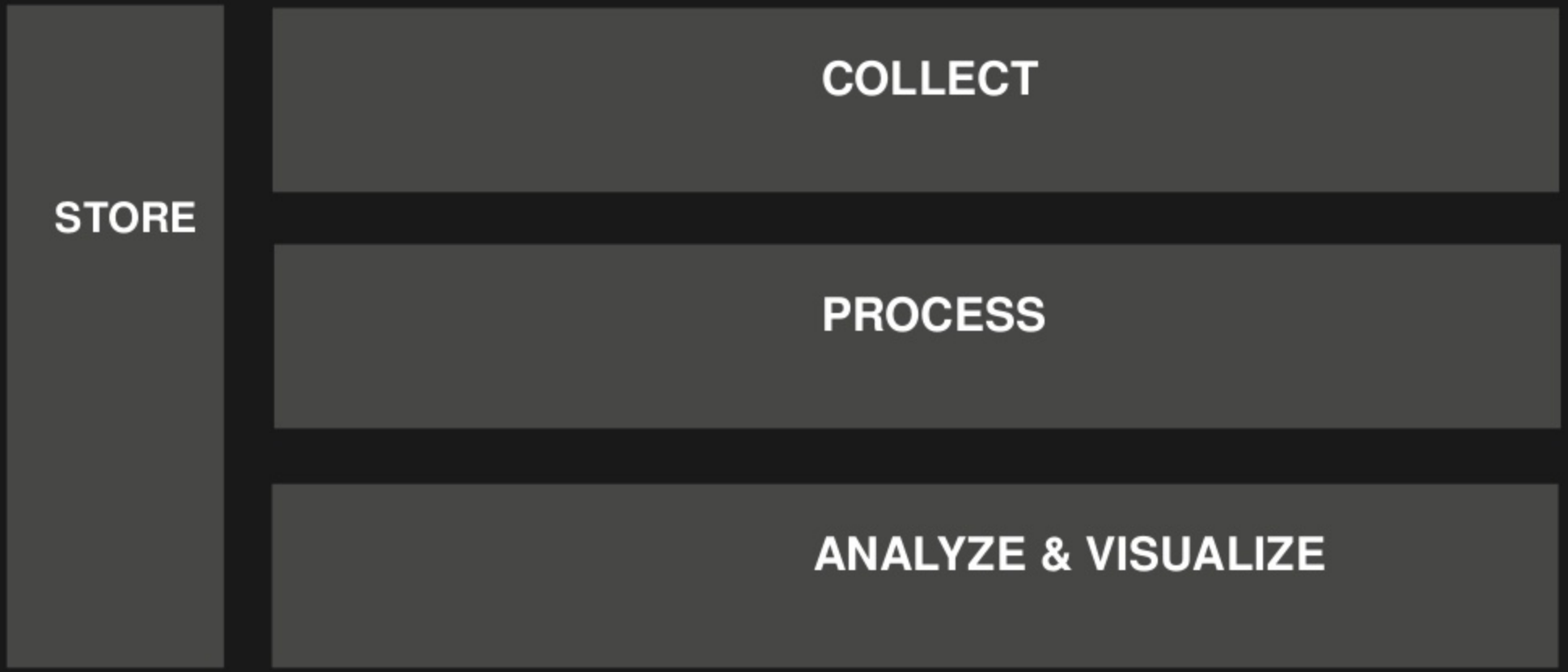
Building Your First Big Data Application on AWS

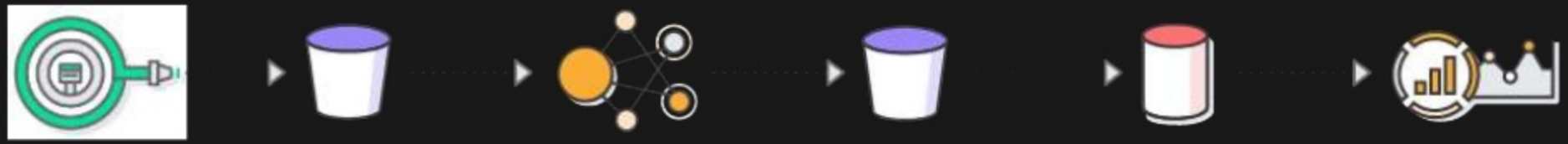
Shree Kenghe and Dario Rivera Solution Architect, AWS

Your First Big Data Application on AWS



Your First Big Data Application on AWS

A Modern Take on the Classic Data Warehouse



Amazon Kinesis Firehose

Amazon S3

Amazon EMR

Amazon S3

Amazon Redshift

Amazon QuickSight

http://aws.amazon.com/big-data/use-cases/

Setting up the environment

Data Storage with Amazon S3

Download all the CLI steps: http://bit.ly/aws-big-data-steps

Create an Amazon S3 bucket to store the data collected with Amazon Kinesis Firehose

```
aws s3 mb s3://YOUR-S3-BUCKET-NAME
```

