Apache Spark and the Hadoop Ecosystem on AWS

Getting started with Amazon EMR

Keith Steward, Ph.D.,
Specialist (EMR) Solution Architect,
Amazon EMR

October 11, 2016

amazon web services

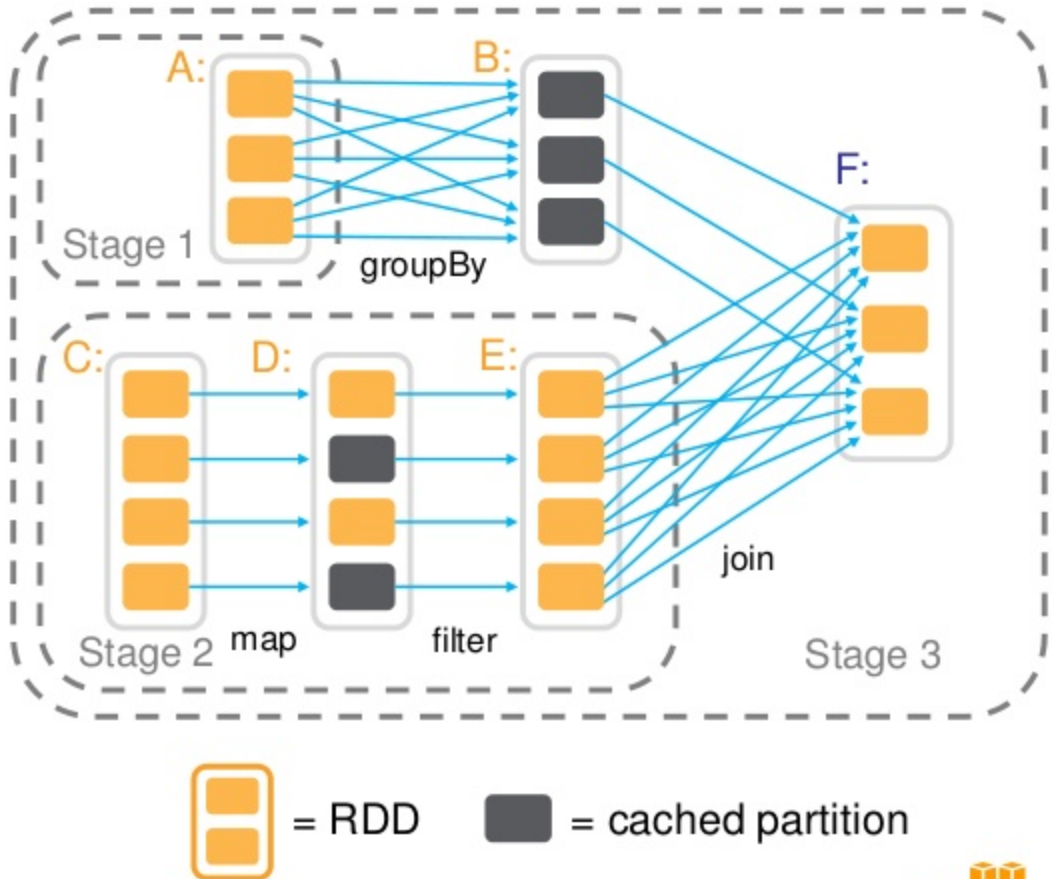
Agenda

1. Quick introduction to Spark, Hive on Tez, and Presto
2. Building data lakes with Amazon EMR and Amazon S3
3. Running jobs and security options
4. Customer use cases
5. Demo(s)

Quick introduction to Spark, Hive on Tez, and Presto

Spark for fast processing

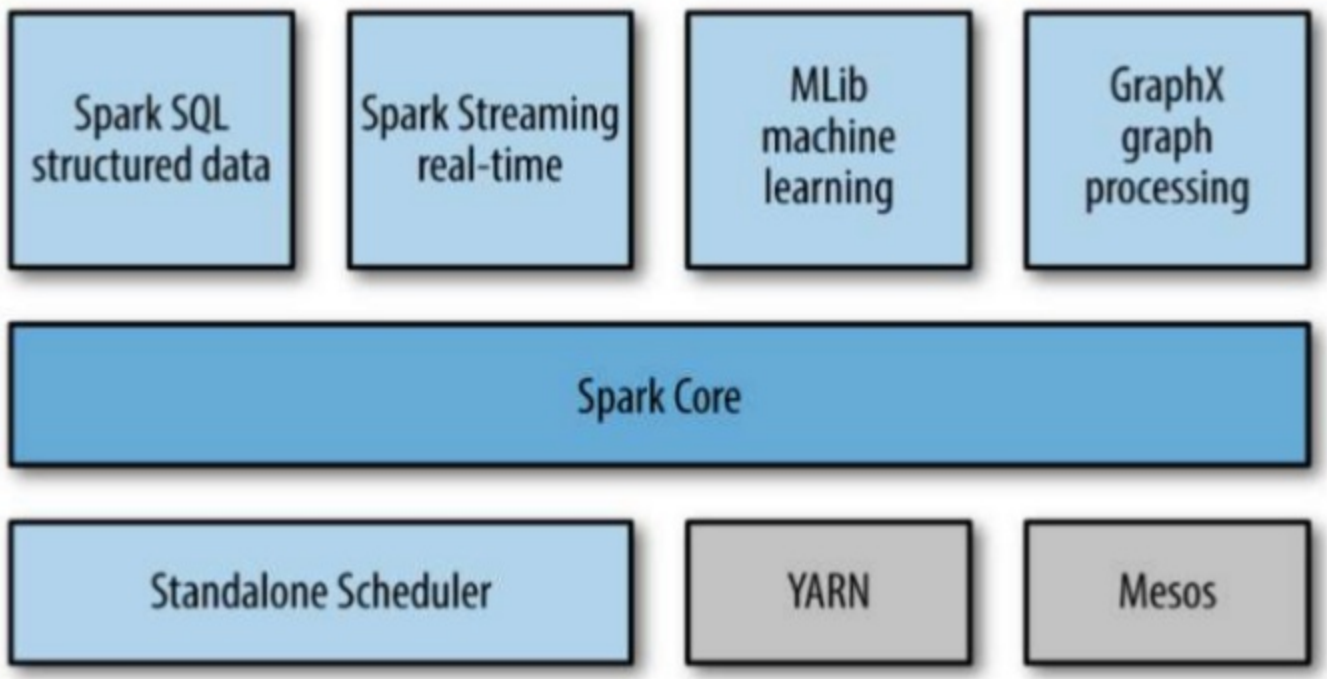
- Massively parallel
- Uses DAGs instead of map-reduce for execution
- Minimizes I/O by storing data in DataFrames in memory
- Partitioning-aware to avoid network-intensive shuffle



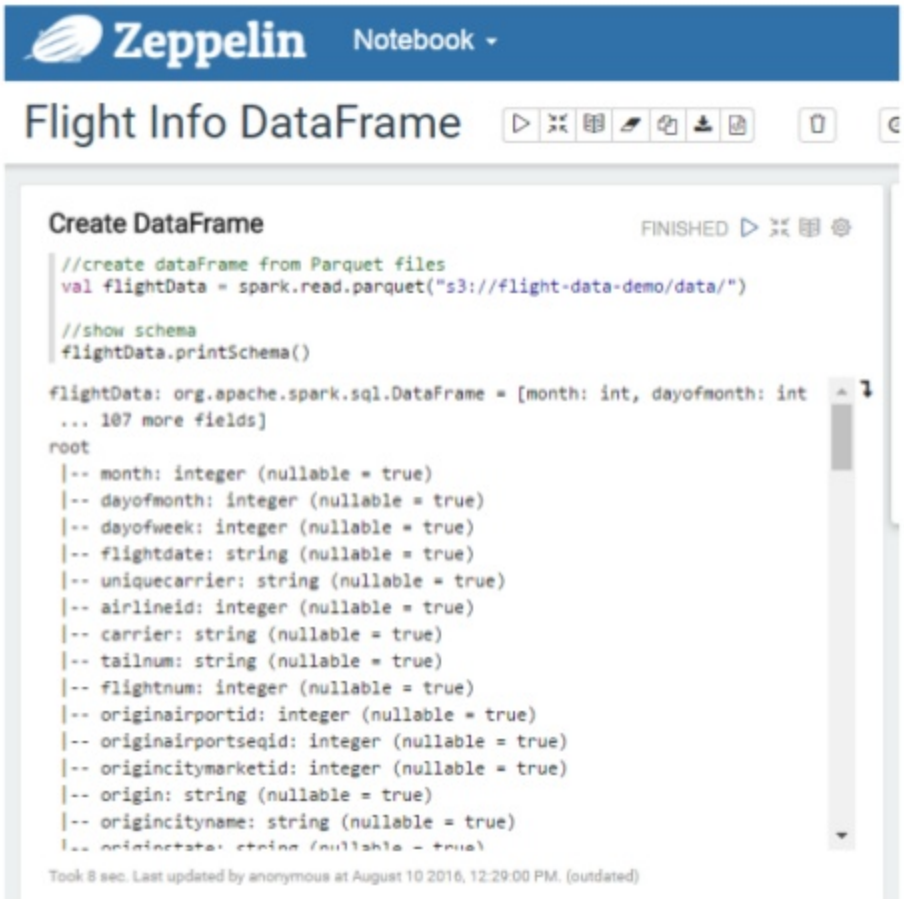
Spark

amazon web services

Spark components to match your use case



Apache Zeppelin notebooks for Spark

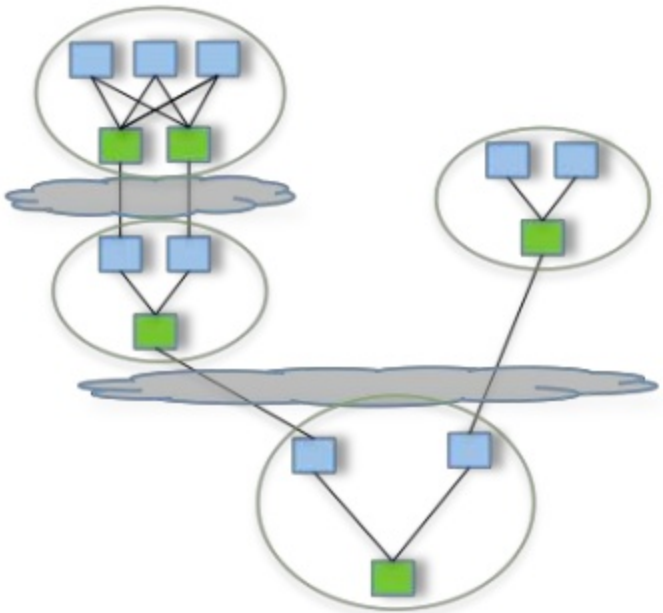



```
// Prepare training documents from a list of (id, text, label) tuple
val training = sqlContext.createDataFrame(Seq(
  (0L, "a b c d e spark", 1.0),
  (1L, "b d", 0.0),
  (2L, "spark f g h", 1.0),
  (3L, "hadoop mapreduce", 0.0)
)).toDF("id", "text", "label")


// Configure an ML pipeline, which consists of three stages: tokeni
val tokenizer = new Tokenizer()
  .setInputCol("text")
  .setOutputCol("words")
val hashingTF = new HashingTF()
  .setNumFeatures(1000)
  .setInputCol(tokenizer.getOutputCol)
  .setOutputCol("features")
val lr = new LogisticRegre
  .setMaxIter(10)
  .setRegParam(0.01)
val pipeline = new Pipeli
  .setStages(Array(tokeni


// Fit the pipeline to tr
val model = pipeline.fi
```

