Your first Big Data Analytics on AWS

Channy Yun

Tech Evangelist, Amazon Web Service Korea

Pre-requirements

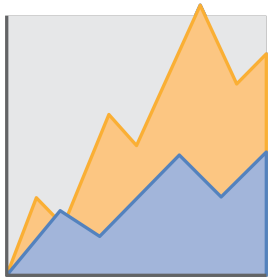
Free Hands-on Labs on Qwiklabs

- AWS CLI: https://aws.amazon.com/ko/cli/
- Amazon S3: https://qwiklab.com/focuses/preview/1170
- Amazon EC2: https://qwiklab.com/focuses/preview/1167
- Amazon EMR: https://qwiklab.com/focuses/preview/1168
- Amazon Redshift: https://qwiklab.com/focuses/preview/1583

Joining AWS and AWS Educate Program

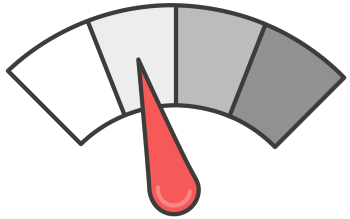
- https://www.awseducate.com/Application

Data-driven development



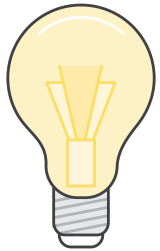
Retrospective
analysis and
reporting

Amazon Redshift,
Amazon RDS
Amazon S3
Amazon EMR



Here-and-now
real-time processing
and dashboards

Amazon Kinesis
Amazon EC2
AWS Lambda



Predictions
to enable smart
applications

Amazon Machine
Learning

AWS Services for Big Data Analytics

DynamoDB

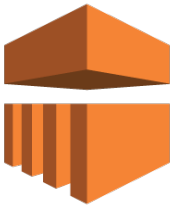
Zero admin **NoSQL DB** with fast, predictable performance