Access Control with IAM

Create an IAM role to allow Firehose to write to the S3 bucket

firehose-policy.json:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "firehose.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
```

Access Control with IAM

Create an IAM role to allow Firehose to write to the S3 bucket

s3-rw-policy.json:

```
{ "Version": "2012-10-17",
  "Statement": {
      "Effect": "Allow",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::YOUR-S3-BUCKET-NAME",
        "arn:aws:s3:::YOUR-S3-BUCKET-NAME/*"
      ]
} }
```

Access Control with IAM

Create an IAM role to allow Firehose to write to the S3 bucket

```
aws iam create-role --role-name firehose-demo \
--assume-role-policy-document file://firehose-policy.json
```

Copy the value in “Arn” in the output,
e.g., *arn:aws:iam::123456789:role/firehose-demo*

```
aws iam put-role-policy --role-name firehose-demo \
--policy-name firehose-s3-rw \
--policy-document file://s3-rw-policy.json
```

Data Collection with Amazon Kinesis Firehose

Create a Firehose stream for incoming log data

```
aws firehose create-delivery-stream \
  --delivery-stream-name demo-firehose-stream \
  --s3-destination-configuration \
RoleARN=YOUR-FIREHOSE-ARN,\
BucketARN="arn:aws:s3:::YOUR-S3-BUCKET-NAME",\
Prefix=firehose\/,\
BufferingHints={IntervalInSeconds=60},\
CompressionFormat=GZIP
```

Data Processing with Amazon EMR

Launch an Amazon EMR cluster with Spark and Hive

```
aws emr create-cluster \
  --name "demo" \
  --release-label emr-4.5.0 \
  --instance-type m3.xlarge \
  --instance-count 2 \
  --ec2-attributes KeyName=YOUR-AWS-SSH-KEY \
  --use-default-roles \
  --applications Name=Hive Name=Spark Name=Zeppelin-Sandbox
```

Record your *ClusterId* from the output.

Access Control with IAM

Create an IAM role to allow Redshift to copy from S3 bucket

```
aws iam create-role --role-name redshift-role \
--assume-role-policy-document file://redshift-policy.json
```

Copy the value in “arn” in the output,
e.g., *arn:aws:iam::123456789:role/redshift-role*

```
aws iam put-role-policy --role-name redshift-role \
--policy-name redshift-s3 \
--policy-document file://redshift-s3-policy.json
```

Data Analysis with Amazon Redshift

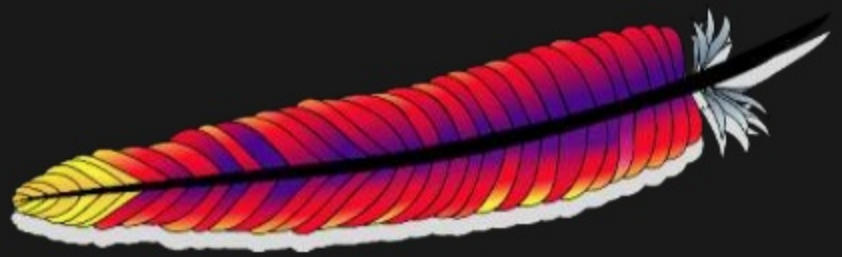
Create a single-node Amazon Redshift data warehouse:

```
aws redshift create-cluster \
  --cluster-identifier demo \
  --db-name demo \
  --node-type dc1.large \
  --cluster-type single-node \
  --iam-roles "arn:aws:iam::YOUR-AWS-ACCOUNT:role/redshift-
copy-role" \
  --master-username master \
  --master-user-password YOUR-REDSHIFT-PASSWORD \
  --publicly-accessible \
  --port 8192
```

Collect

Weblogs – Common Log Format (CLF)

```
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)"
```



Writing into Amazon Kinesis Firehose

Download the demo weblog: `http://bit.ly/aws-big-data`

Open Python and run the following code to import the log into the stream:

