# Deep Dive: Amazon EMR

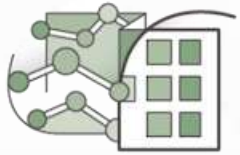
# What is EMR?

Hosted framework that allows you to run Hadoop, Spark and other distributed compute frameworks on AWS

Makes it easy to quickly and cost-effectively process vast amounts of data.

# Just some of the organizations using EMR:

# Why Amazon EMR?



**Easy to Use**
Launch a cluster in minutes

**Low** Cost
Pay an hourly rate

**Elastic**
Easily add or remove capacity

**Reliable**
Spend less time monitoring

**Secure**
Managed firewalls

**Flexible**
You control the cluster

# The Hadoop ecosystem can run in Amazon EMR











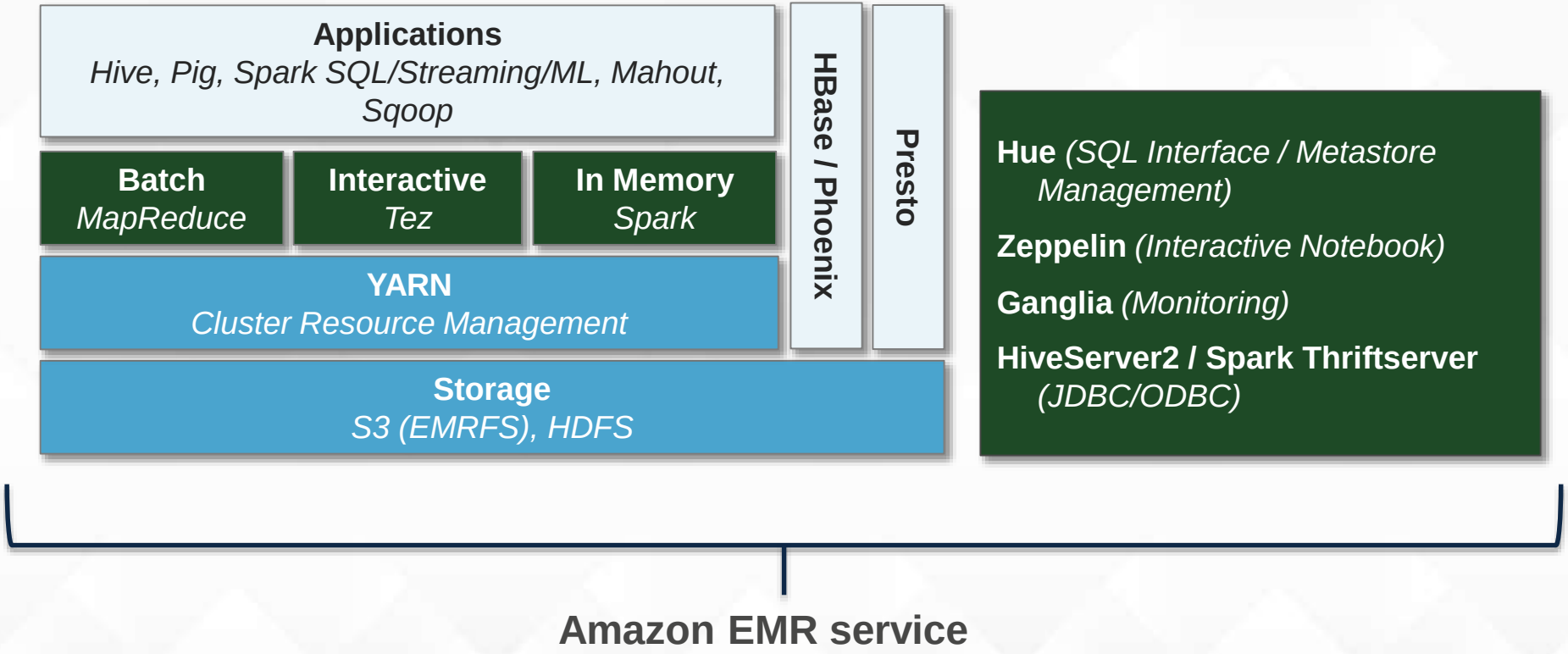






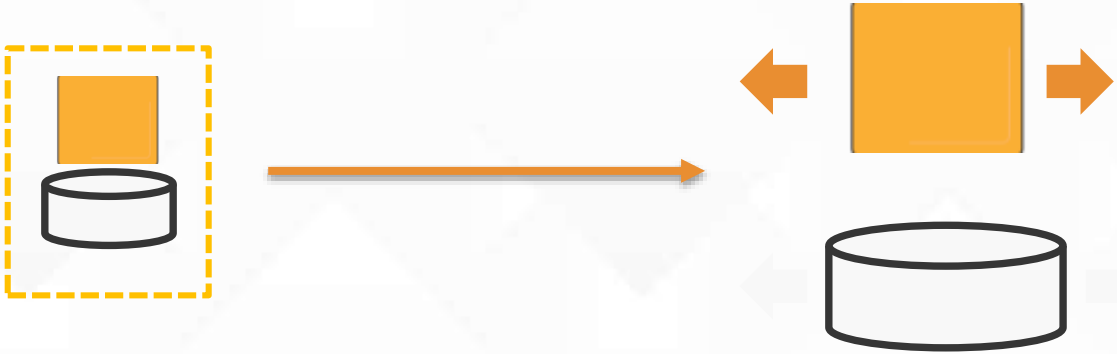

# Architecture of Amazon EMR

amazon web services



# Unlike on-prem or EC2-fleet Hadoop,

# EMR Decouples Storage & Compute



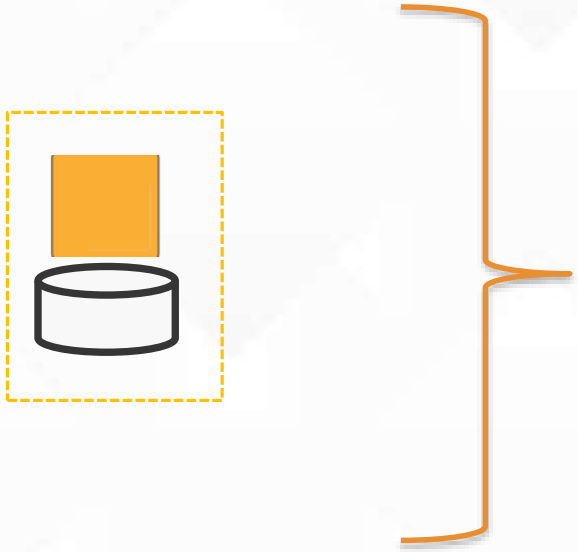
**Traditional Hadoop**

Tightly-coupled compute & storage
➔ inflexibility

**Amazon EMR**

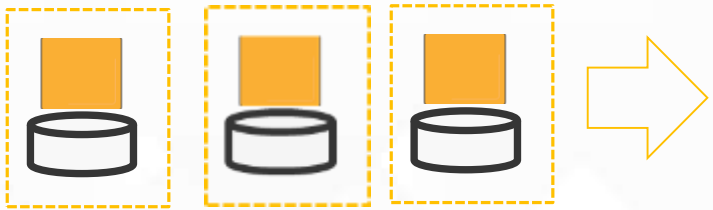
Decoupled compute & storage
➔ flexible storage
➔ Right-sizing compute