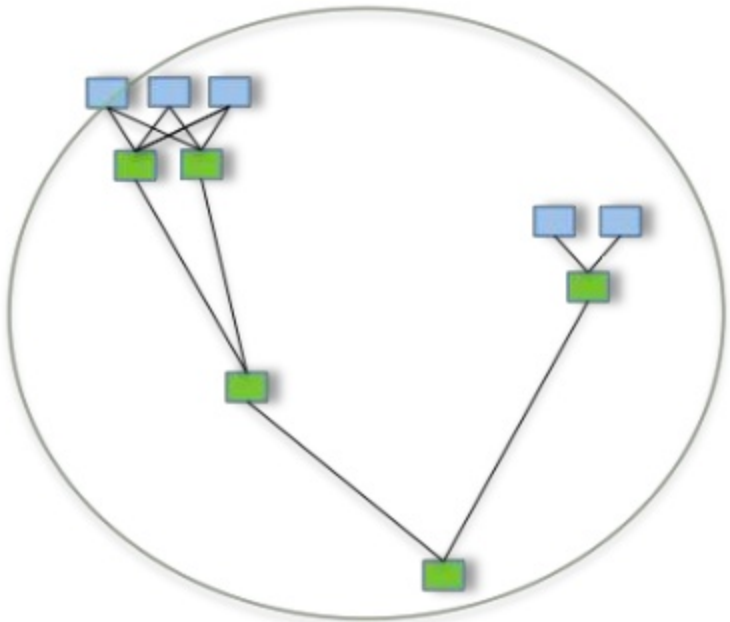
```
Fit                          local
final                        keyword
finally                      keyword
Configure                    local
classification               local
NoSuchFieldException         keyword
NoSuchFieldError             keyword
UnsatisfiedLinkError         keyword
```

amazon web services

Hive & Tez for batch ETL and SQL

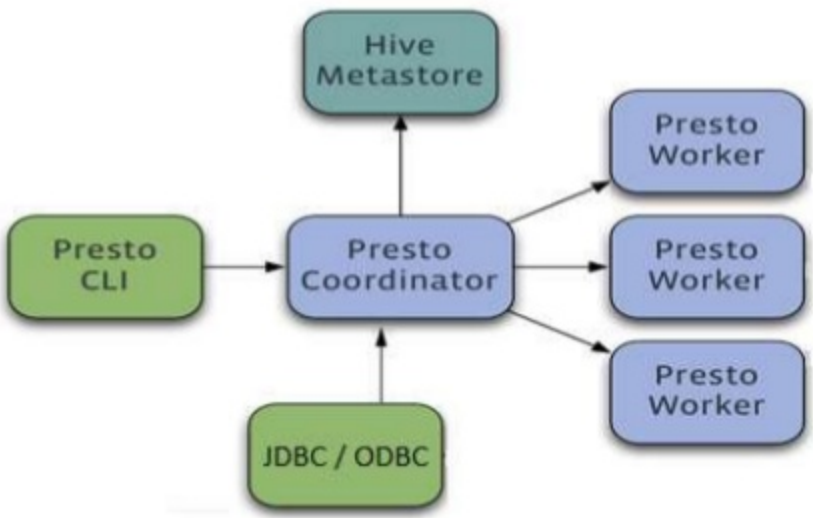


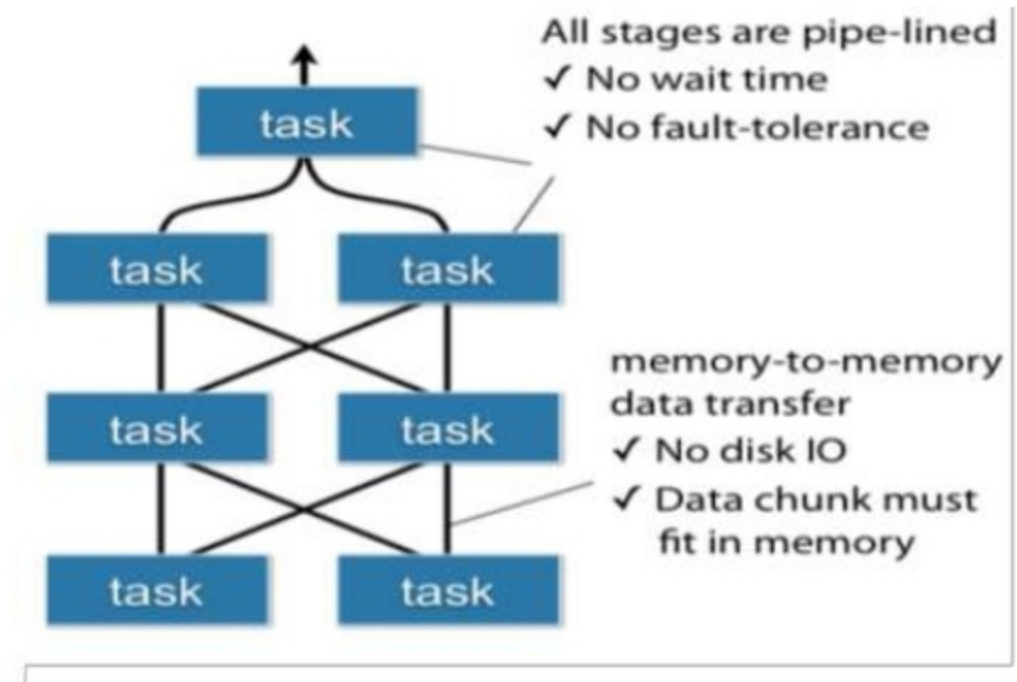
Pig/Hive - MR



Pig/Hive - Tez

Presto: interactive SQL for analytics





presto

amazon web services

Important Presto Features

High Performance

- E.g. Netflix: runs 3500+ Presto queries / day on 25+ PB dataset in S3 with 350 active platform users

Extensibility

- Pluggable backends: Hive, Cassandra, JMX, Kafka, MySQL, PostgreSQL, MySQL, and more
- JDBC, ODBC for commercial BI tools or dashboards
- Client Protocol: HTTP+JSON, support various languages (Python, Ruby, PHP, Node.js, Java(JDBC), C#,…)

ANSI SQL

- complex queries, joins, aggregations, various functions (Window functions)

Building data lakes with Amazon EMR and Amazon S3

Amazon S3

Store anything (object storage)

Scalable / Elastic

99.999999999% durability

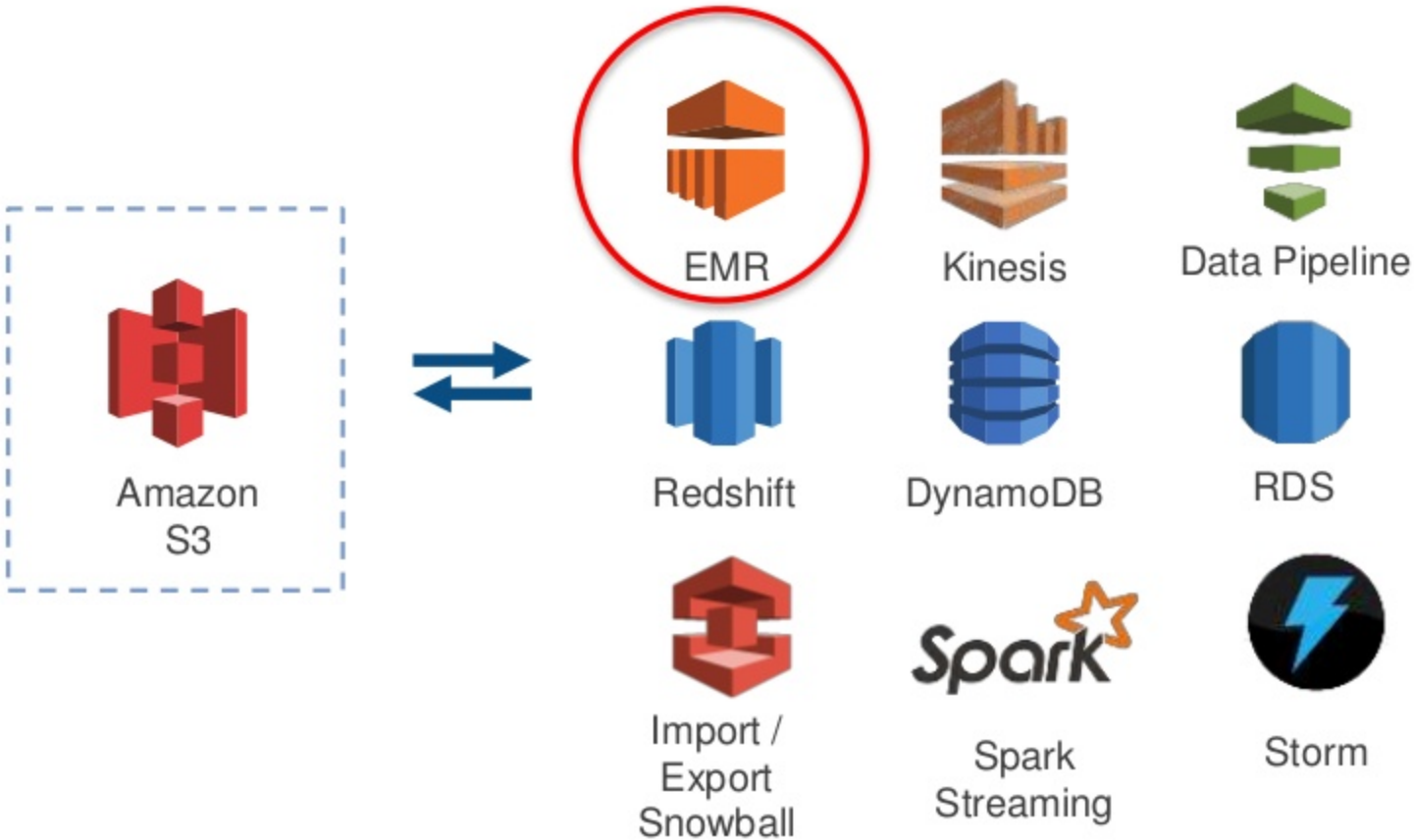
Effectively infinite inbound bandwidth

Extremely low cost: $0.03/GB-Mo; $30.72/TB-Mo

Data layer for virtually all AWS services

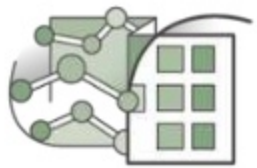
amazon web services

Future-Proof Data: Aggregate all Data in S3 as your *Data Lake*
Surrounded by a collection of the right tools



amazon web services

Why Amazon EMR?