Redshift

Petabyte-scale **data warehouse** service

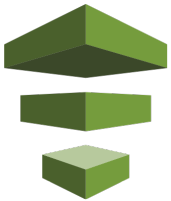
Elastic MapReduce

Hosted **Hadoop framework**



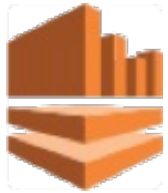
Data Pipeline

Move data among AWS services and on-premises data sources



Kinesis

Real-time processing of streaming data at massive scale



Compute

EC2 WorkSpaces

Storage

S3 EBS Glacier Storage Gateway

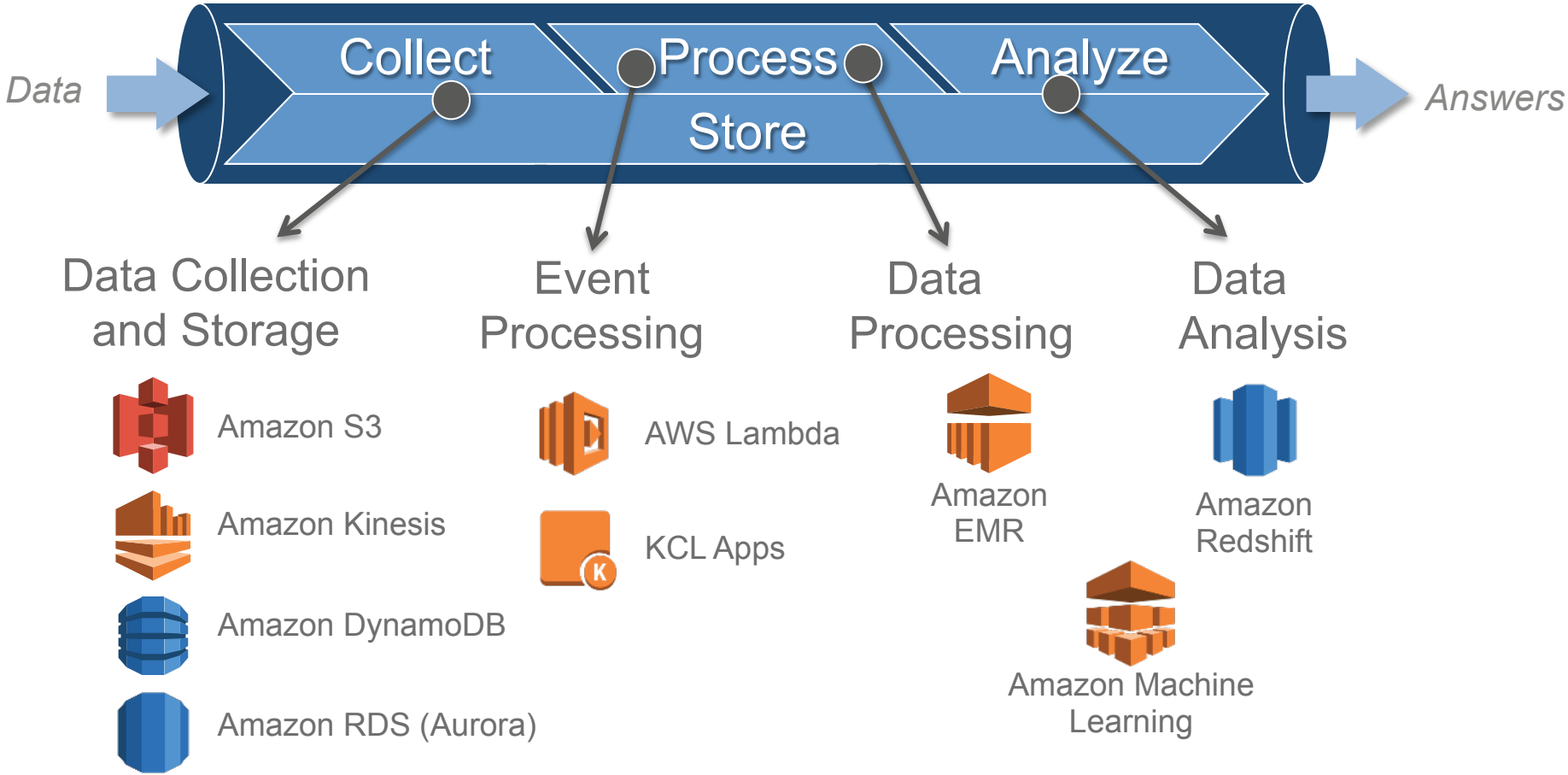
Networking

VPC Direct Connect ELB Route53

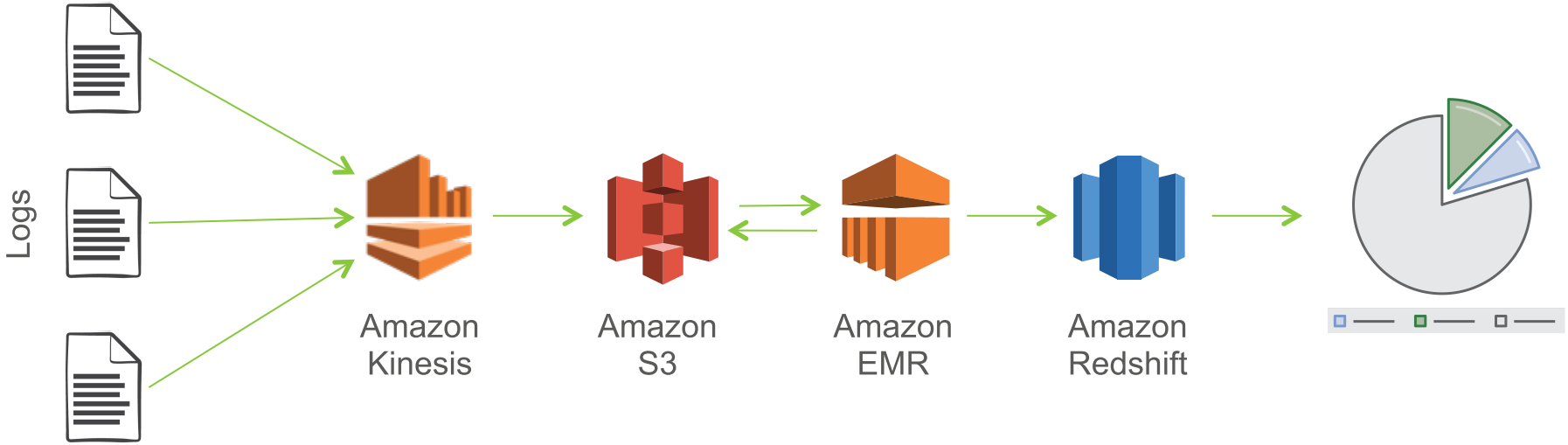
Supercell Use-cases

- **Kinesis:** Real-time data stream of in-game activity
- **Multiple Kinesis applications:** Dashboards, analytics and storage
- **S3 and Glacier:** Data storage and long term archival
- **Data Warehouse:** BI reporting and interactive queries