```
import boto3
iam = boto3.client('iam')
firehose = boto3.client('firehose')

with open('weblog', 'r') as f:
    for line in f:
        firehose.put_record(
                DeliveryStreamName='demo-firehose-stream',
                Record={'Data': line}
                )
        print 'Record added'
```

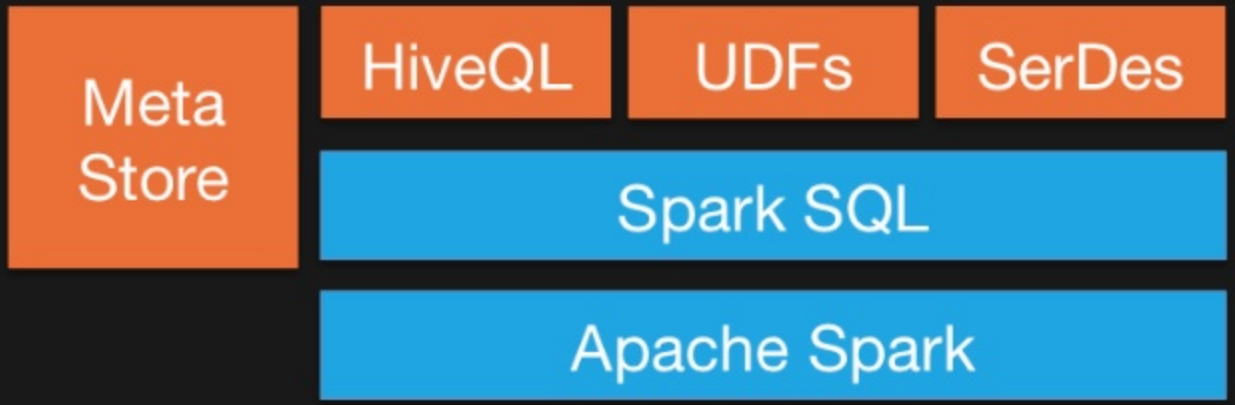
Process

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

Spark SQL

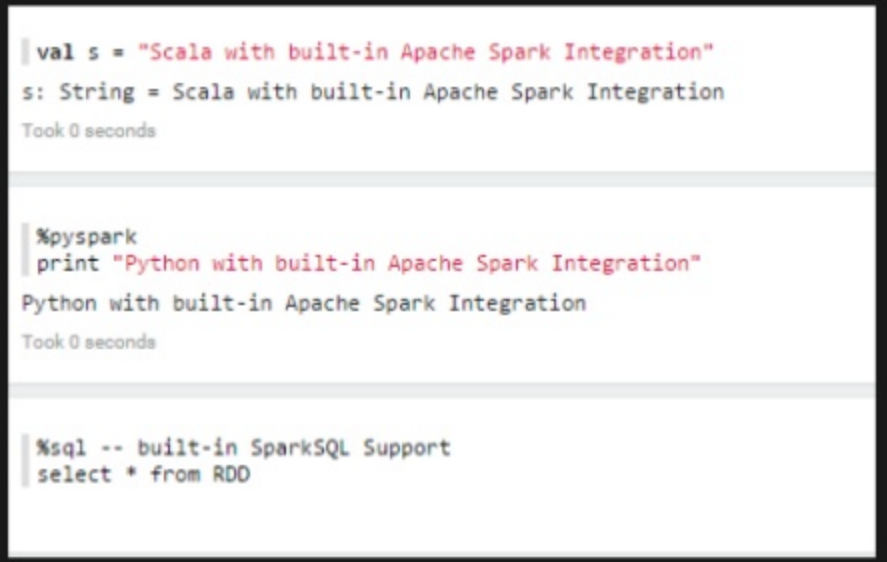
Spark's module for working with structured data using SQL



Run unmodified Hive queries on existing data

Apache Zeppelin

- Web-based notebook for interactive analytics
- Multiple language backend
- Apache Spark integration
- Data visualization
- Collaboration



https://zeppelin.incubator.apache.org/

View the Output Files in Amazon S3

After about 1 minute, you should see files in your S3 bucket:

```
aws s3 ls s3://YOUR-S3-BUCKET-NAME/firehose/ --recursive
```

Connect to Your EMR Cluster and Zeppelin

```
aws emr describe-cluster --cluster-id YOUR-EMR-CLUSTER-ID
```

Copy the *MasterPublicDnsName*. Use port forwarding so you can access Zeppelin at http://localhost:18890 on your local machine.

```
ssh -i PATH-TO-YOUR-SSH-KEY -L 18890:localhost:8890 \
hadoop@YOUR-EMR-DNS-NAME
```

Open Zeppelin with your local web browser and create a new “Note”:
`http://localhost:18890`

Exploring the Data in Amazon S3 using Spark

Download the Zeppelin notebook: http://bit.ly/aws-big-data-zeppelin

```
// Load all the files from S3 into a RDD
val accessLogLines = sc.textFile("s3://YOUR-S3-BUCKET-NAME/firehose/*/*/*/*")

// Count the lines
accessLogLines.count

// Print one line as a string
accessLogLines.first

// delimited by space so split them into fields
var accessLogFields = accessLogLines.map(_.split(" ").map(_.trim))

// Print the fields of a line
accessLogFields.first
```

Combine Fields: “A, B, C” → “A B C”