Easy to Use
Launch a cluster in minutes

Low Cost
Pay an hourly rate

Open-Source Variety
Latest versions of software

Managed
Spend less time monitoring

Secure
Easy to manage options

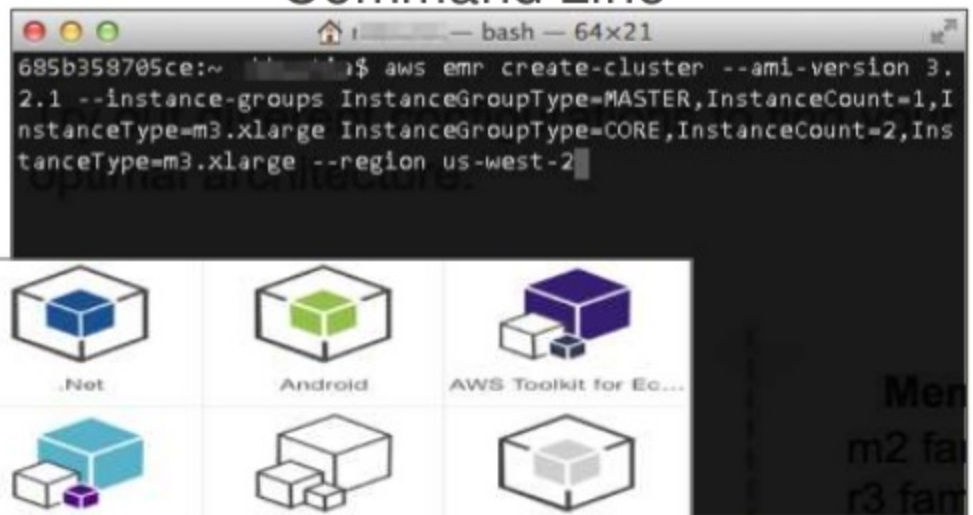
Flexible
Customize the cluster

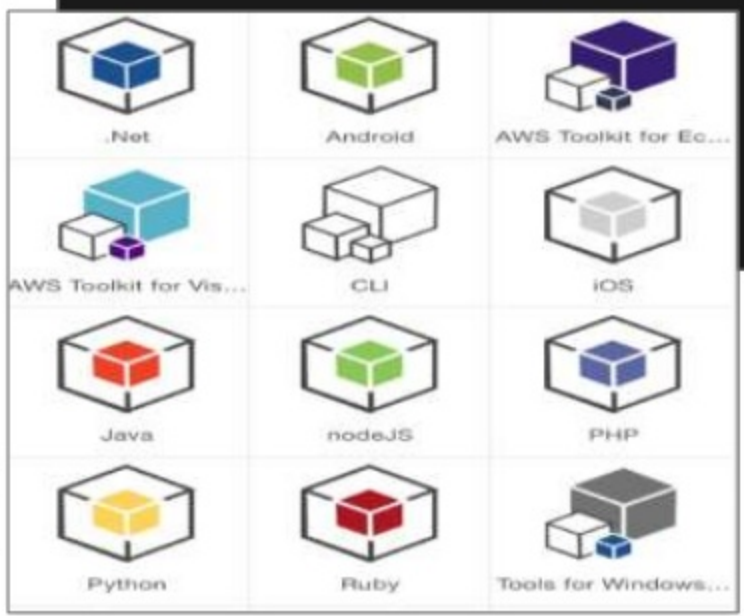
amazon web services

Create a fully configured cluster with the latest versions of Presto and Spark in minutes

AWS Management Console

Command Line



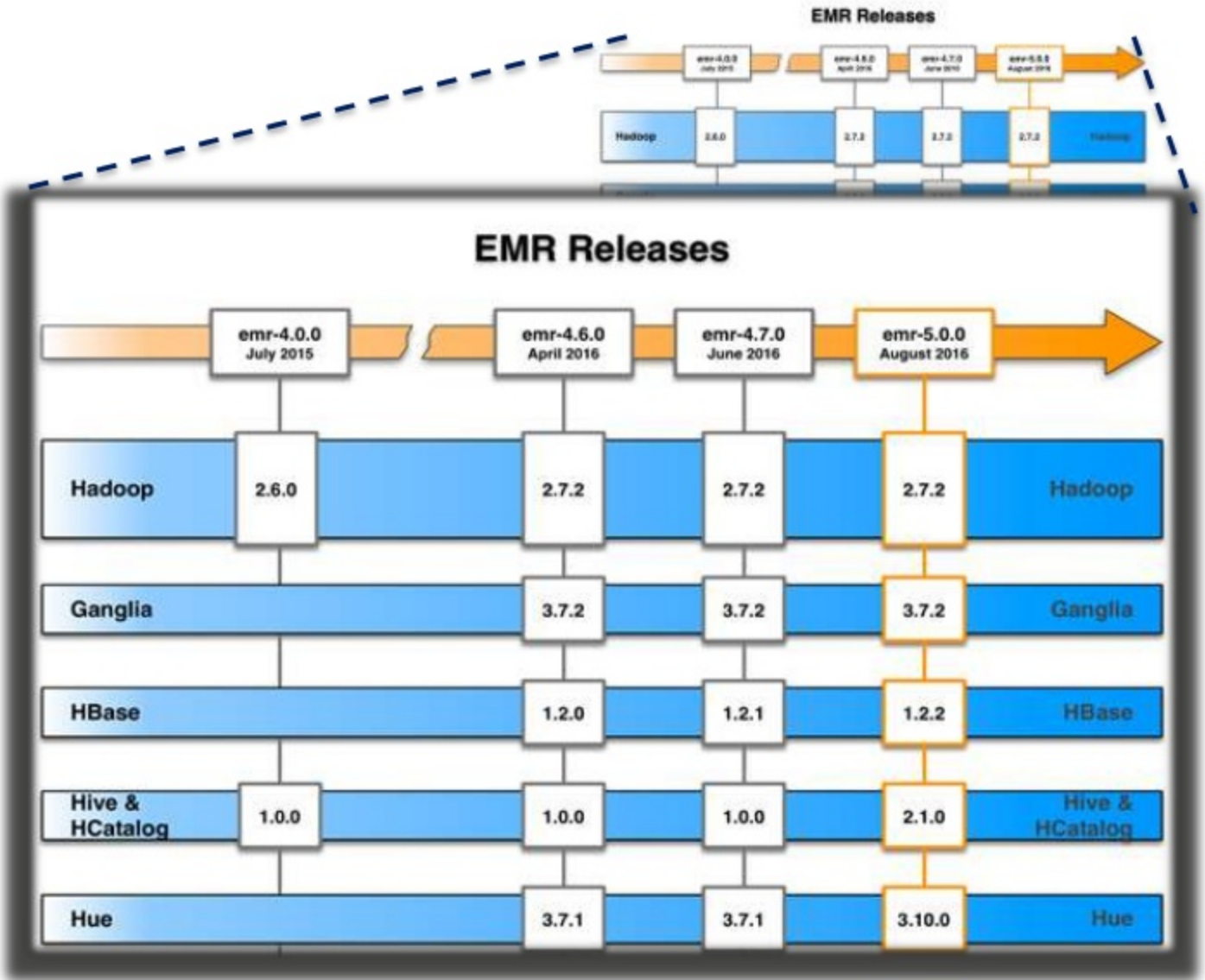


or use the EMR API with your favorite SDK

amazon web services

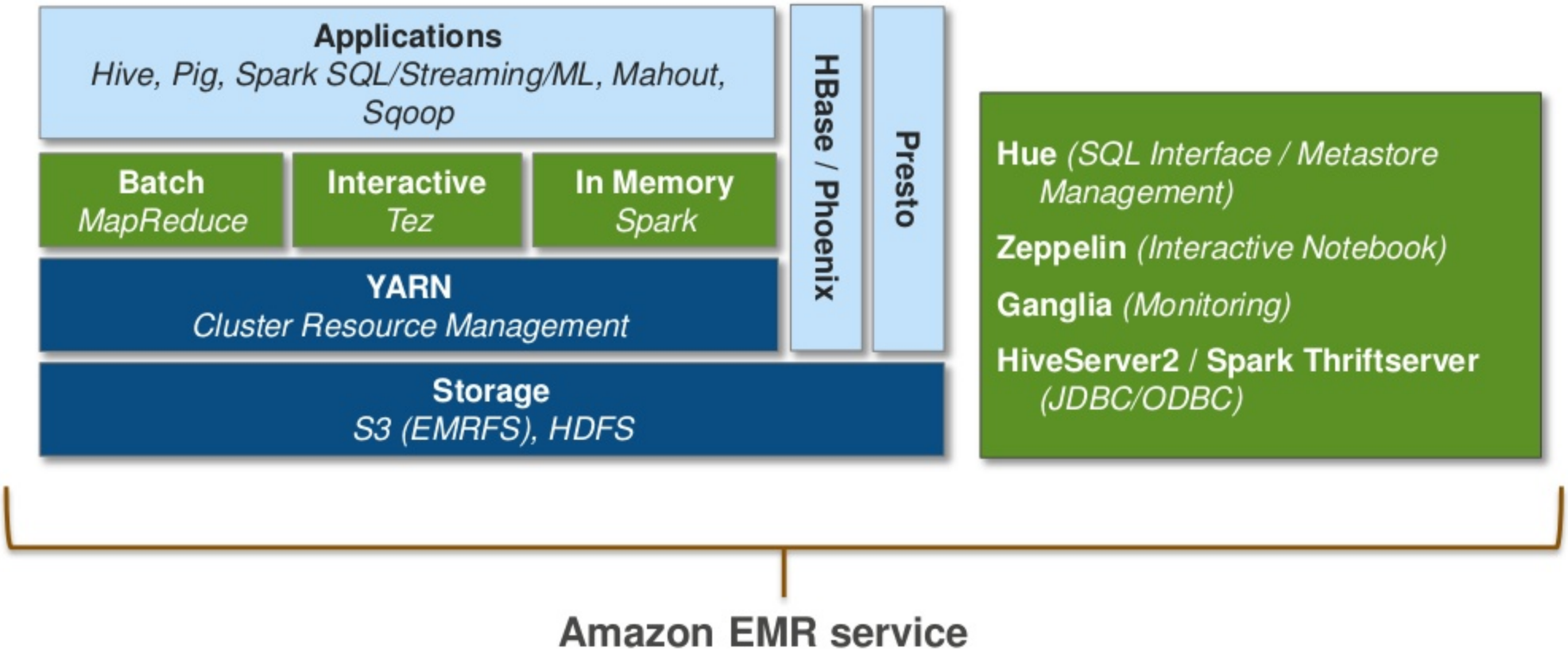
EMR 5.0 - Applications

- [x] Hadoop 2.7.2
- [ ] Ganglia 3.7.2
- [x] Hive 2.1.0
- [ ] Sqoop 1.4.6
- [ ] Phoenix 4.7.0
- [ ] HCatalog 2.1.0
- [ ] Zeppelin 0.6.1
- [ ] HBase 1.2.2
- [ ] Presto 0.150
- [ ] Mahout 0.12.2
- [ ] Oozie 4.2.0
- [ ] Tez 0.8.4
- [ ] Pig 0.16.0
- [ ] ZooKeeper 3.4.8
- [ ] Hue 3.10.0
- [x] Spark 2.0.0