# On an On-premises Environment



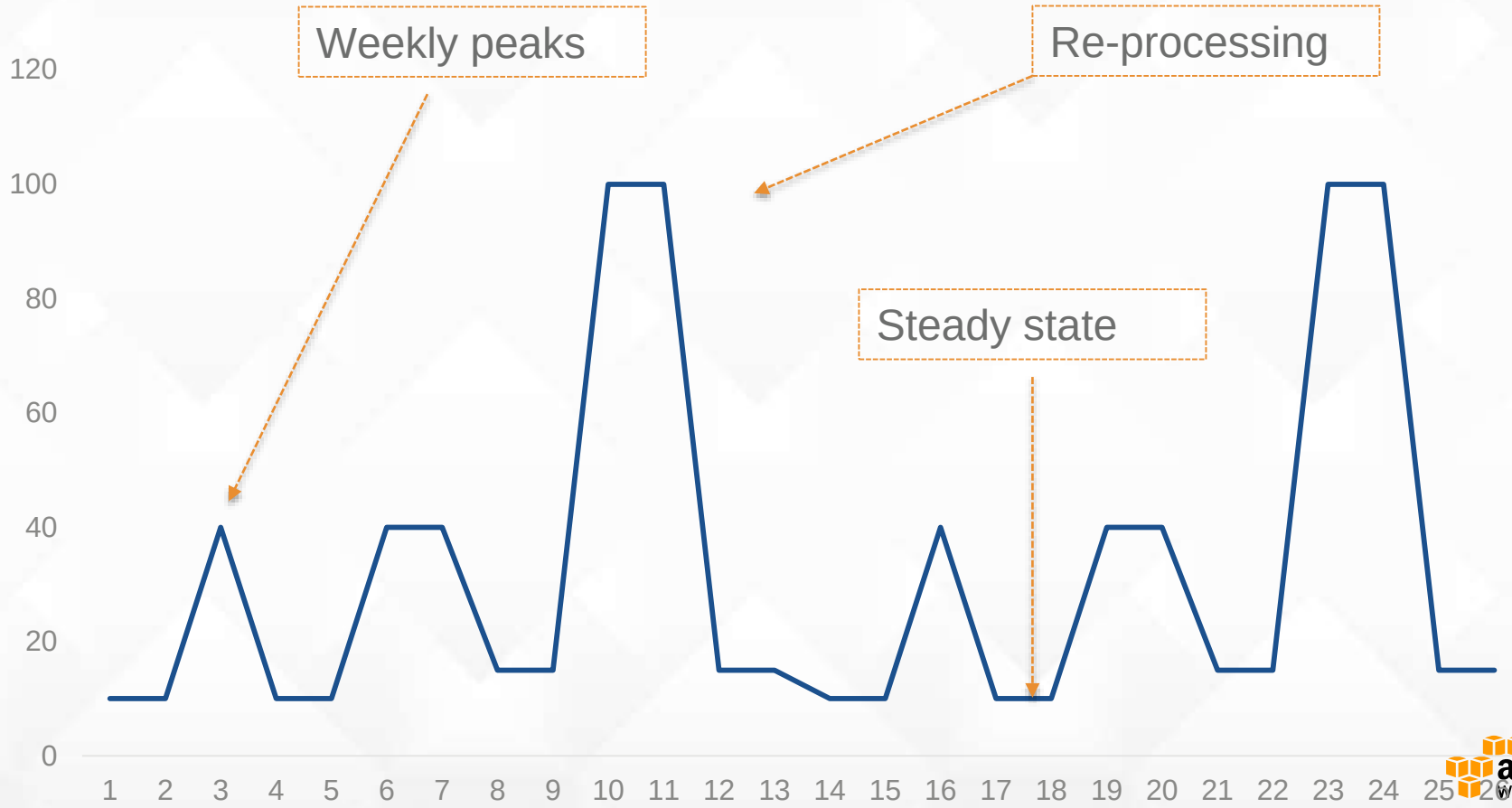
Tightly coupled

# Compute and Storage Grow Together

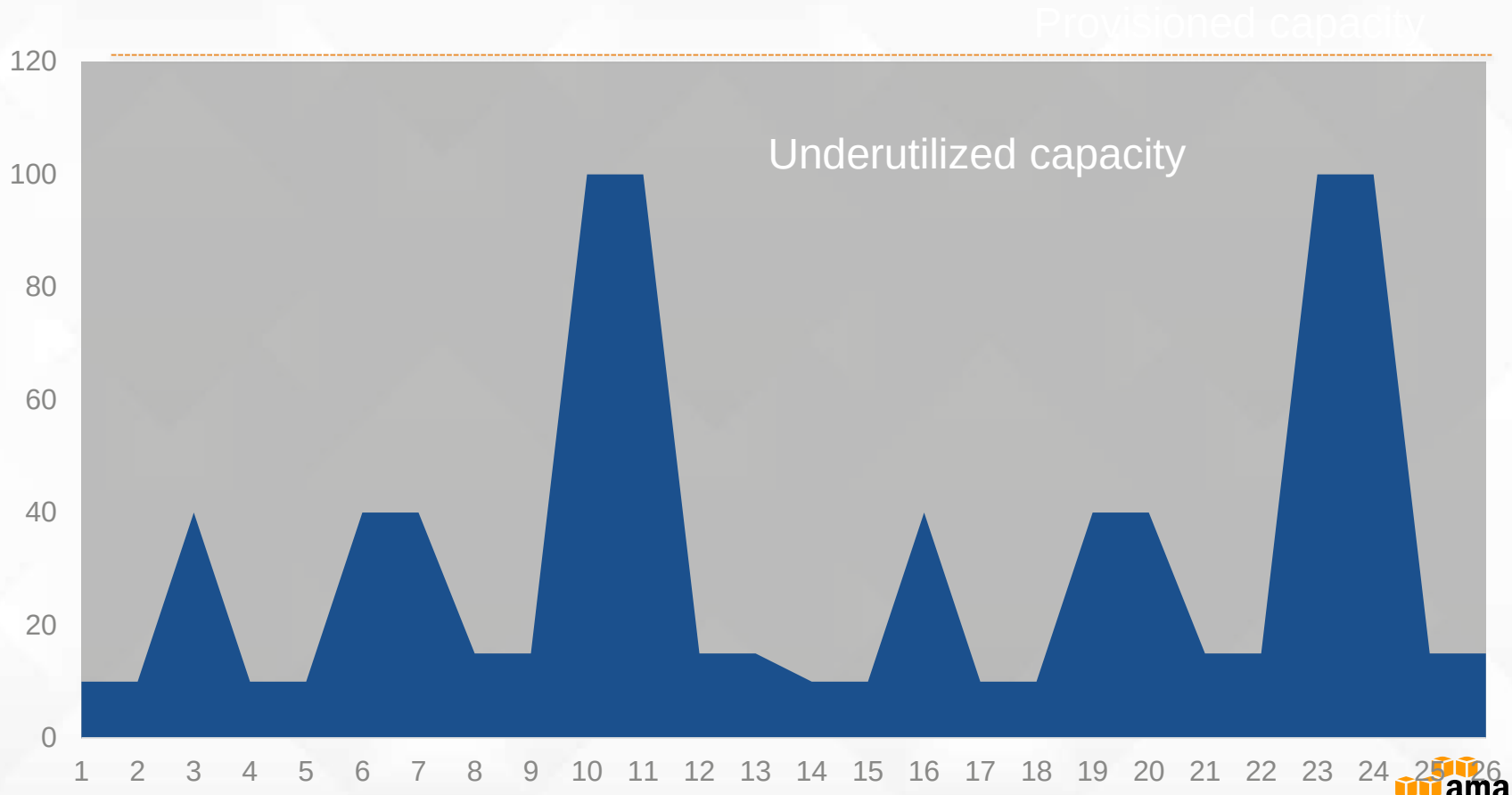


- Storage grows along with compute
- Compute requirements vary

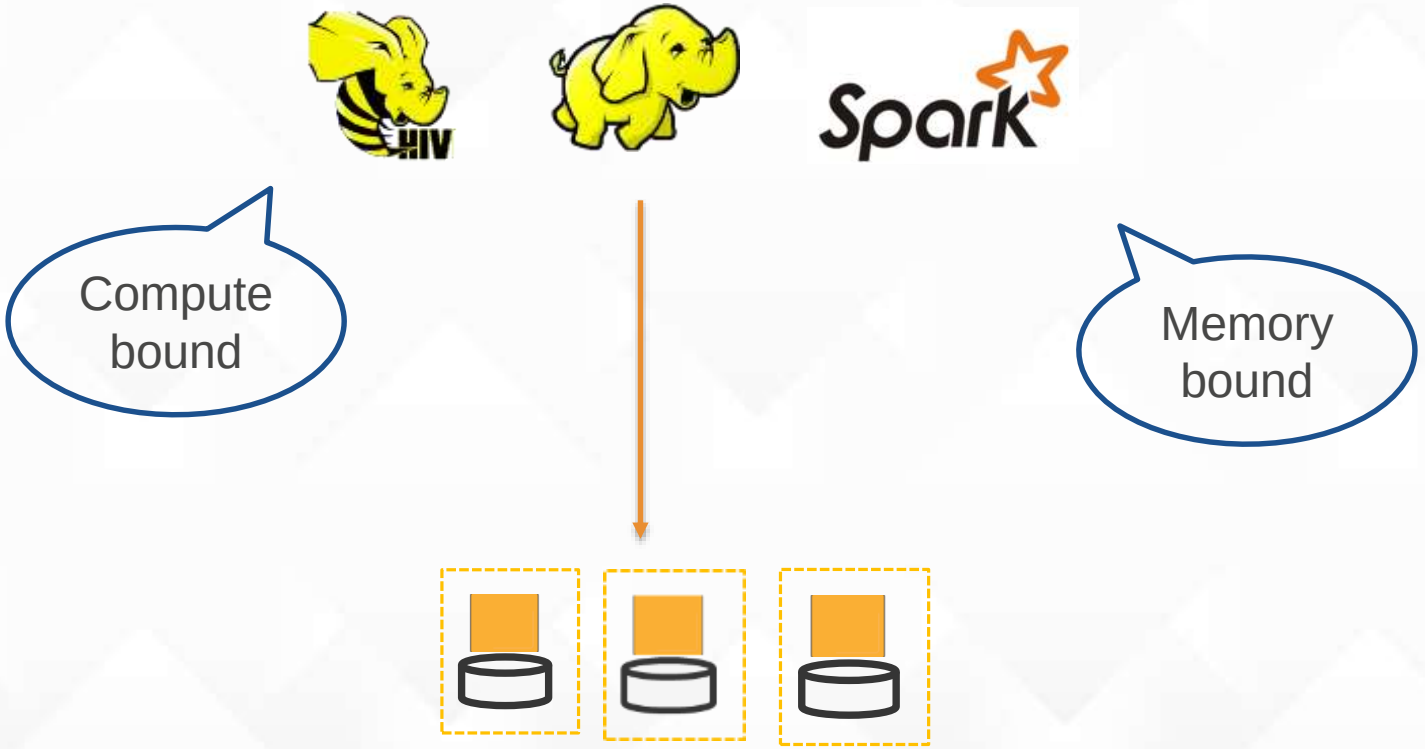
# Underutilized or Scarce Resources



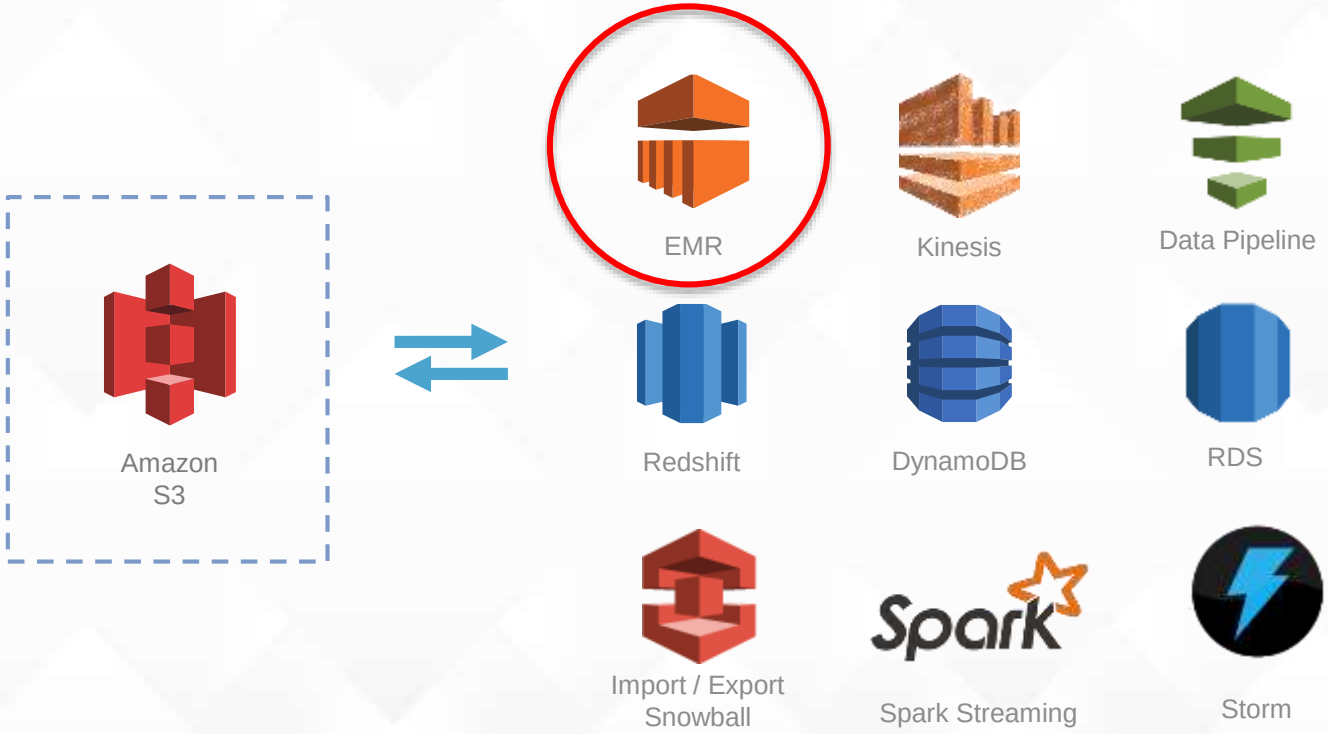
# Underutilized or Scarce Resources



# Contention for Same Resources

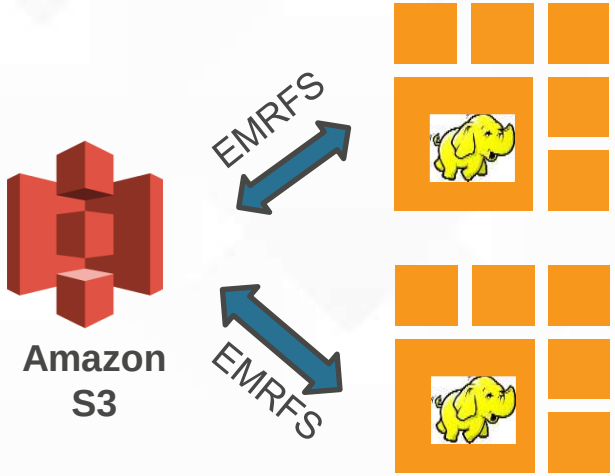


