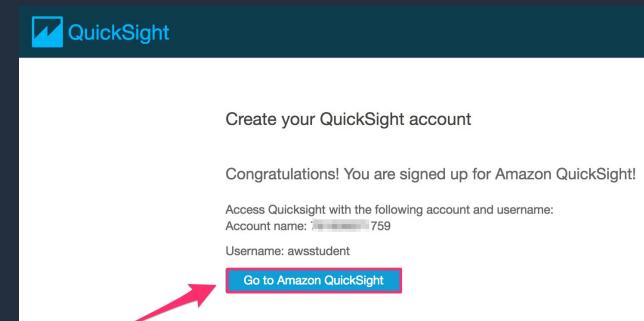
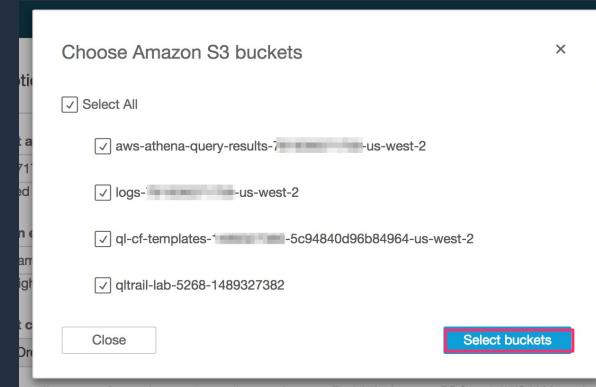
# Activity 7A: Amazon QuickSight registration

- On the "Subscription type" page, enter the account name (see **note** below)
- Enter your **email address**
- Select **US West region**
- Check the "**Amazon S3 (all buckets)**" box
- **Note: Amazon QuickSight account name is your AWS account number**



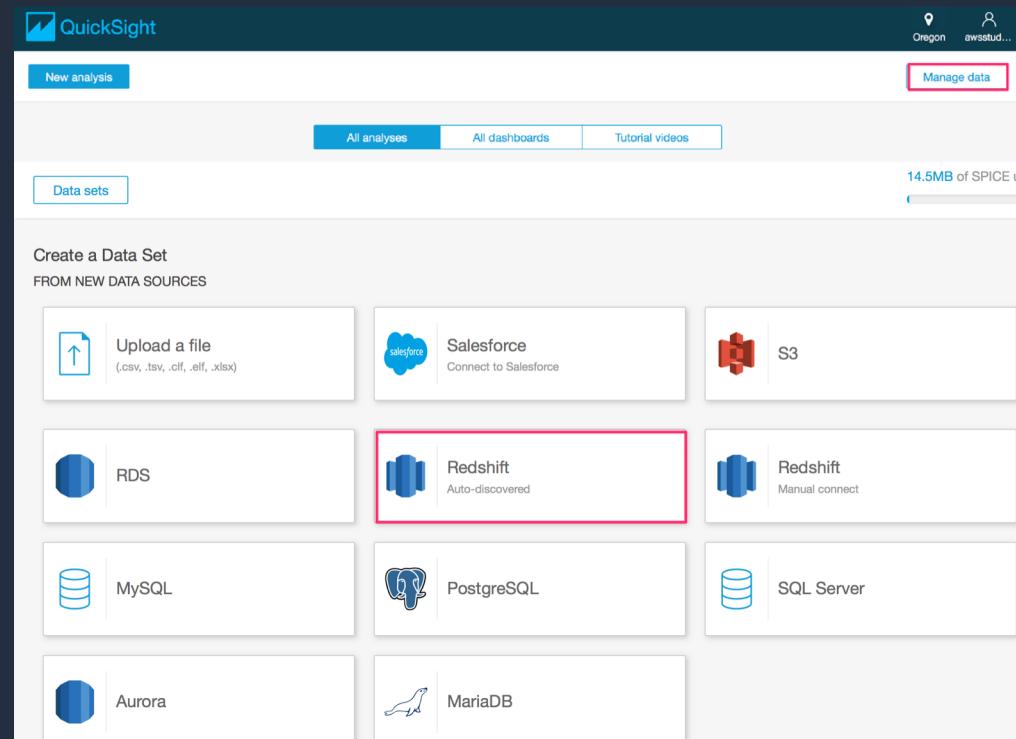
# Activity 7A: Amazon QuickSight registration