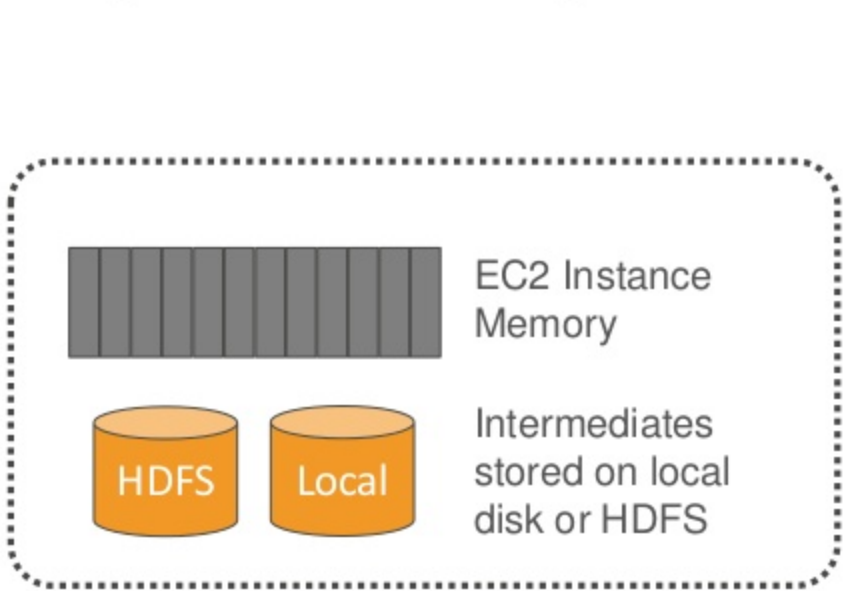
See AWS documentation website for components and versions in each EMR release

amazon web services

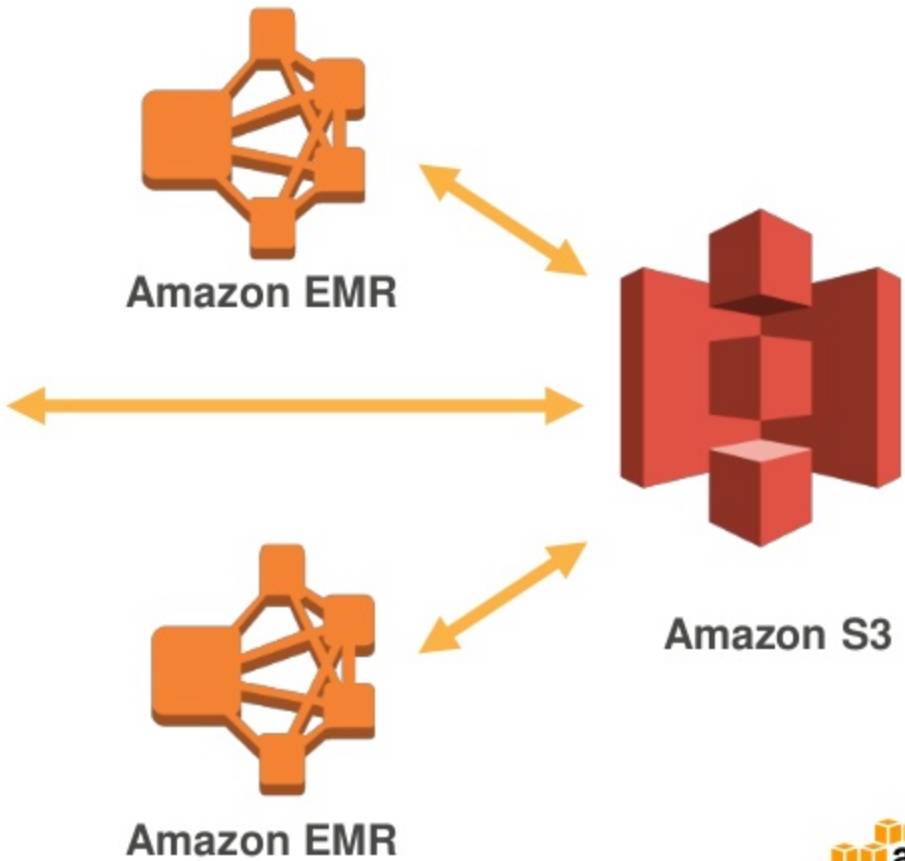


amazon web services

Decouple compute and storage by using S3 as your data layer



S3 is designed for 11 9's of durability and is massively scalable



S3 tips: Partitions, compression, and file formats

- Avoid key names in lexicographical order
- Improve throughput and S3 list performance
- Use hashing / random prefixes, or reverse the date-time
- Compress data set to minimize bandwidth from S3 to EC2
    - Make sure you use splittable compression or have each file be the optimal size for parallelization on your cluster
- Columnar file formats like Parquet can give increased performance on reads

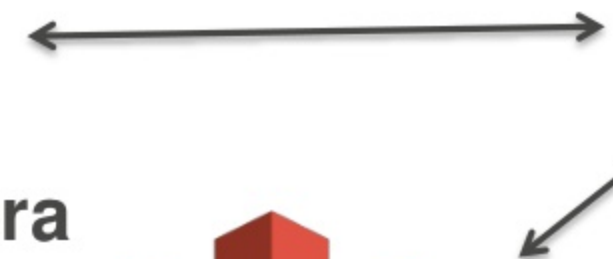
Many storage layers to choose from

Use RDS / Aurora for an external Hive metastore

Hive Metastore for
external tables on S3

Amazon Aurora

Amazon S3



Set metastore location
in `hive-site`

amazon web services

Use Spot & Reserved Instances to lower costs

Meet SLA at predictable cost

Exceed SLA at lower cost

On-demand for core nodes

Standard Amazon EC2 pricing for on-demand capacity

Spot for task nodes

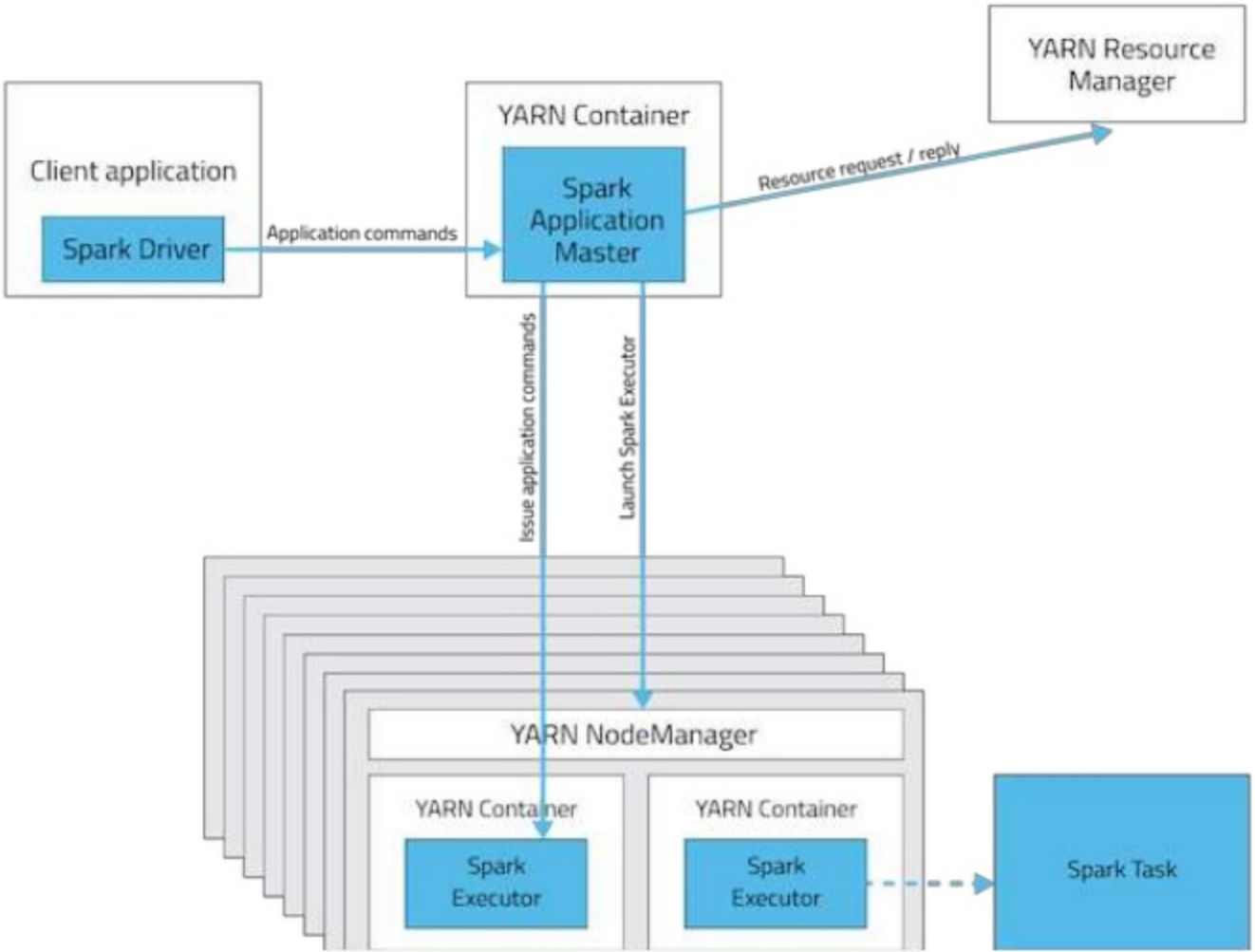
Up to 80% discount compared to on-demand pricing

Amazon EMR supports most EC2 instance types

amazon web services

Running Jobs and Security Options

Amazon EMR runs Spark & Tez on YARN



- Run Spark Driver in Client or Cluster mode
- Spark and Tez applications run as a YARN application
- Spark Executors and Tez Workers run in YARN Containers on NodeManagers in your cluster
- Presto uses its own resource manager

amazon web services

YARN schedulers - CapacityScheduler

- Default scheduler specified in Amazon EMR
- Queues
  - Single queue is set by default
  - Can create additional queues for workloads based on multitenancy requirements
- Capacity Guarantees
  - set minimal resources for each queue
  - Programmatically assign free resources to queues
- Adjust these settings using the classification `capacity-scheduler` in an EMR configuration object

Configuring Executors – Dynamic Allocation

- Optimal resource utilization
- YARN dynamically creates and shuts down executors based on the resource needs of the Spark application
- Spark uses the executor memory and executor cores settings in the configuration for each executor
- Amazon EMR uses dynamic allocation by default, and calculates the default executor size to use based on the instance family of your Core Group

Options to submit jobs – on cluster

Web UIs: Hue SQL editor,
Zeppelin notebooks,
R Studio, Airpal, and more!

Use Hive and Spark Actions in your Apache Oozie workflow to create DAGs of jobs.