# EMR: Aggregate all Data in S3 as your *Data Lake* Surrounded by a collection of the right tools



# EMRFS: Amazon S3 as your persistent data store

- Decouple compute & storage
- Shut down EMR clusters with no data loss
- Right-size EMR cluster independently of storage
- Multiple Amazon EMR clusters can concurrently use same data in Amazon S3



# HDFS is still there if you need it

- Iterative workloads
  - If you're processing the same dataset more than once
  - Consider using Spark & RDDs for this too
- Disk I/O intensive workloads
- Persist data on Amazon S3 and use S3DistCp to copy to/from HDFS for processing

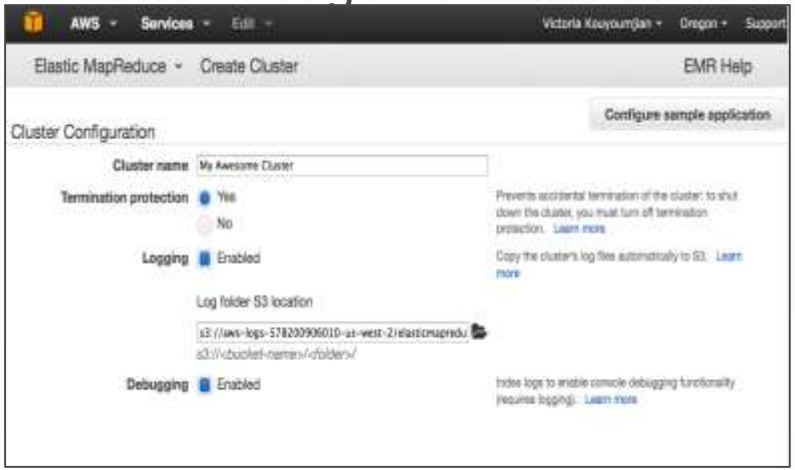
# Provisioning clusters

# What Do I Need to Build a Cluster ?

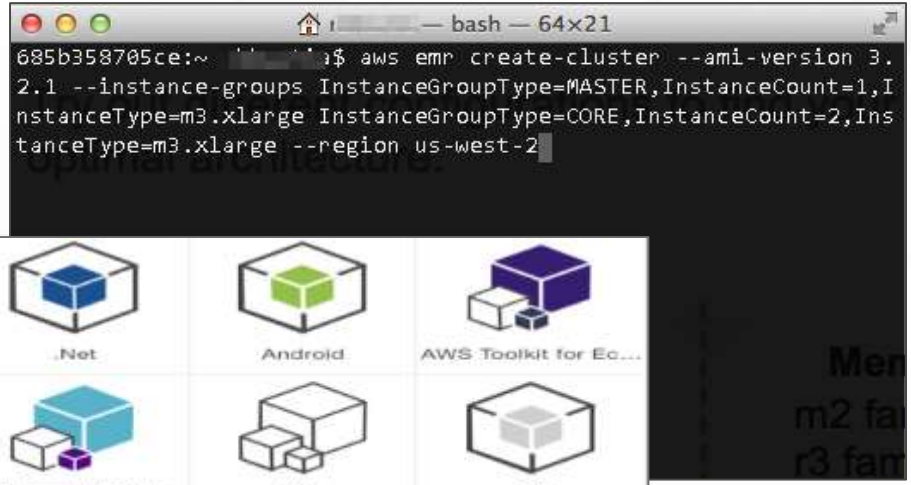
1. Choose instances
2. Choose your software
3. Choose your access method