- If a popup box to choose Amazon S3 buckets appears, click "**Select buckets**"
- Click on "**Go to Amazon QuickSight**"
- Dismiss welcome screen
- Make sure you are in us-west-2 (Oregon)

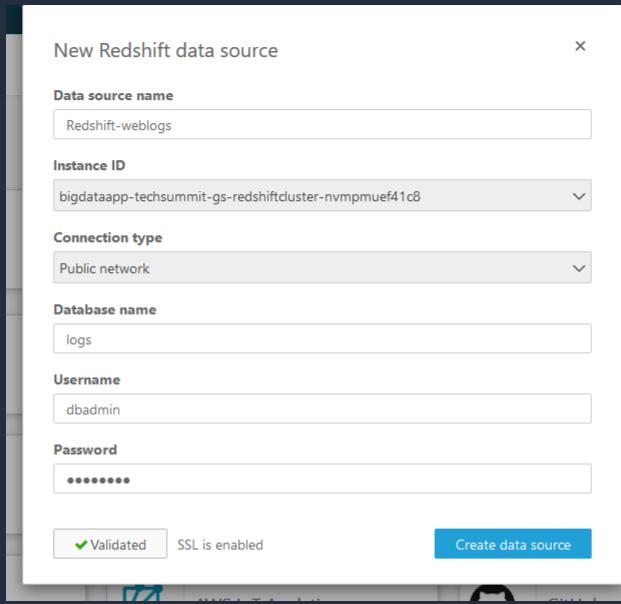


# Activity 7B: Connect to data source

- Click on "Manage Data" and then select "New Data set" to create a new data set in Amazon QuickSight.
- Choose "Redshift (Auto-discovered)" as the data source. Amazon QuickSight auto-discovers databases associated with your AWS account (Amazon Redshift database in this case).



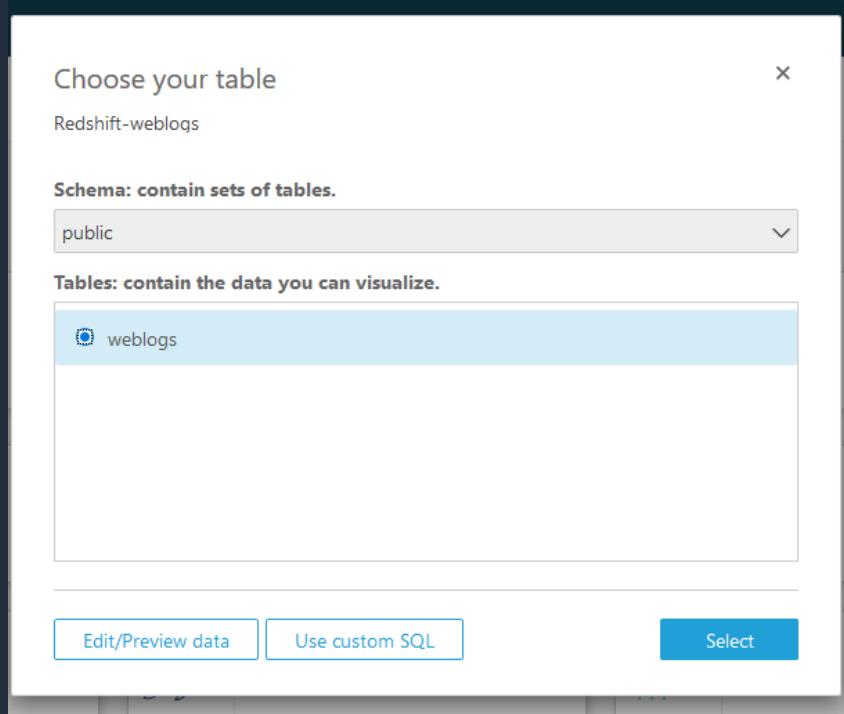
# Activity 7B: Connect to Amazon Redshift



**Note: Use "dbadmin" as the username. You can get the Amazon Redshift database password by navigating to the AWS CloudFormation outputs section.**