AWS Building Blocks for Big Data Analytics

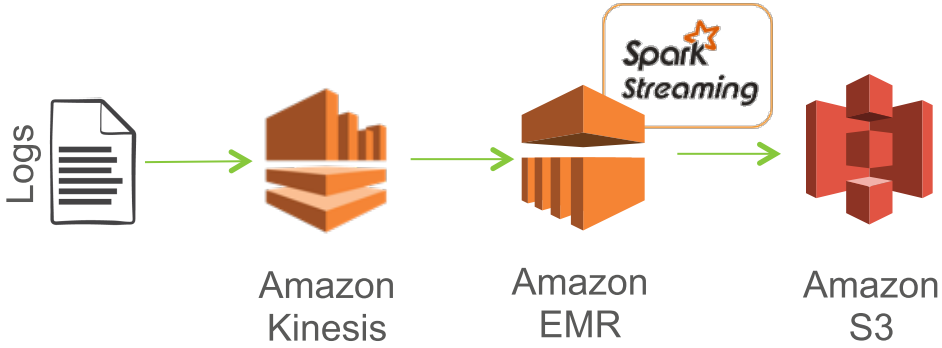


Your first big data application on AWS



1. Collect


Data
Collect
Process
Analyze
Store
Answers


Logs
Spark Streaming
Amazon Kinesis
Amazon EMR
Amazon S3

2. Process


Data
Collect
Process
Analyze
Store
Answers


Spark
Amazon S3
Amazon EMR

3. Analyze


Data
Collect
Process
Analyze
Store
Answers


Amazon S3
Amazon Redshift
SQL

Set up with the AWS CLI

1. Installation

Mac:

```
$ pip install awscli
```

Windows:

32-bit MSI: http://s3.amazonaws.com/aws-cli/AWSCLI32.msi
64-bit MSI: http://s3.amazonaws.com/aws-cli/AWSCLI64.msi

2. Configuration

```
$ aws configure
AWS Access Key ID [None]:
AWS Secret Access Key [None]:
Default region name [None]:
Default output format [json]:
```

3. Usage

```
$ aws ec2 describe-instances
```

service (command) operation (subcommand)

Amazon Kinesis

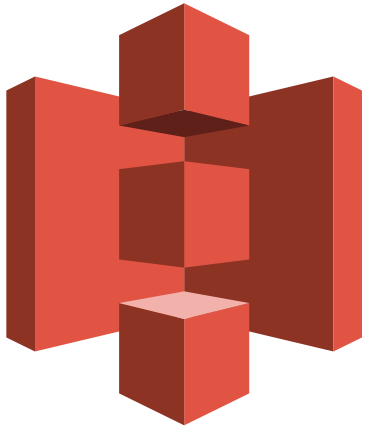
Create a single-shard Amazon Kinesis stream for incoming log data:

```
aws kinesis create-stream \
  --stream-name AccessLogStream \
  --shard-count 1
```

Amazon S3

Create an Amazon S3 bucket to hold the files for Amazon EMR processing, plus input files for Amazon Redshift:

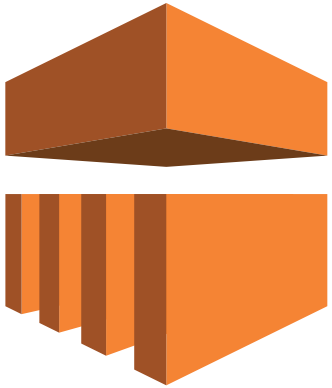
```
aws s3 mb s3://YOUR-BUCKET-NAME
```



Amazon EMR

Launch a 3-node Amazon EMR cluster with Spark and Hive:

```
aws emr create-cluster \
  --name "demo" \
  --instance-type m3.xlarge \
  --instance-count 2 \
  --release-label emr-4.0.0 \
  --ec2-attributes KeyName=YOUR-AWS-SSH-KEY \
  --use-default-roles \
  --applications Name=Hive Name=Spark
```



Amazon Redshift

Create a single-node Amazon Redshift data warehouse:

```
aws redshift create-cluster \
  --cluster-identifier demo \
  --db-name demo \
  --node-type dc1.large \
  --cluster-type single-node \
  --master-username master \
  --master-user-password CHOOSE-A-REDSHIFT-PASSWORD \
  --publicly-accessible \
  --port 8192
```