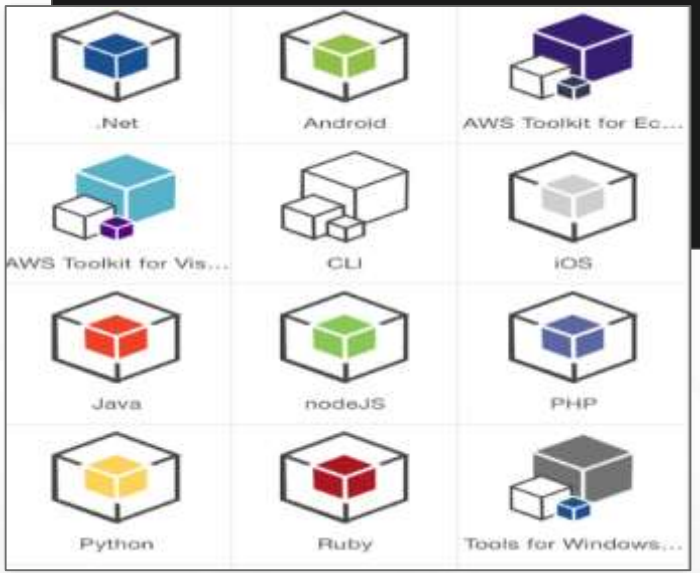
# EMR is Easy to Deploy

## AWS Management Console




## Command Line




or use the EMR API with your favorite SDK

amazon web services

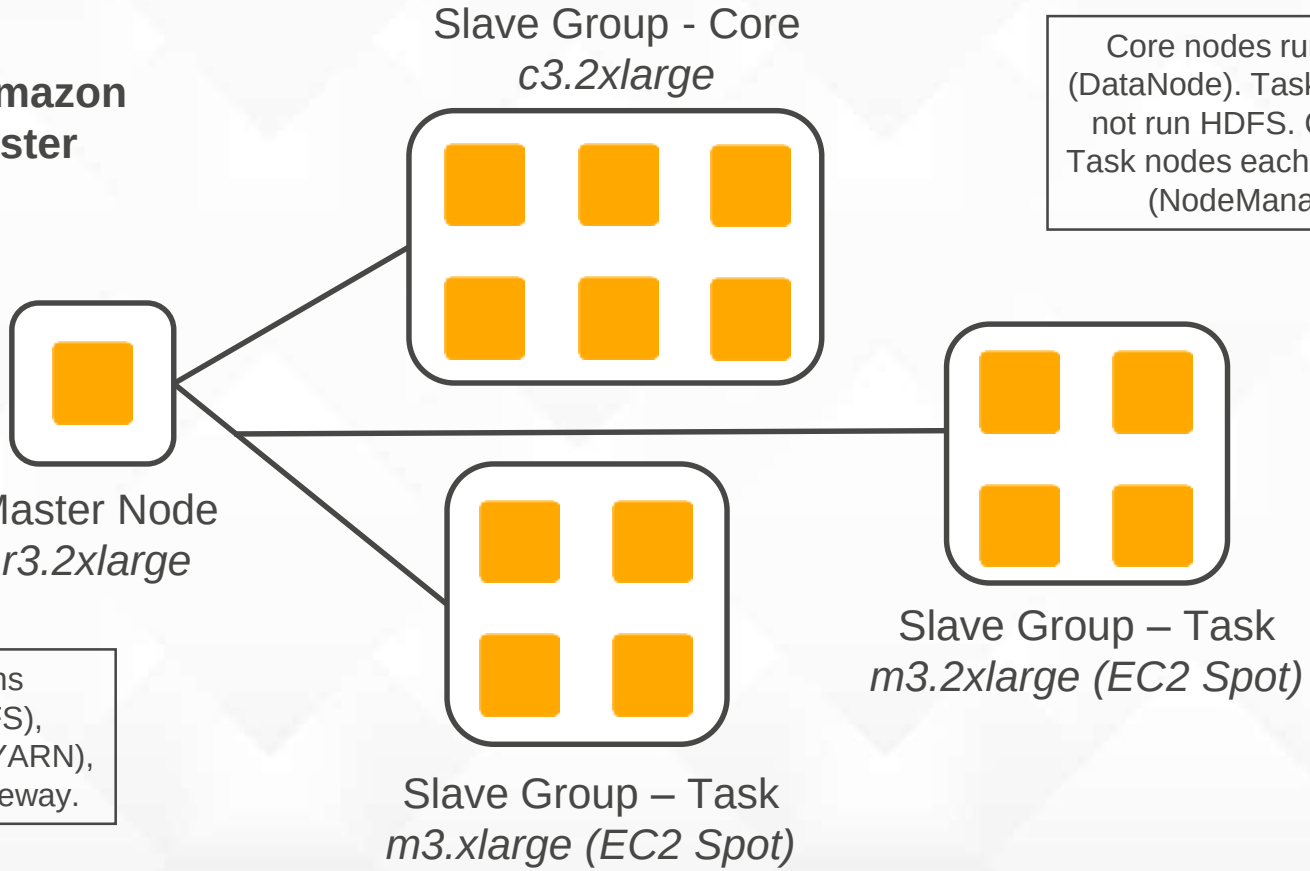
# Choose your instance types

Try different configurations to find your optimal architecture

| General | CPU | Memory | Disk/IO |
| --- | --- | --- | --- |
| m1 family<br>m3 family | c3 family<br>cc1.4xlarge<br>cc2.8xlarge | m2 family<br>r3 family<br>m4 family | d2 family<br>i2 family |
| **Batch process** | **Machine learning** | **Spark and interactive** | **Large HDFS** |

# Use multiple EMR instance groups

**Example Amazon EMR Cluster**




Core nodes run HDFS (DataNode). Task nodes do not run HDFS. Core and Task nodes each run YARN (NodeManager).

Master node runs NameNode (HDFS), ResourceManager (YARN), and serves as a gateway.

amazon web services

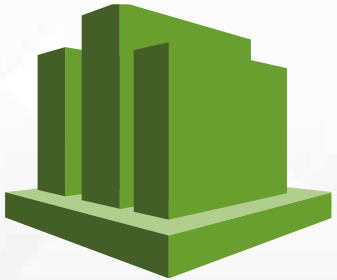
# EMR 5.0 - Applications

- [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

# Easy to monitor and debug

## Monitor

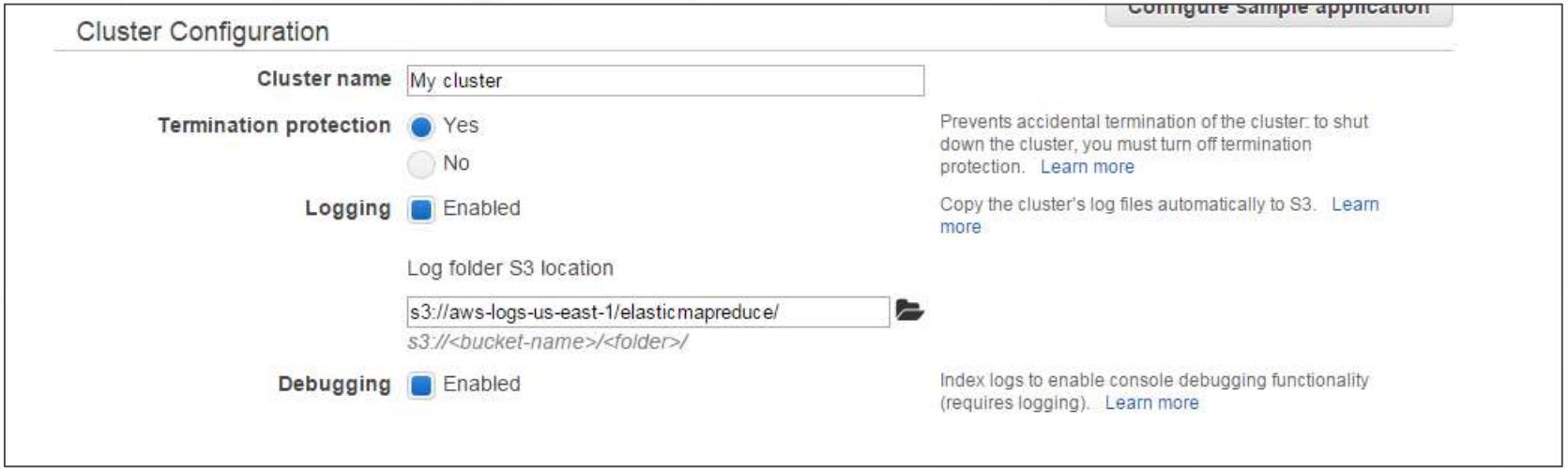

## Debug

Integrated with Amazon CloudWatch
Monitor cluster, node, and I/O, and 20+ custom Hadoop metrics
Task level debugging information already pushed in a datastore

amazon web services

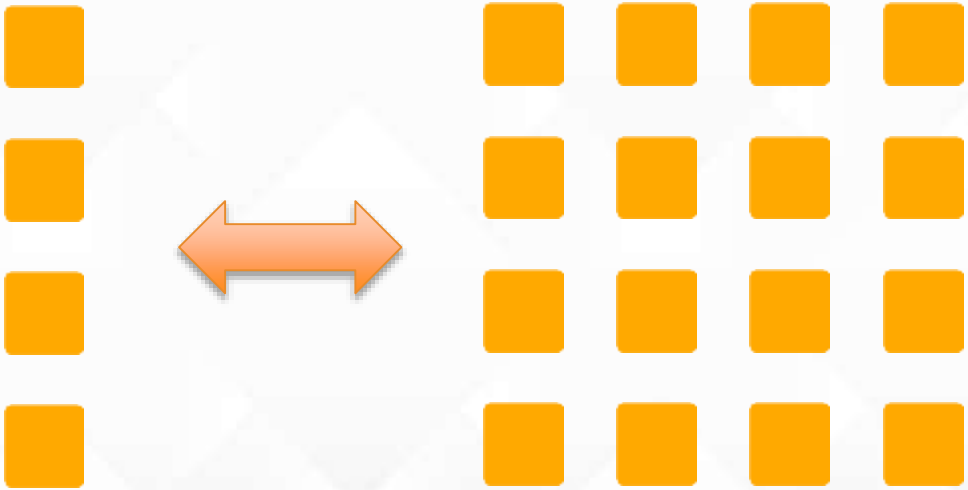
# EMR logging to S3 makes logs easily available




Logs in S3. Can use ELK to visualize

# Resizable clusters

Easy to add and remove compute capacity on your cluster.