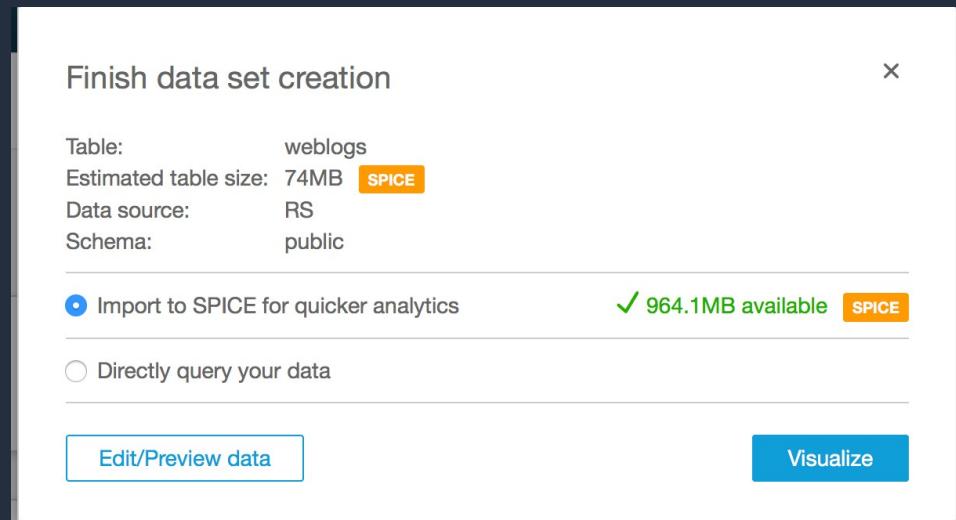
# Activity 7C: Choose your weblogs Amazon Redshift table

- Select “public” schema
- Select “weblogs” in tables



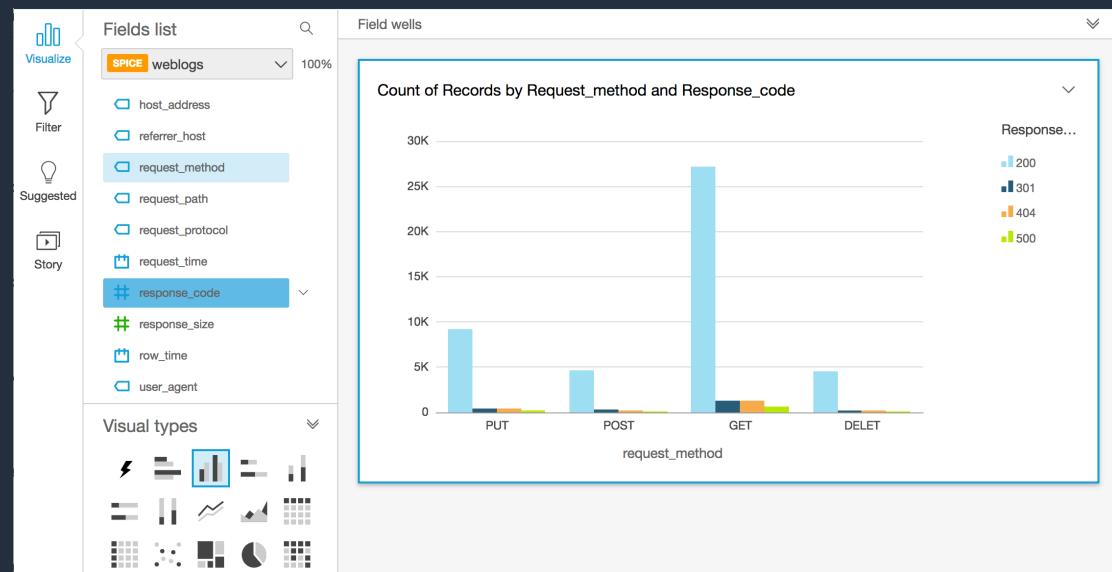
# Activity 7D: Ingest data into SPICE

- SPICE is Amazon QuickSight's in-memory optimized calculation engine, designed specifically for fast, interactive data visualization
- You can improve the performance of database data sets by importing the data into SPICE instead of using a direct query to the database



# Activity 7E: Creating your first analysis

- What are the most requested http request types and their corresponding response codes for this site?
- Simply select request, response, and let AUTOGRAPH create the optimal visualization



# Review - Creating your analysis

- Exercise: Add a visual to demonstrate which URI are the most requested