Your first big data application on AWS

1. Collect

Amazon Kinesis Log4J Appender

In a separate terminal window on your local machine, download Log4J Appender:

```
wget http://emr-kinesis.s3.amazonaws.com/publisher/\
kinesis-log4j-appender-1.0.0.jar
```

Then download and save the sample Apache log file:

```
wget http://elasticmapreduce.s3.amazonaws.com/samples/\
pig-apache/input/access_log_1
```

Amazon Kinesis Log4J Appender

Create a file called `AwsCredentials.properties` with credentials for an IAM user with permission to write to Amazon Kinesis:

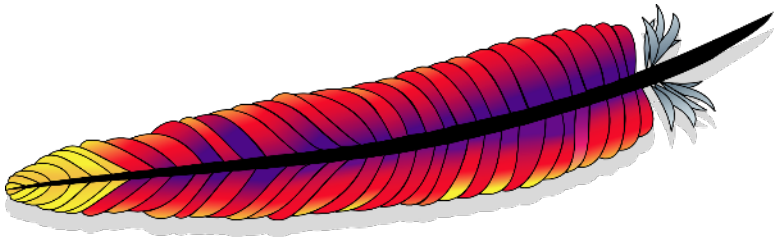
```
accessKey=YOUR-IAM-ACCESS-KEY
secretKey=YOUR-SECRET-KEY
```

Then start the Amazon Kinesis Log4J Appender:

```
java -cp .:kinesis-log4j-appender-1.0.0.jar \
com.amazonaws.services.kinesis.log4j.FilePublisher \
access_log_1 &
```

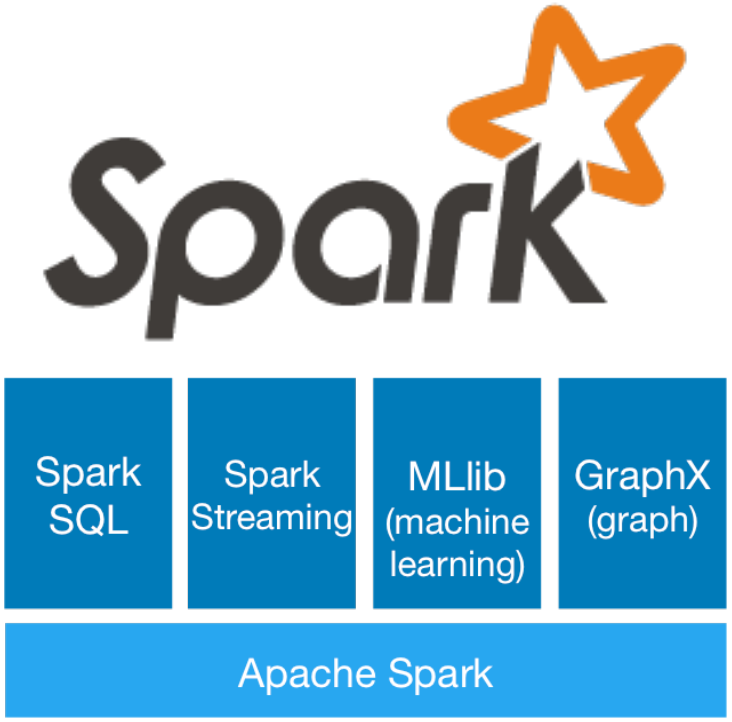
Log file format

```
75.35.230.210 - - [20/Jul/2009:22:22:42 -0700]
"GET /images/pigtrihawk.jpg HTTP/1.1" 200 29236
"http://www.swivel.com/graphs/show/1163466"
"Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:
1.9.0.11) Gecko/2009060215 Firefox/3.0.11 (.NET CLR
3.5.30729)"
```