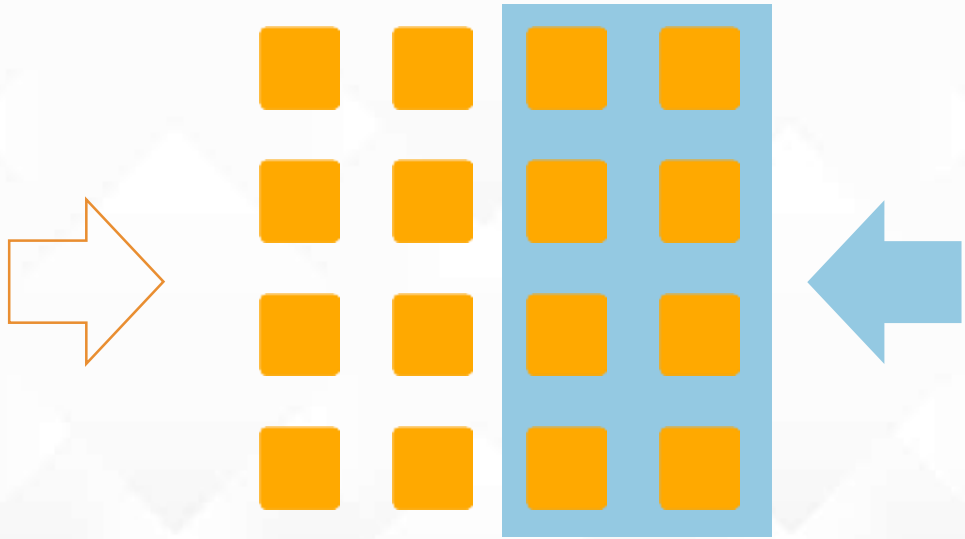
Match compute demands with cluster sizing.

# Easy to use Spot Instances

**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 90% off EC2 on-demand pricing

amazon web services

# The spot advantage

- Lets say a 7 hour job needing 100 nodes with each node at $1 = 7 x 100 x $1 = $700
- Scale up to 200 nodes
  - 4 * 100 * $1 = $400
  - 4 * 100 * $0.50 = $200
  - Save $100 dollars and finish the job fast
- Run on-demand for worst acceptable SLA
  - Scale up to meet demand with Spot instances

## Spot Bid Advisor

Region: US West (Northern California) | OS: Linux/UNIX | Bid Price: 50% On-Demand

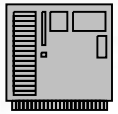
**Instance type filter:**

vCPU (min): 1 | Memory GiB (min): 0 | Instance types supported by EMR

| Instance Type | vCPU | Memory GiB | Savings over On-Demand* | Frequency of being outbid (month) ▾ | Frequency of being outbid (week) |
|---|---|---|---|---|---|
| m3.xlarge | 4 | 15 | 89% | Low | Low |
| m1.small | 1 | 1.7 | 83% | Low | Low |
| m1.medium | 1 | 3.75 | 90% | Low | Low |
| m1.large | 2 | 7.5 | 91% | Low | Low |
| c3.xlarge | 4 | 7.5 | 81% | Low | Low |
| c3.4xlarge | 16 | 30 | 80% | Low | Low |
| c1.xlarge | 8 | 7 | 89% | Low | Low |
| c3.2xlarge | 8 | 15 | 82% | Medium | Medium |
| m1.xlarge | 4 | 15 | 90% | Medium | Low |
| c3.8xlarge | 32 | 60 | 83% | Medium | High |

# Options to submit jobs – Off Cluster

Submit a Spark application



Amazon EMR Step API

Use AWS Lambda to submit applications to EMR Step API or directly to Spark on your cluster

AWS Lambda

AWS Data Pipeline

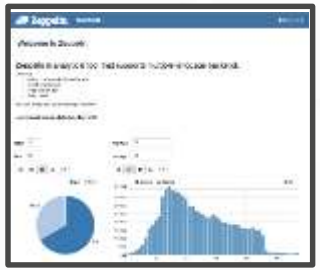
Create a pipeline to schedule job submission or create complex workflows

Airflow, Luigi, or other schedulers on EC2

**Amazon EMR**

amazon web services

# Options to submit jobs – On Cluster



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

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

*(start using start-thriftserver.sh)*

Connect with ODBC / JDBC using HiveServer2/Spark Thriftserver

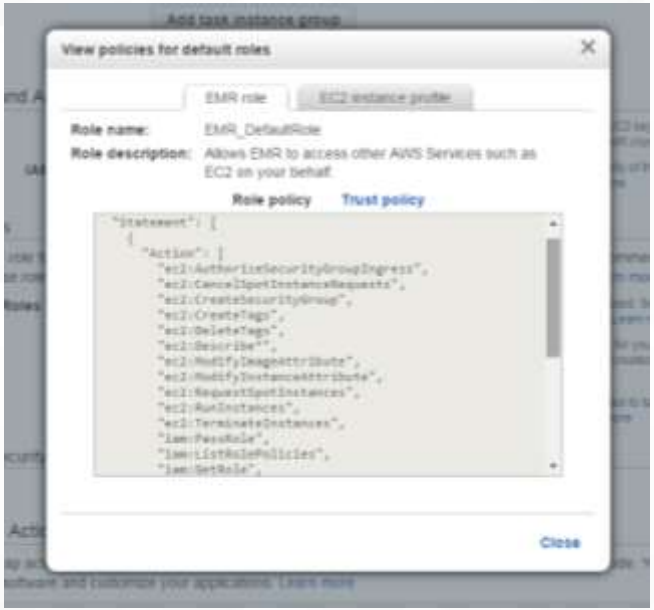
Or, use the native APIs and CLIs for each application

amazon web services

# EMR Security

# EMRFS client-side encryption

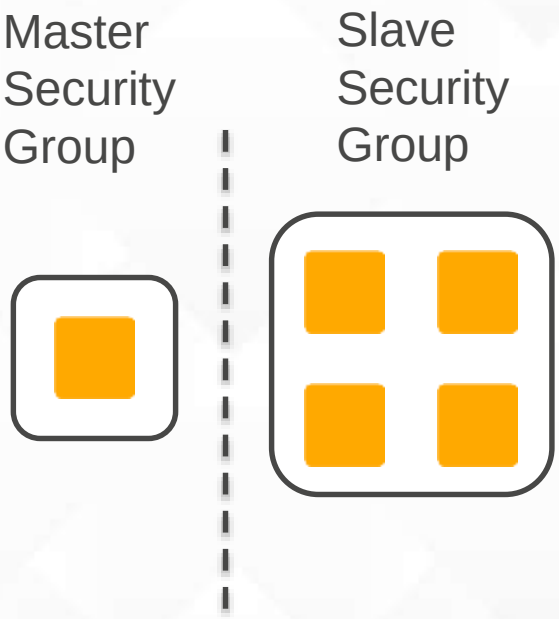
# Use Identity and Access Management (IAM) roles with your Amazon EMR cluster