```
var accessLogColumns = accessLogFields
    .map( arrayOfFields => { var temp1 =""; for (field <- arrayOfFields) yield {
        var temp2 = ""
        if (temp1.replaceAll("\\[","\"").startsWith("\"") && !temp1.endsWith("\""))
            temp1 = temp1 + " " + field.replaceAll("\\[|\\]","\"")
        else temp1 = field.replaceAll("\\[|\\]","\"")
        temp2 = temp1
        if (temp1.endsWith("\"")) temp1 = ""
        temp2
    }})
    .map( fields => fields.filter(field => (field.startsWith("\"") &&
field.endsWith("\"")) || !field.startsWith("\"") ))
    .map(fields => fields.map(_.replaceAll("\"","")))
```

Create a Data Frame and Transform the Data

```
import java.sql.Timestamp
import java.net.URL
case class accessLogs(
  ipAddress: String,
  requestTime: Timestamp,
  requestMethod: String,
  requestPath: String,
  requestProtocol: String,
  responseCode: String,
  responseSize: String,
  referrerHost: String,
  userAgent: String
)
```

Create a Data Frame and Transform the Data

```
val accessLogsDF = accessLogColumns.map(line => {
    var ipAddress        = line(0)
    var requestTime      = new Timestamp(new
java.text.SimpleDateFormat("dd/MMM/yyyy:HH:mm:ss Z").parse(line(3)).getTime())
    var requestString    = line(4).split(" ").map(_.trim())
    var requestMethod    = if (line(4).toString() != "-") requestString(0) else ""
    var requestPath      = if (line(4).toString() != "-") requestString(1) else ""
    var requestProtocol = if (line(4).toString() != "-") requestString(2) else ""
    var responseCode     = line(5).replaceAll("-","")
    var responseSize     = line(6).replaceAll("-","")
    var referrerHost     = line(7)
    var userAgent        = line(8)
    accessLogs(ipAddress, requestTime, requestMethod, requestPath,
requestProtocol,responseCode, responseSize, referrerHost, userAgent)
}).toDF()
```

Create an External Table Backed by Amazon S3

```
%sql
CREATE EXTERNAL TABLE access_logs
(
 ip_address String,
 request_time Timestamp,
 request_method String,
 request_path String,
 request_protocol String,
 response_code String,
 response_size String,
 referrer_host String,
 user_agent String
)
PARTITIONED BY (year STRING,month STRING, day STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
LOCATION 's3://YOUR-S3-BUCKET-NAME/access-log-processed'
```

Configure Hive Partitioning and Compression

```
// set up Hive's "dynamic partitioning”
%sql
SET hive.exec.dynamic.partition=true

// compress output files on Amazon S3 using Gzip
%sql
SET hive.exec.compress.output=true

%sql
SET mapred.output.compression.codec=org.apache.hadoop.io.compress.GzipCodec

%sql
SET io.compression.codecs=org.apache.hadoop.io.compress.GzipCodec

%sql
SET hive.exec.dynamic.partition.mode=nonstrict;
```

Write Output to Amazon S3

```
import org.apache.spark.sql.SaveMode
accessLogsDF
    .withColumn("year", year(accessLogsDF("requestTime")))
    .withColumn("month", month(accessLogsDF("requestTime")))
    .withColumn("day", dayofmonth(accessLogsDF("requestTime")))
    .write
    .partitionBy("year","month","day")
    .mode(SaveMode.Overwrite)
    .insertInto("access_logs")
```

Query the Data Using Spark SQL

```
// Check the count of records
%sql
select count(*) from access_log_processed


// Fetch the first 10 records
%sql
select * from access_log_processed limit 10
```

View the Output Files in Amazon S3

Leave Zeppelin and go back to the console…

List the partition prefixes and output files:

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

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
(
        host_address varchar(512),
        request_time timestamp,
        request_method varchar(5),
        request_path varchar(1024),
        request_protocol varchar(10),
        response_code Int,
        response_size Int,
        referrer_host varchar(1024),
        user_agent varchar(512)
)
DISTKEY(host_address)
SORTKEY(request_time);
```

Loading Data into Amazon Redshift

"COPY" command loads files in parallel from Amazon S3:

```
COPY accesslogs
FROM 's3://YOUR-S3-BUCKET-NAME/access-log-processed'
CREDENTIALS
'aws_iam_role=arn:aws:iam::YOUR-AWS-ACCOUNT-ID:role/ROLE-NAME'
DELIMITER '\t'
MAXERROR 0
GZIP;
```

Amazon Redshift Test Queries

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

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

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

Visualize the Results

DEMO
Amazon QuickSight

Learn from big data experts

blogs.aws.amazon.com/bigdata

Thank you!