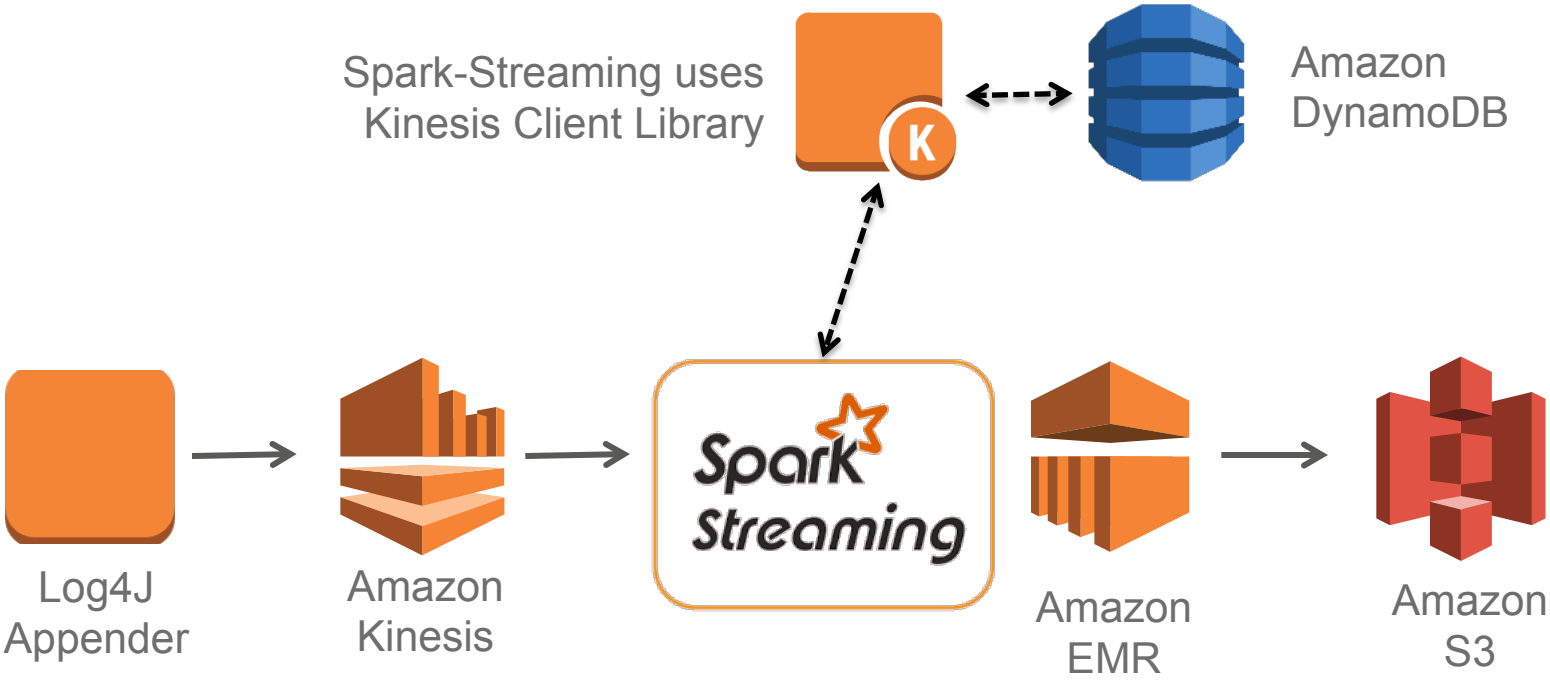


Spark

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



Amazon Kinesis and Spark Streaming



Using Spark Streaming on Amazon EMR

Use SSH to log in to your cluster:

```
ssh -o TCPKeepAlive=yes -o ServerAliveInterval=30 \
-i YOUR-AWS-SSH-KEY hadoop@YOUR-EMR-HOSTNAME
```

On your cluster, download the Amazon Kinesis client for Spark:

```
wget http://repo1.maven.org/maven2/com/amazonaws/amazon-
kinesis-client/1.6.0/amazon-kinesis-client-1.6.0.jar
```

Using Spark Streaming on Amazon EMR

Cut down on console noise:

```
sudo sed -i 's/INFO/ERROR/g' /usr/lib/spark/conf/spark-
defaults.conf
sudo sed -i 's/INFO/ERROR/g' /usr/lib/spark/conf/log4j.properties
```

Start the Spark shell:

```
spark-shell --jars /usr/lib/spark/extras/lib/spark-streaming-
kinesis-asl.jar,amazon-kinesis-client-1.6.0.jar --driver-java-
options "-Dlog4j.configuration=file:///etc/spark/conf/
log4j.properties"
```

Using Spark Streaming on Amazon EMR

```
/* import required libraries */
import org.apache.spark.SparkContext
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming.Seconds
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming.StreamingContext.toPairDStreamFunctions
import org.apache.spark.streaming.kinesis.KinesisUtils
import com.amazonaws.auth.DefaultAWSCredentialsProviderChain
import com.amazonaws.services.kinesis.AmazonKinesisClient
import com.amazonaws.services.kinesis.clientlibrary.lib.worker._
import java.util.Date
import org.apache.hadoop.io.compress._
```