Spark
(start using start-thriftserver.sh)

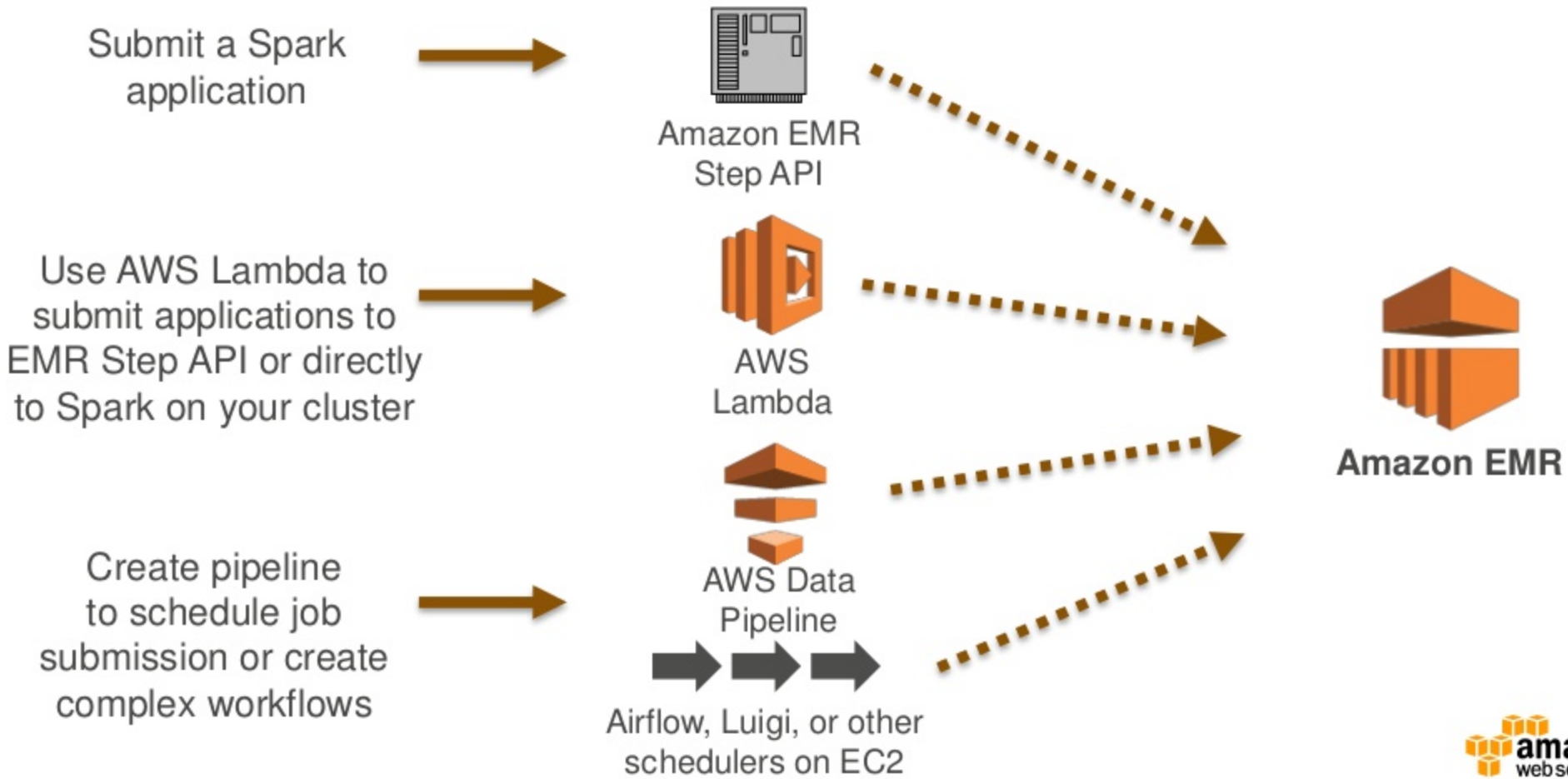
presto

Connect with ODBC / JDBC to
HiveServer2, Spark Thriftserver, or Presto

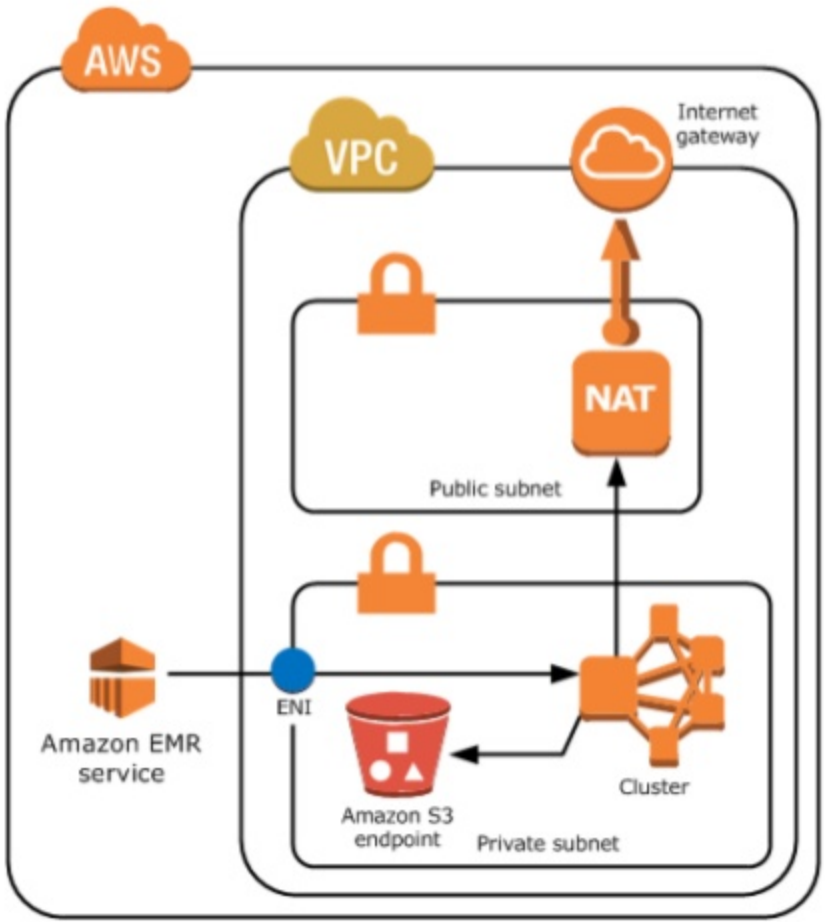
Or, use the native APIs and CLIs for each application

amazon web services

Options to submit jobs to EMR – off cluster



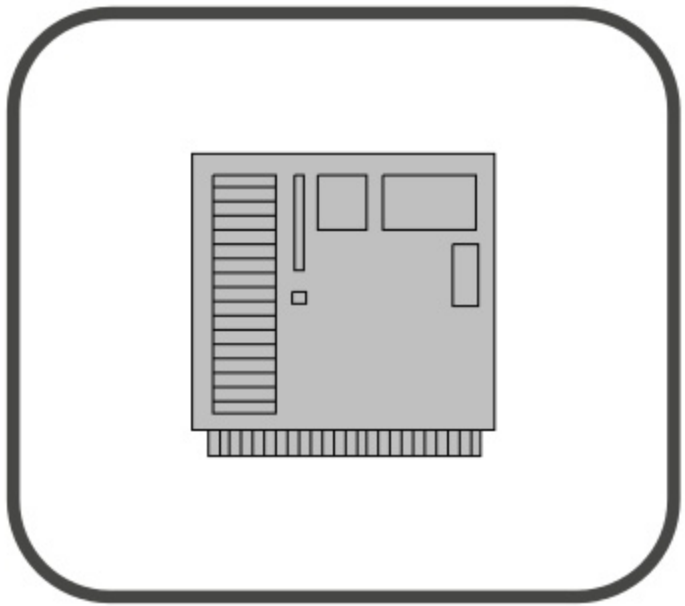
Security - configuring VPC subnets



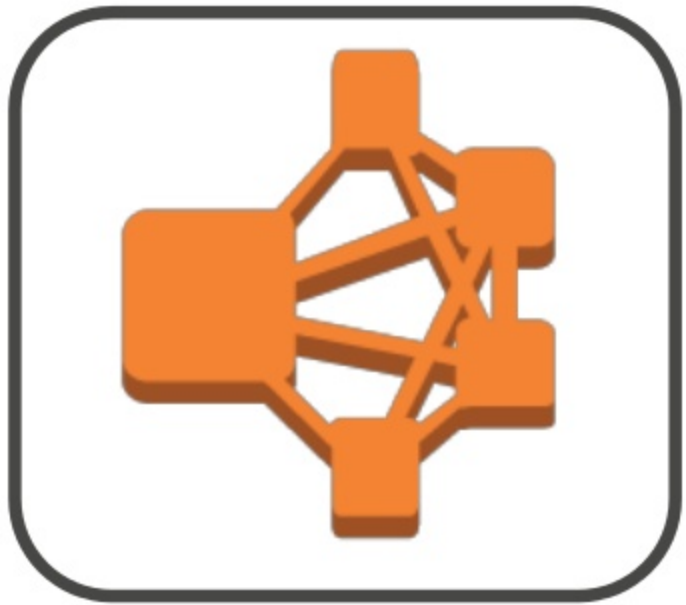
- Use Amazon S3 Endpoints in VPC for connectivity to S3
- Use Managed NAT for connectivity to other services or the Internet
- Control the traffic using Security Groups
  - ElasticMapReduce-Master-Private
  - ElasticMapReduce-Slave-Private
  - ElasticMapReduce-ServiceAccess