- IAM roles give AWS services fine grained control over delegating permissions to AWS services and access to AWS resources
- EMR uses two IAM roles:
  - EMR service role is for the Amazon EMR control plane
  - EC2 instance profile is for the actual instances in the Amazon EMR cluster
- Default IAM roles can be easily created and used from the AWS Console and AWS CLI

amazon web services

# EMR Security Groups: default and custom



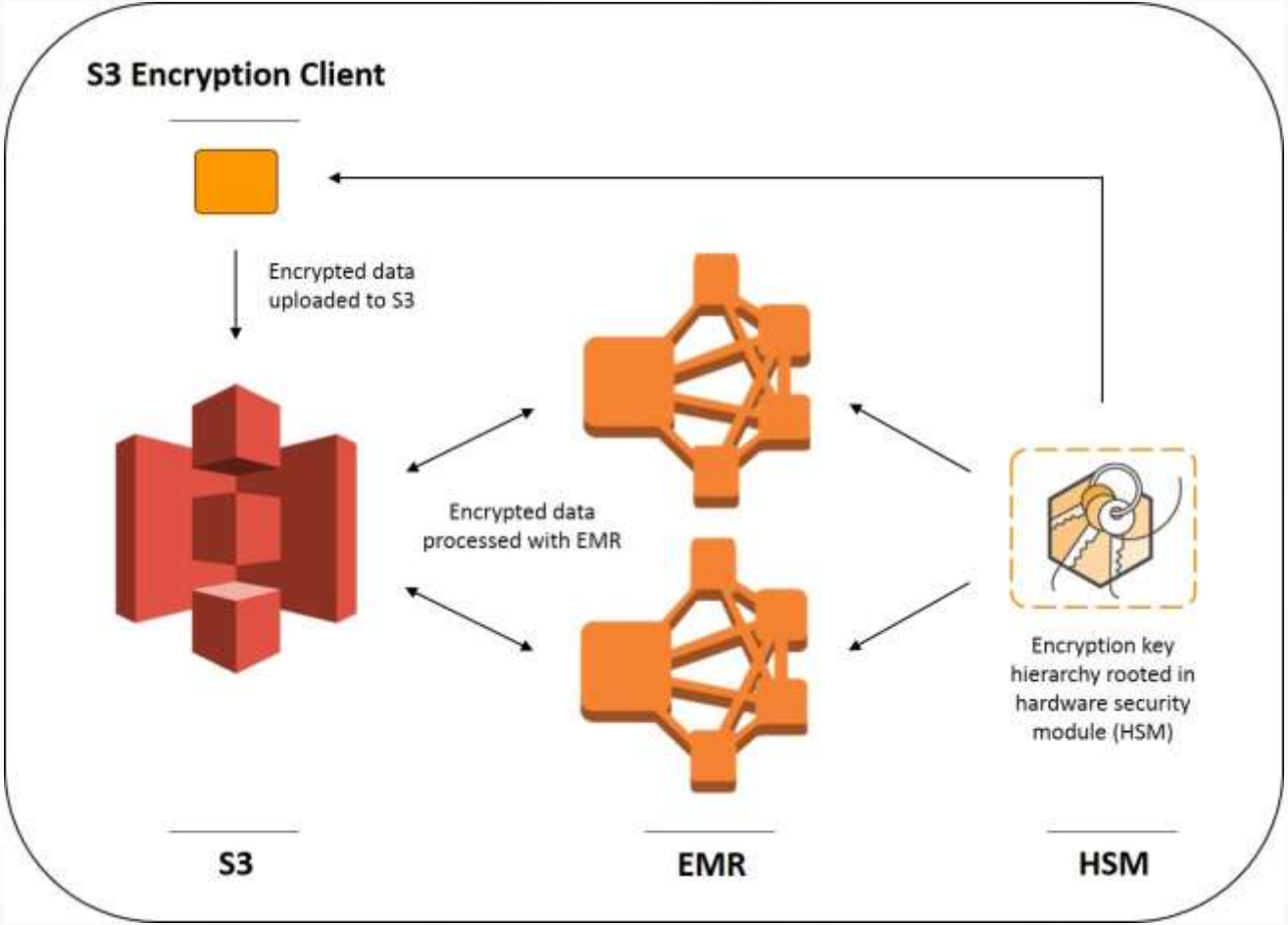

- A security group is a virtual firewall which controls access to the EC2 instances in your Amazon EMR cluster
  - There is a single default master and default slave security group across all of your clusters
  - The master security group has port 22 access for SSHing to your cluster
- You can add additional security groups to the master and slave groups on a cluster to separate them from the default master and slave security groups, and further limit ingress and egress policies.

# Encryption

- EMRFS encryption options
  - S3 server-side encryption
  - S3 client-side encryption (use AWS Key Management Service keys or custom keys)
- CloudTrail integration
  - Track Amazon EMR API calls for auditing
- Launch your Amazon EMR clusters in a VPC
  - Logically isolated portion of the cloud (“Virtual Private Network”)
  - Enhanced networking on certain instance types

# NASDAQ

- Encrypt data in the cloud, but the keys need to be on on-premises
- SafeNet Luna SA – KMS
- EMRFS
  - Utilizes S3 encryption clients envelope encryption and provides a hook
  - Custom Encryption Materials providers