Using Spark Streaming on Amazon EMR

```
/* Set up the variables as needed */
val streamName = "AccessLogStream"
val endpointUrl = "https://kinesis.YOUR-REGION.amazonaws.com"
val outputDir = "s3://YOUR-S3-BUCKET/access-log-raw"
val outputInterval = Seconds(60)

/* Reconfigure the spark-shell */
val sparkConf = sc.getConf
sparkConf.setAppName("S3Writer")
sparkConf.remove("spark.driver.extraClassPath")
sparkConf.remove("spark.executor.extraClassPath")
sc.stop
val sc = new SparkContext(sparkConf)
```

Reading Amazon Kinesis with Spark Streaming

```
/* Setup the KinesisClient */
val kinesisClient = new AmazonKinesisClient(new
DefaultAWSCredentialsProviderChain())
kinesisClient.setEndpoint(endpointUrl)

/* Determine the number of shards from the stream */
val numShards =
kinesisClient.describeStream(streamName).getStreamDescription().getShard
s().size()

/* Create one worker per Kinesis shard */
val ssc = new StreamingContext(sc, outputInterval)
val kinesisStreams = (0 until numShards).map { i =>
  KinesisUtils.createStream(ssc, streamName,
endpointUrl,outputInterval,InitialPositionInStream.TRIM_HORIZON,
StorageLevel.MEMORY_ONLY)
}
```

Writing to Amazon S3 with Spark Streaming

```
/* Merge the worker Dstreams and translate the byteArray to string */
val unionStreams = ssc.union(kinesisStreams)
val accessLogs = unionStreams.map(byteArray => new String(byteArray))

/* Write each RDD to Amazon S3 */
accessLogs.foreachRDD( (rdd,time) => {
  if (rdd.count > 0) {
    val outPartitionFolder = new
java.text.SimpleDateFormat("'year='yyyy/'month='MM/'day='dd/'hour='hh/'min='m
m").format(new Date(time.milliseconds))
    rdd.saveAsTextFile("%s/
%s".format(outputDir,outPartitionFolder),classOf[GzipCodec])
}})
ssc.start()
ssc.awaitTermination()
```

View the output files in Amazon S3

List all of the partition prefixes:

```
aws s3 ls s3://YOUR-S3-BUCKET/access-log-raw/ --
recursive
```

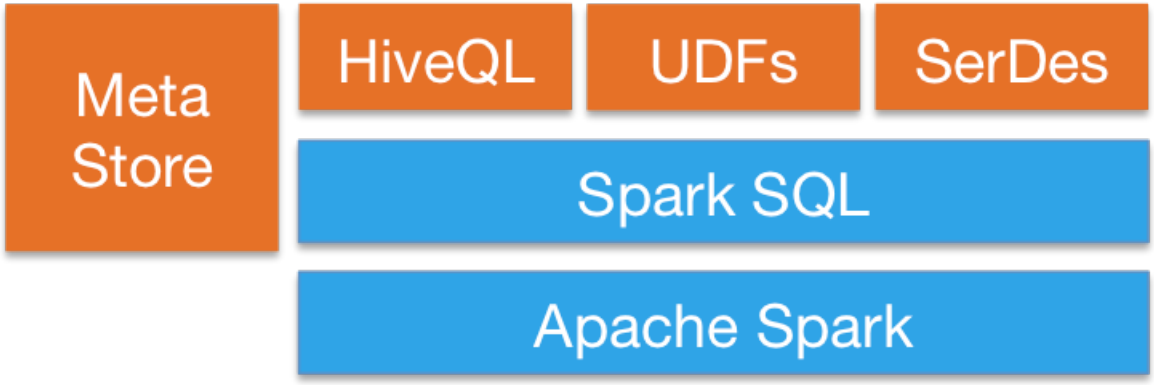
List all the output files:

```
aws s3 ls s3://YOUR-S3-BUCKET/access-log-raw/
year=yyyy/month=mm/day=dd/hour=HH/
```

2. Process

Spark SQL

Spark's module for working with structured data using SQL



Run unmodified Hive queries on existing data

Using Spark SQL on Amazon EMR

Use SSH to log in to your Amazon EMR cluster:

```
ssh -i YOUR-AWS-SSH-KEY YOUR-EMR-HOSTNAME
```