amazon web services

IAM Roles – managed or custom policies

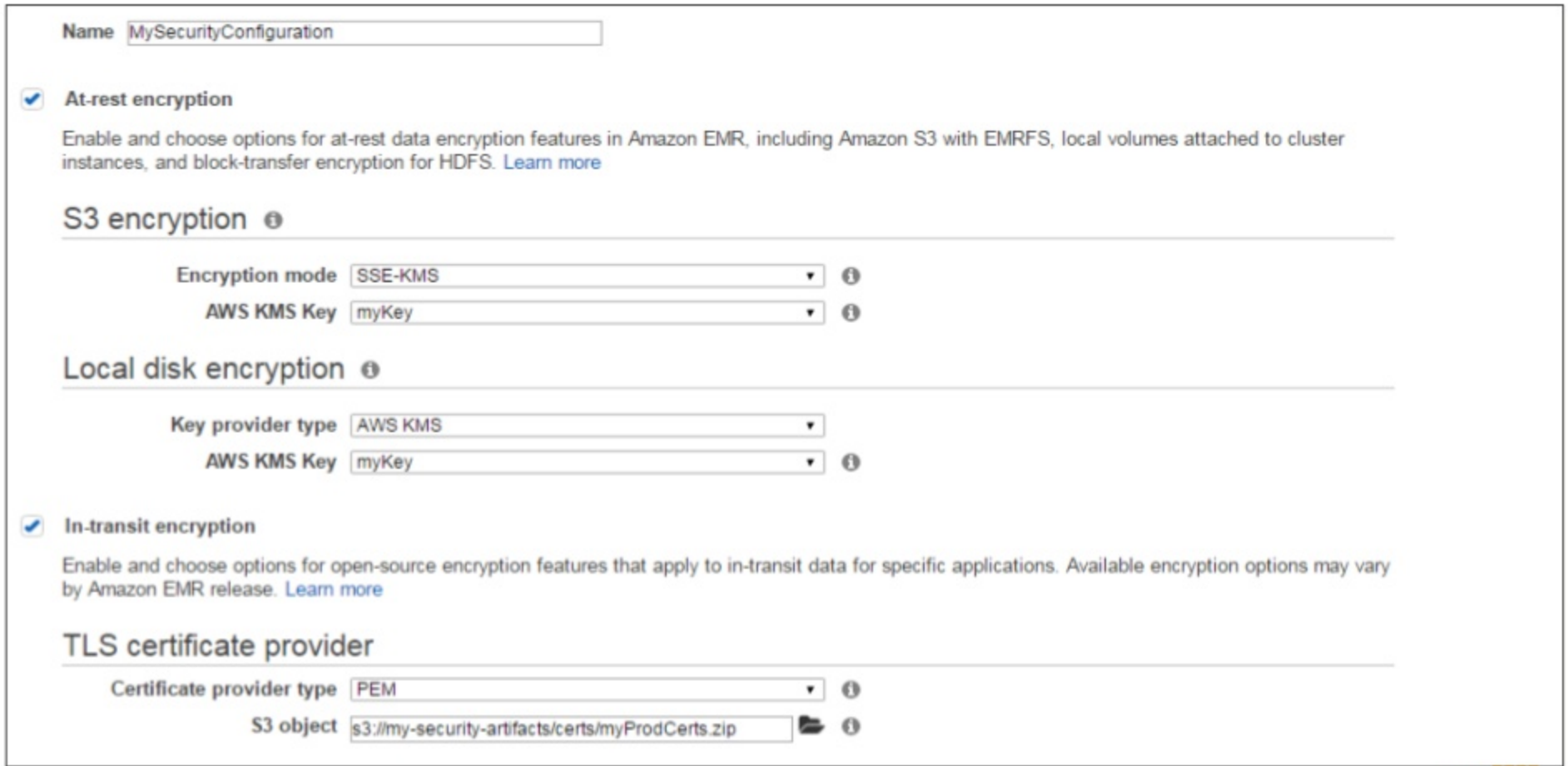


EMR Service Role



EC2 Role

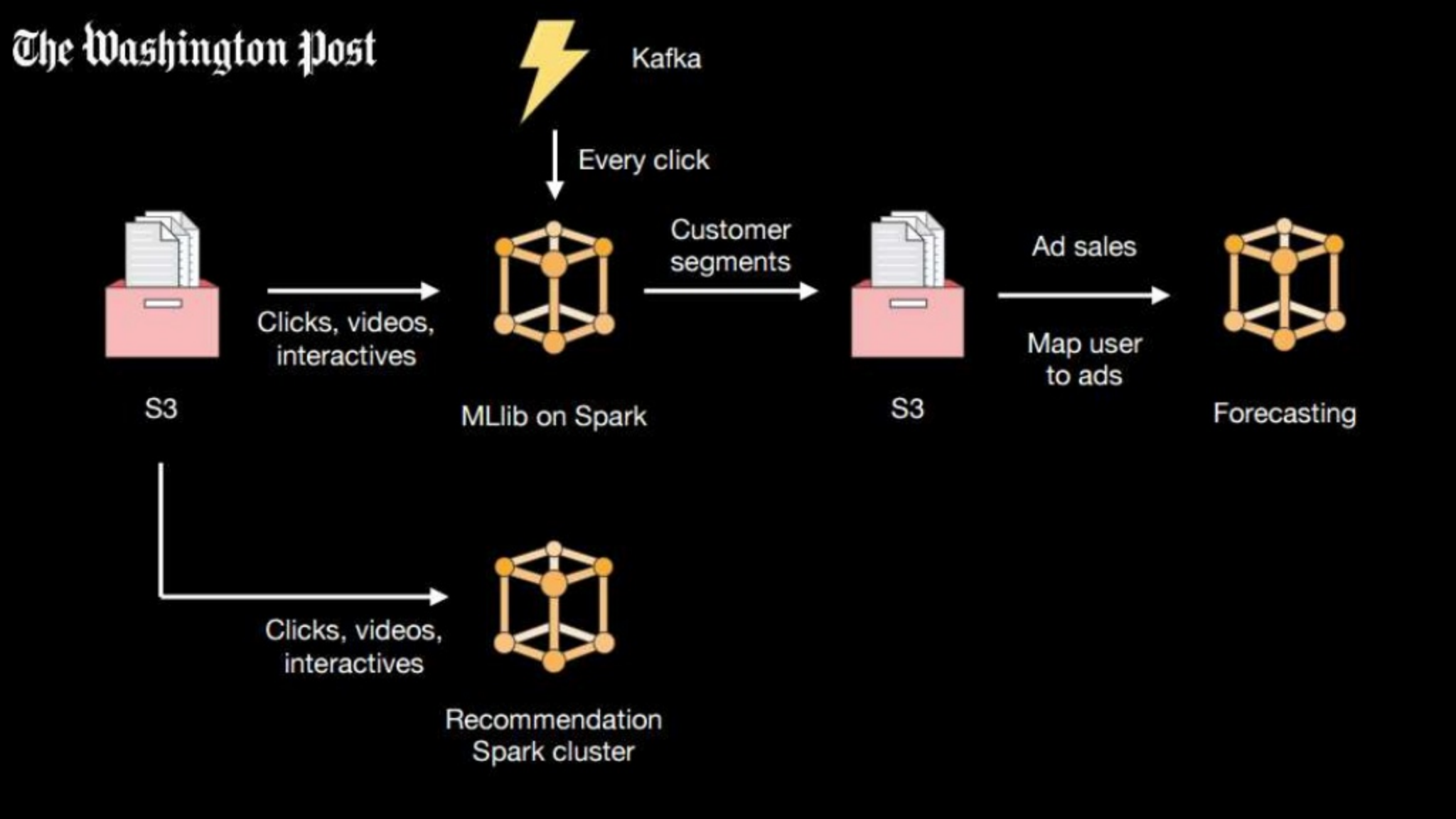
Encryption – use security configurations (New!)