Start the Spark SQL shell:

```
spark-sql --driver-java-options "-
Dlog4j.configuration=file:///etc/spark/conf/
log4j.properties"
```

Create a table that points to your Amazon S3 bucket

```
CREATE EXTERNAL TABLE access_log_raw(
  host STRING, identity STRING,
  user STRING, request_time STRING,
  request STRING, status STRING,
  size STRING, referrer STRING,
  agent STRING
)
PARTITIONED BY (year INT, month INT, day INT, hour INT, min INT)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
WITH SERDEPROPERTIES (
  "input.regex" = "([^ ]*) ([^ ]*) ([^ ]*) (-|\\[[^\\]]*\\]) ([^ \"]*|\"[^
\"]*\") (-|[0-9]*) (-|[0-9]*)(?: ([^ \"]*|\"[^\"]*\") ([^ \"]*|\"[^\"]*
\"))?"
)
LOCATION 's3://YOUR-S3-BUCKET/access-log-raw';

msck repair table access_log_raw;
```

Query the data with Spark SQL

```
-- return the first row in the stream
SELECT * FROM access_log_raw LIMIT 1;

-- return count all items in the Stream
SELECT COUNT(1) FROM access_log_raw;

-- find the top 10 hosts
SELECT host, COUNT(1) FROM access_log_raw GROUP BY
host ORDER BY 2 DESC LIMIT 10;
```

Preparing the data for Amazon Redshift import

We will transform the data that is returned by the query before writing it to our Amazon S3-stored external Hive table

Hive user-defined functions (UDF) in use for the text transformations: from_unixtime, unix_timestamp and hour

The “hour” value is important: this is what’s used to split and organize the output files before writing to Amazon S3. These splits will allow us to more efficiently load the data into Amazon Redshift later in the lab using the parallel “COPY” command.

Create an external table in Amazon S3

```
CREATE EXTERNAL TABLE access_log_processed (
  request_time STRING,
  host STRING,
  request STRING,
  status INT,
  referrer STRING,
  agent STRING
)
PARTITIONED BY (hour STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION 's3://YOUR-S3-BUCKET/access-log-processed';
```

Configure partition and compression

```
-- setup Hive's "dynamic partitioning"
-- this will split output files when writing to Amazon S3
SET hive.exec.dynamic.partition.mode=nonstrict;
SET hive.exec.dynamic.partition=true;


-- compress output files on Amazon S3 using Gzip
SET mapred.output.compress=true;
SET hive.exec.compress.output=true;
SET mapred.output.compression.codec=
org.apache.hadoop.io.compress.GzipCodec;
SET io.compression.codecs=org.apache.hadoop.io.compress.GzipCodec;
```

Write output to Amazon S3

```
-- convert the Apache log timestamp to a UNIX timestamp
-- split files in Amazon S3 by the hour in the log lines
INSERT OVERWRITE TABLE access_log_processed PARTITION (hour)
  SELECT
    from_unixtime(unix_timestamp(request_time,
      '[dd/MMM/yyyy:HH:mm:ss Z]')),
    host,
    request,
    status,
    referrer,
    agent,
    hour(from_unixtime(unix_timestamp(request_time,
      '[dd/MMM/yyyy:HH:mm:ss Z]'))) as hour
  FROM access_log_raw;
```

View the output files in Amazon S3

List all of the partition prefixes:

```
aws s3 ls s3://YOUR-S3-BUCKET/access-log-processed/
```

List one of the split output files:

```
aws s3 ls s3://YOUR-S3-BUCKET/access-log-processed/
hour=22/
```

3. Analyze

Connect to Amazon Redshift

```
# using the PostgreSQL CLI
psql -h YOUR-REDSHIFT-ENDPOINT \
     -p 8192 -U master demo
```

Or use any JDBC or ODBC SQL client with the PostgreSQL 8.x drivers or native Amazon Redshift support

- Aginity Workbench for Amazon Redshift
- SQL Workbench/J

Create an Amazon Redshift table to hold your data

```
CREATE TABLE accesslogs (
  request_time timestamp,
  host varchar(50),
  request varchar(1024),
  status int,
  referrer varchar(1024),
  agent varchar(1024)
)
DISTKEY(host)
SORTKEY(request_time);
```

Loading data into Amazon Redshift

“COPY” command loads files in parallel

```
COPY accesslogs
FROM 's3://YOUR-S3-BUCKET/access-log-processed'
CREDENTIALS
  'aws_access_key_id=YOUR-IAM-
ACCESS_KEY;aws_secret_access_key=YOUR-IAM-SECRET-KEY'
DELIMITER '\t' IGNOREHEADER 0
MAXERROR 0
GZIP;
```

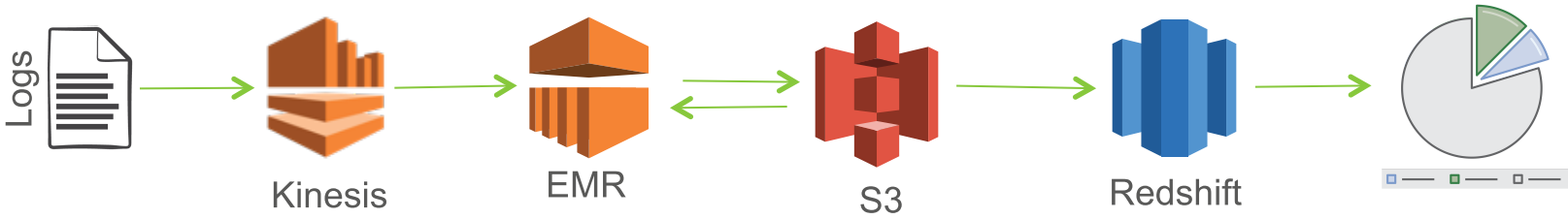
Amazon Redshift test queries

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

-- find the 404 status codes
SELECT COUNT(1) FROM accessLogs WHERE status = 404;

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

Your first big data application on AWS



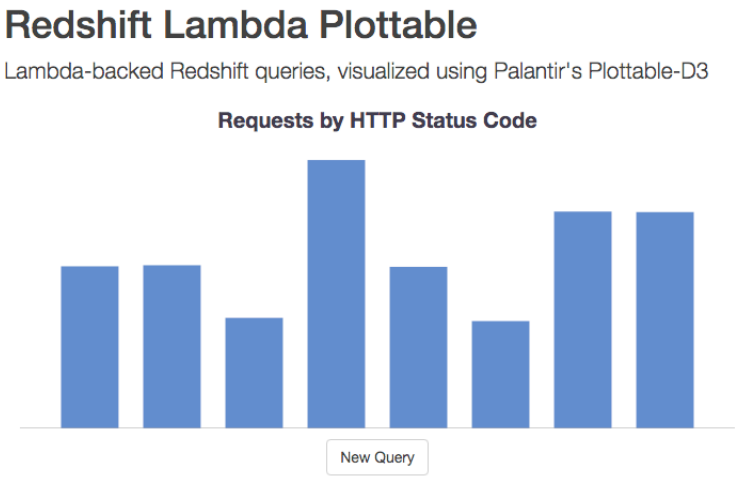

```
logs=# SELECT COUNT(1) FROM accessLogs WHERE status = 404;
count
-------
   977
(1 row)

logs=# SELECT TOP 1 request,COUNT(1) FROM accesslogs WHERE status = 404 GROUP BY
1 ORDER BY 2 DESC;
             request             | count
---------------------------------+-------
 "GET /favicon.ico HTTP/1.1" |   398
(1 row)
```

A favicon would fix 398 of the total 977 PAGE NOT FOUND (404) errors

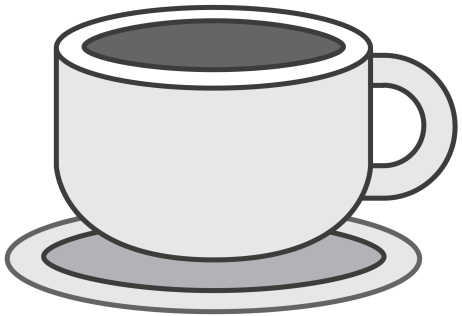
Visualize the results

- Client-side JavaScript example using Plottable, a library built on D3
- Hosted on Amazon S3 for pennies a month
- AWS Lambda function used to query Amazon Redshift



Try it yourself on the AWS cloud…

…around the same cost as a cup of coffee



Service	Est. Cost*
Amazon Kinesis	$1.00
Amazon S3 (free tier)	$0
Amazon EMR	$0.44
Amazon Redshift	$1.00
Est. Total	$2.44

*Estimated costs assumes: use of free tier where available, lower cost instances, dataset no bigger than 10MB and instances running for less than 4 hours. Costs may vary depending on options selected, size of dataset, and usage.