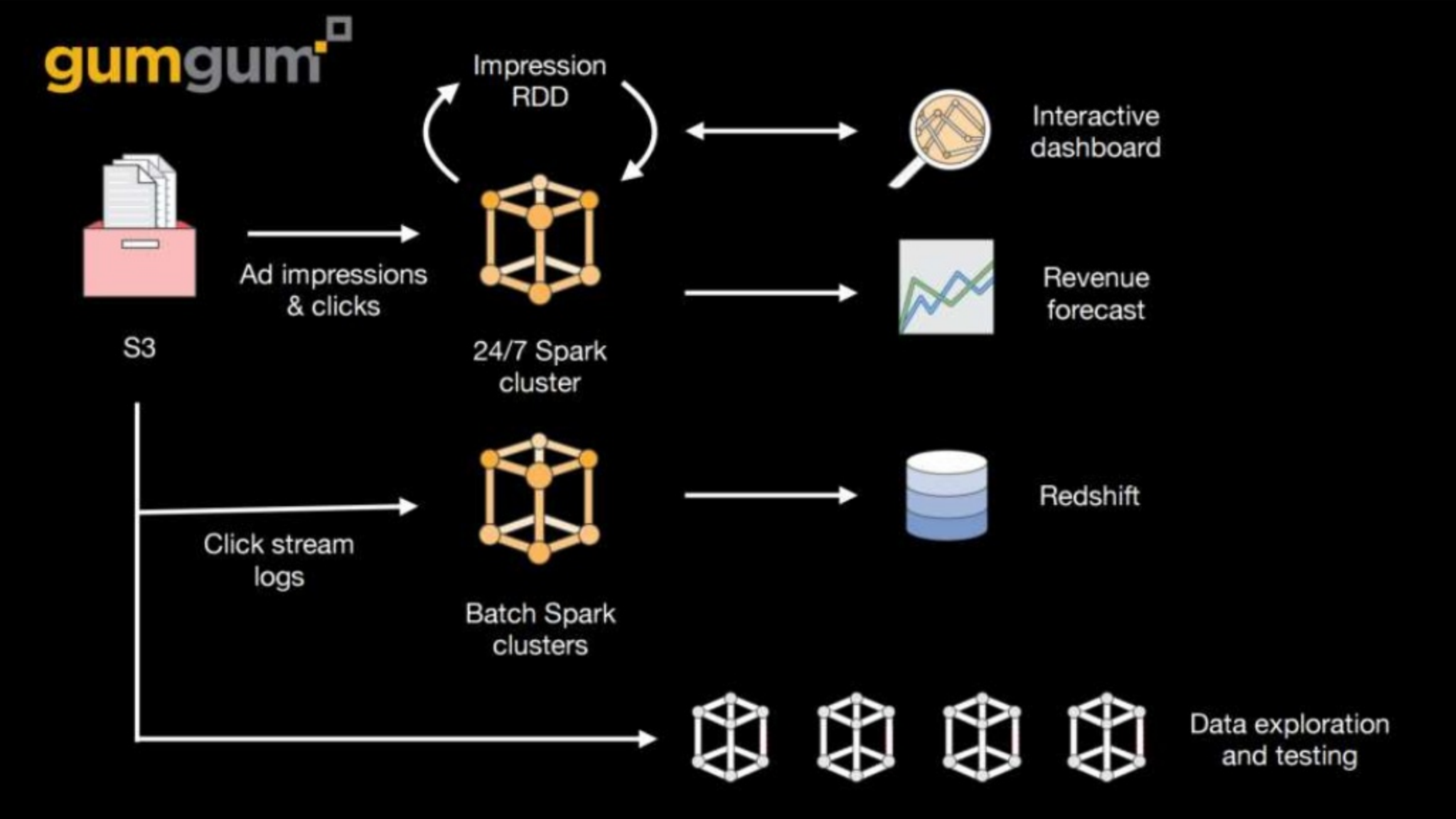


Note: Presto doesn't support in-transit encryption at this time

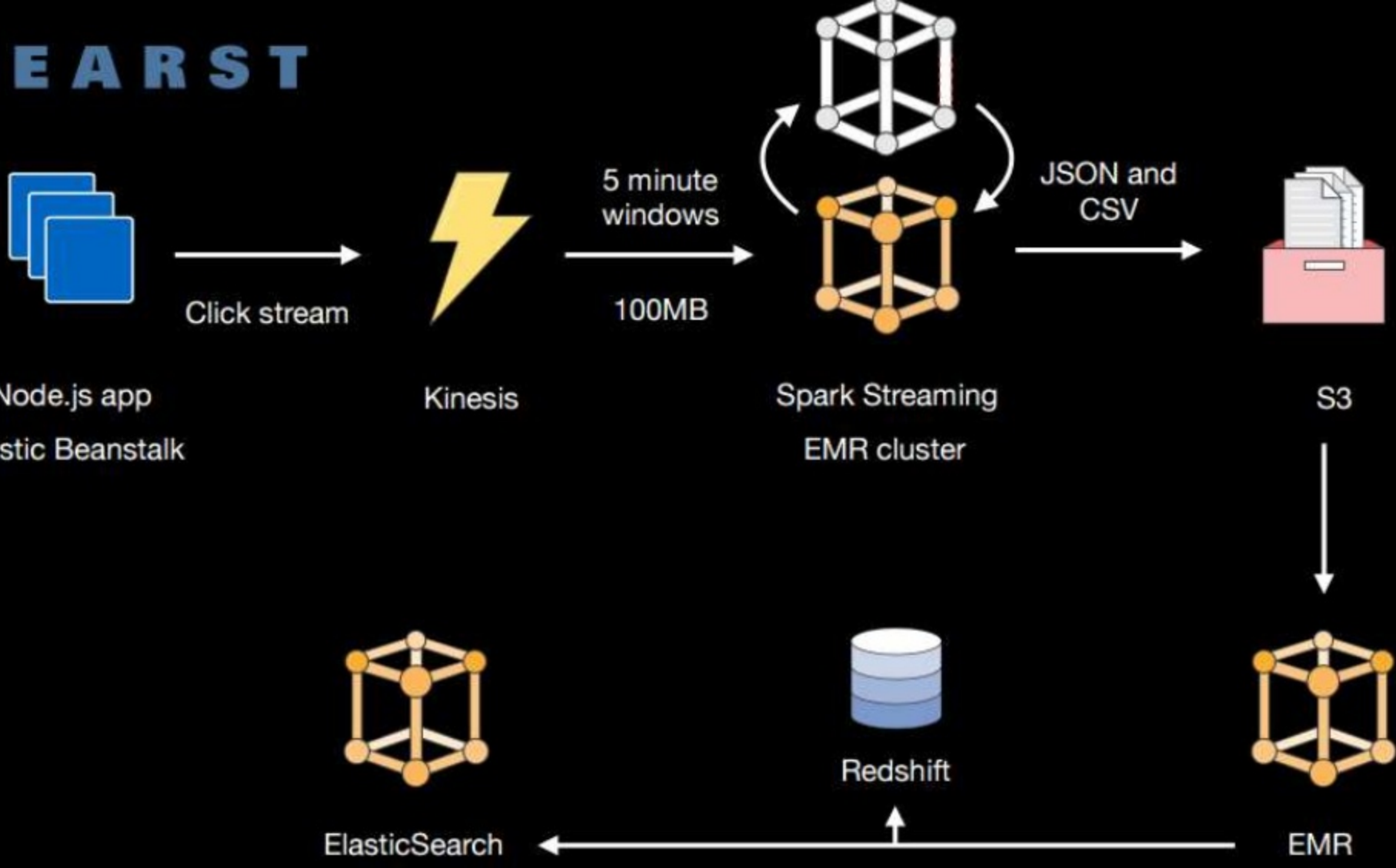
Customer Use Cases

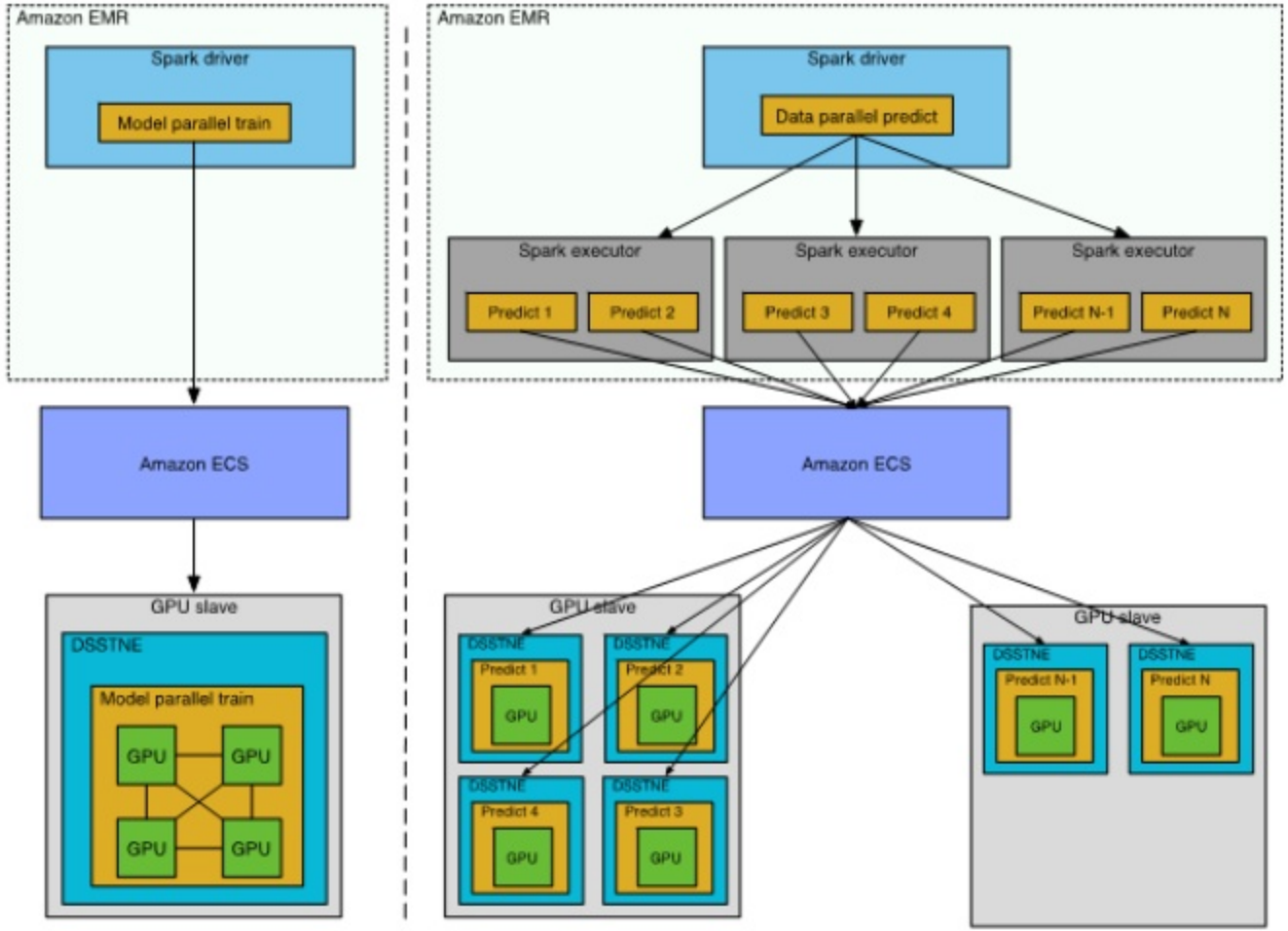
Just some of the organizations using EMR:


The Washington Post
Kafka
Every click
Clicks, videos, interactives
S3
MLlib on Spark
Customer segments
S3
Ad sales
Map user to ads
Forecasting
Clicks, videos, interactives
Recommendation Spark cluster


gumgum
Impression RDD
Interactive dashboard
Ad impressions & clicks
S3
24/7 Spark cluster
Revenue forecast
Click stream logs
Batch Spark clusters
Redshift
Data exploration and testing

HEARST

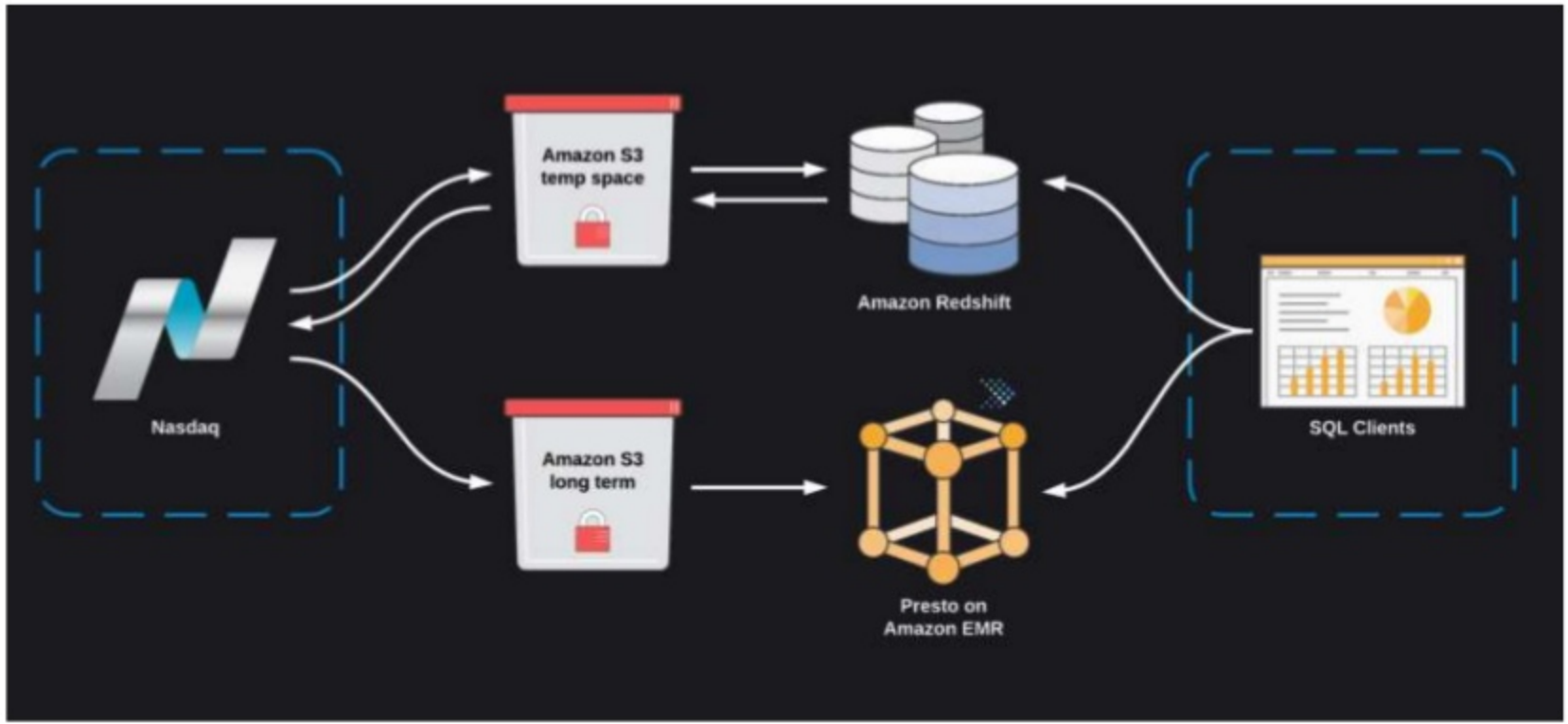

Click stream
5 minute windows
100MB
JSON and CSV
Node.js app
Elastic Beanstalk
Kinesis
Spark Streaming
EMR cluster
S3
Redshift
ElasticSearch
EMR



Amazon.com Personalization Team Using Spark + DSSTNE
http://blogs.aws.amazon.com/bigdata/

amazon web services

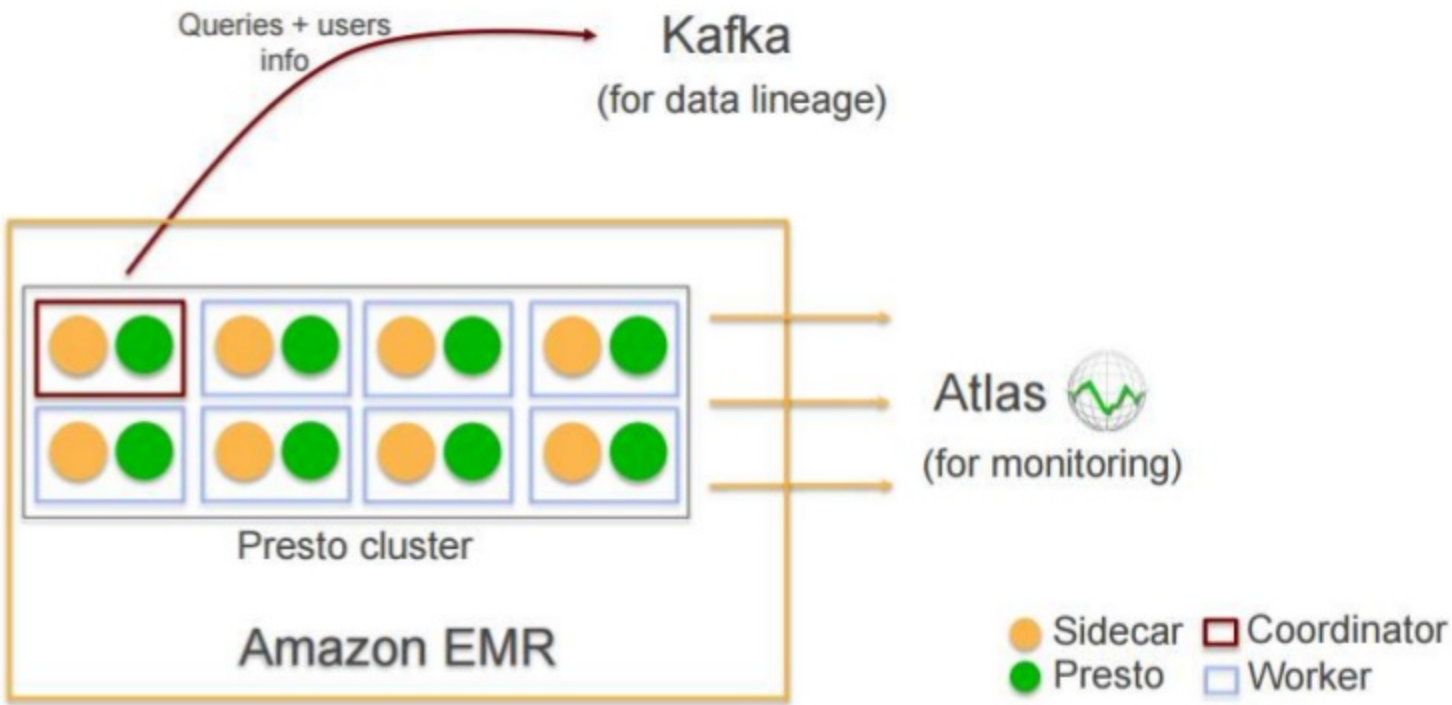
Nasdaq uses Presto on Amazon EMR and Amazon Redshift as a tiered data lake



Full Presentation: https://www.youtube.com/watch?v=LuHxnOQarXU

amazon web services

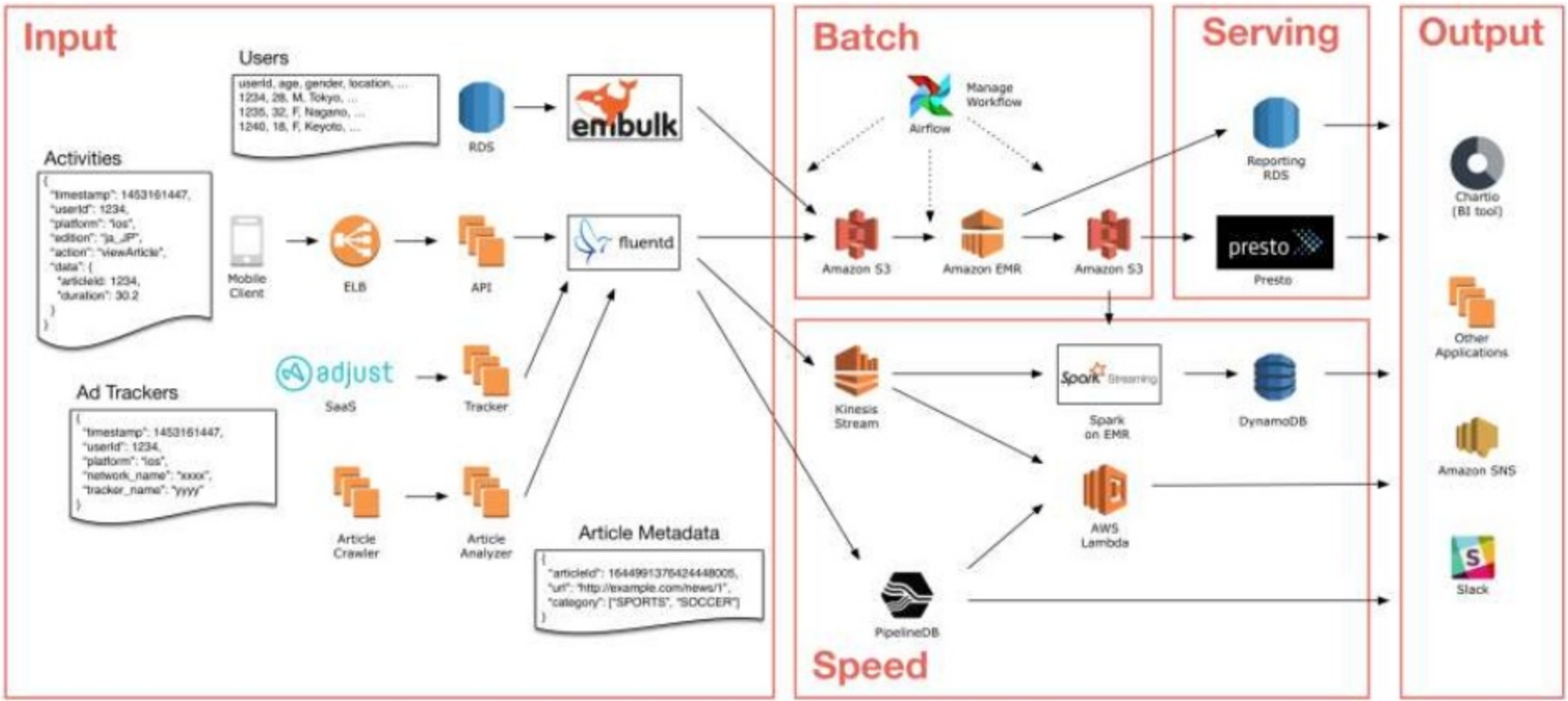
Netflix uses Presto on Amazon EMR with a 25 PB dataset in Amazon S3



Full Presentation: https://www.youtube.com/watch?v=A4OU6i4AQsI

amazon web services

SmartNews uses Presto as a reporting front-end



AWS Big Data Blog: https://blogs.aws.amazon.com/bigdata/post/Tx2V1BSKGITCMTU/How-SmartNews-Built-a-Lambda-Architecture-on-AWS-to-Analyze-Customer-Behavior-an

amazon web services

Demos:

Demo-1: Spark ML on EMR training a Decision Tree Classifier
http://bit.ly/28LG5yr

Demo-2: Spark on EMR doing Real-Time Stream Processing in Zeppelin
https://goo.gl/MbH1Pf

Demo-2: Real Time Processing Application

Produce

Collect

Process

Analyze

Kinesis Producer (KPL)

Amazon Kinesis

Apache Zeppelin

+

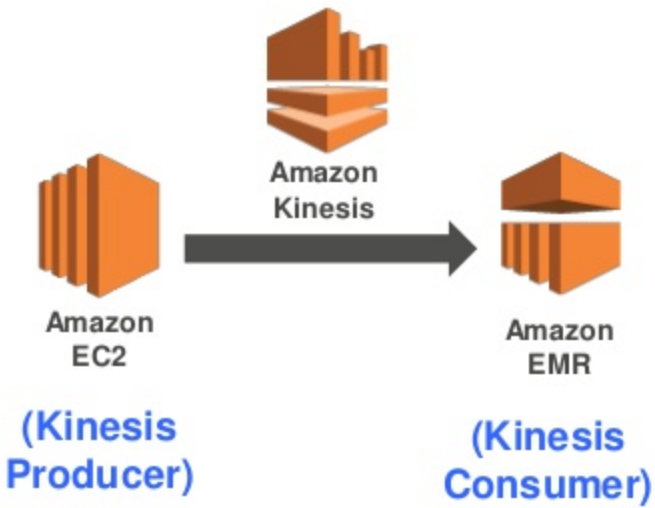
Amazon EMR

+

amazon web services

Real Time Processing Application – 5 Steps

http://bit.ly/realtime-aws



1. Create Kinesis Stream
2. Create Amazon EMR Cluster
3. Start Kinesis Producer
4. Ad-hoc Analytics
   - Analyze data using Apache Zeppelin on Amazon EMR with Spark Streaming

amazon web services

Real Time Processing Application

1. Create Kinesis Stream:

```
$ aws kinesis create-stream --stream-name spark-demo --shard-count 2
```

Real Time Processing Application

2. Create Amazon EMR Cluster with Spark and Zeppelin:

```
$ aws emr create-cluster --release-label emr-5.0.0 \
  --applications Name=Zeppelin Name=Spark Name=Hadoop \
  --enable-debugging \
  --ec2-attributes KeyName=test-key-1,AvailabilityZone=us-east-1d \
  --log-uri s3://kinesis-spark-streaming1/logs \
  --instance-groups \
   Name=Master,InstanceGroupType=MASTER,InstanceType=m3.xlarge,InstanceCount=1 \
   Name=Core,InstanceGroupType=CORE,InstanceType=m3.xlarge,InstanceCount=2 \
   Name=Task,InstanceGroupType=TASK,InstanceType=m3.xlarge,InstanceCount=2 \
  --name "kinesis-processor"
```

Real Time Processing Application

3. Start Kinesis Producer (Using Kinesis Producer Library)

a. On a host (e.g. EC2), download JAR and run Kinesis Producer:

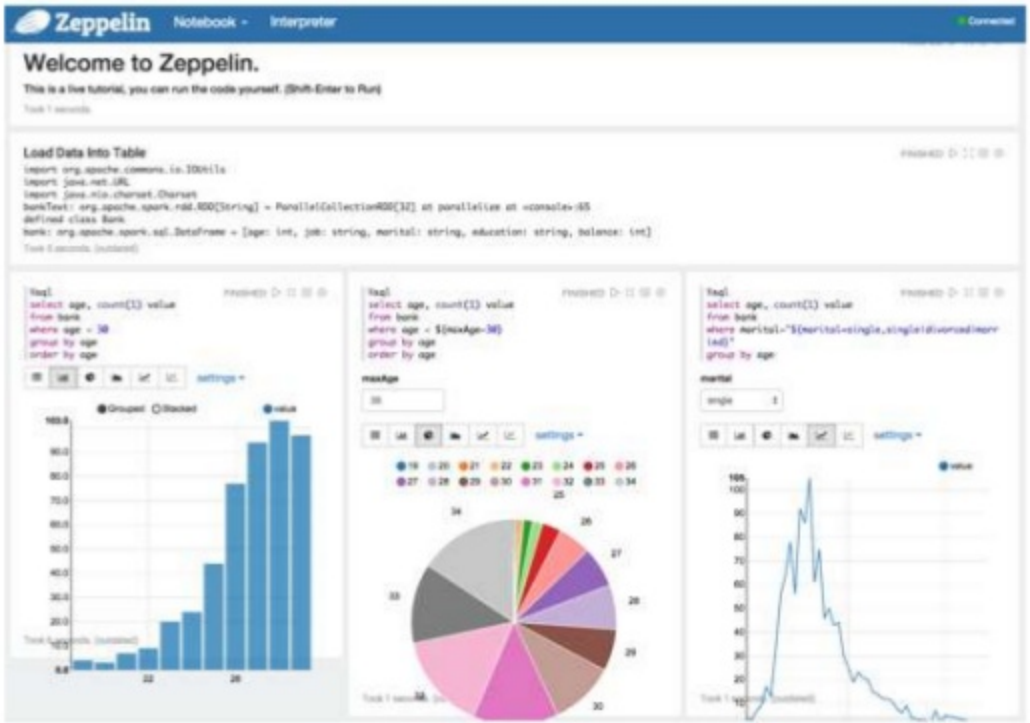
```
$ wget https://s3.amazonaws.com/chayel-emr/KinesisProducer.jar
$ java -jar KinesisProducer.jar
```

b. Continuous stream of data records will be fed into Kinesis in CSV format:

```
… device_id,temperature,timestamp …
```

Git link: `https://github.com/manjeetchayel/emr-kpl-demo`

Real Time Processing Application



Use Zeppelin on Amazon EMR:

- Configure Spark interpreter to use `org.apache.spark:spark-streaming-kinesis-asl_2.11:2.0.0` dependency
- Import notebook from https://raw.githubusercontent.com/manjeetchayel/aws-big-data-blog/master/aws-blog-realtime-analytics-using-zeppelin/Spark_Streaming.json
- Run spark code blocks to generate real-time analytics.

amazon web services

Thank you!

Keith Steward – stewardk@amazon.com

- aws.amazon.com/emr
- blogs.aws.amazon.com/bigdata