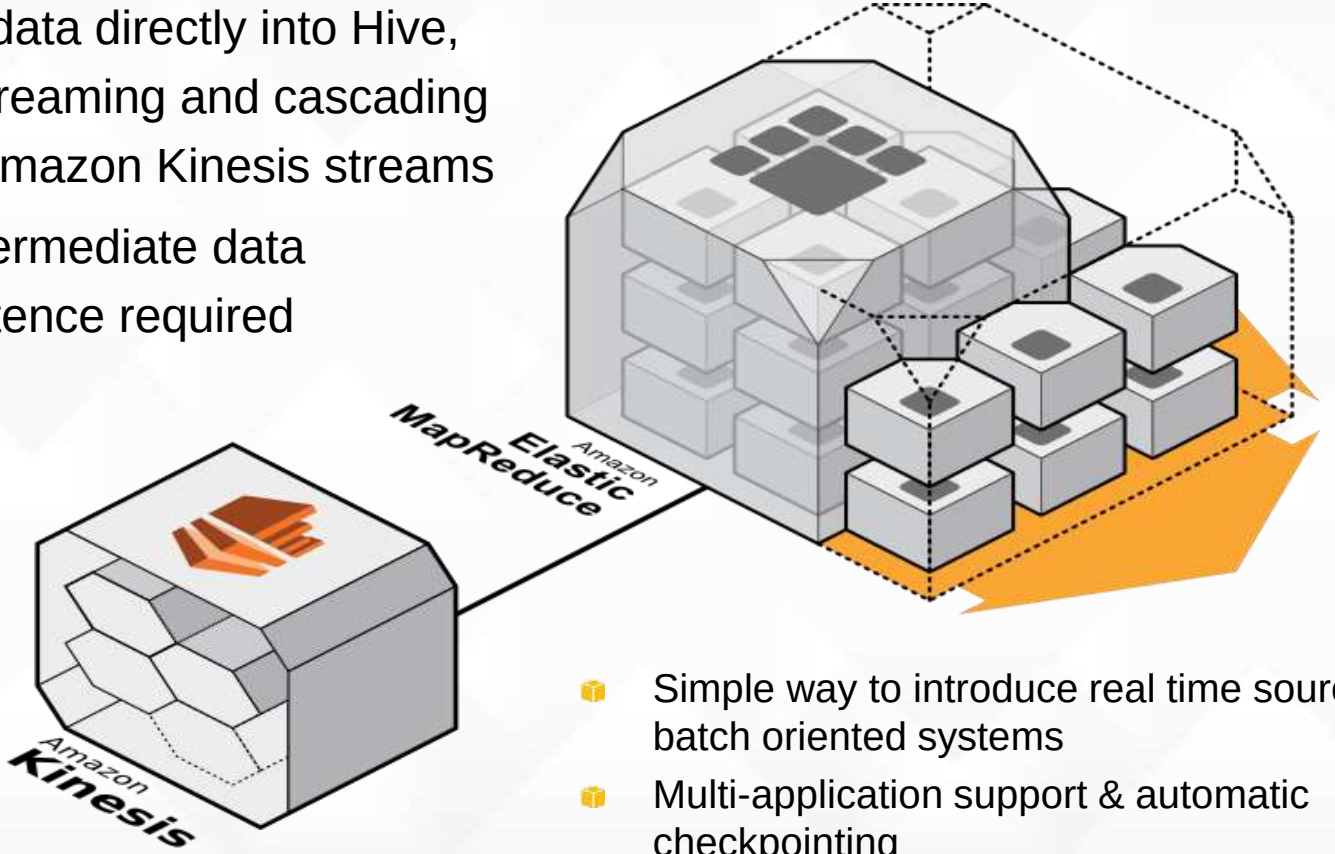


# NASDAQ

- Write your custom encryption materials provider
- -emrfs Encryption=ClientSide,ProviderType=Custom,CustomProviderLocation=s3://mybucket/myfolder/myprovider.jar,CustomProviderClass=providerclassname
- EMR pulls and installs the JAR on every node
- More info @ “NASDAQ AWS Big Data blog”
- Code samples

# Amazon EMR integration with Amazon Kinesis

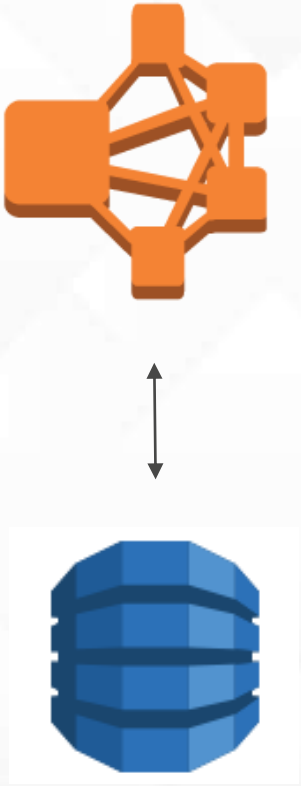
- Read data directly into Hive, Pig, streaming and cascading from Amazon Kinesis streams
- No intermediate data persistence required




- Simple way to introduce real time sources into batch oriented systems
- Multi-application support & automatic checkpointing

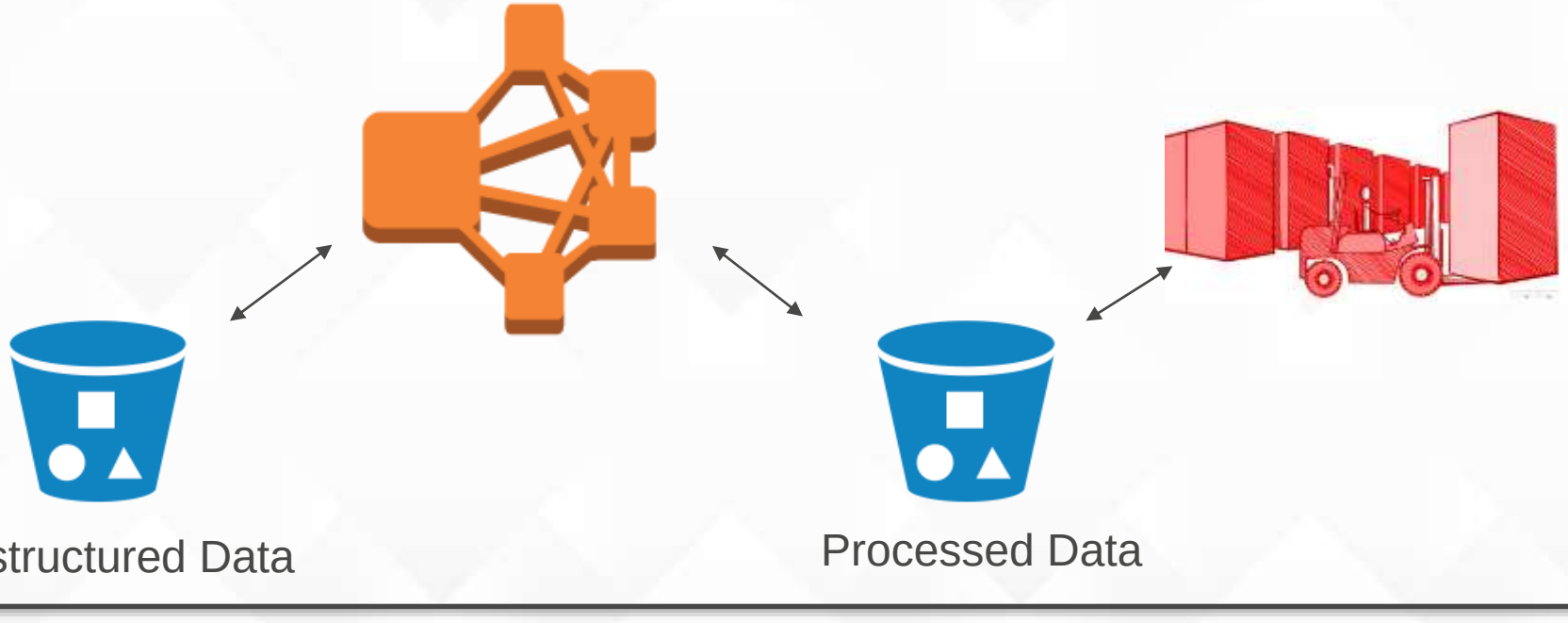
amazon web services

# Use Hive with EMR to query data DynamoDB



- Export data stored in DynamoDB to Amazon S3
- Import data in Amazon S3 to DynamoDB
- Query live DynamoDB data using SQL-like statements (HiveQL)
- Join data stored in DynamoDB and export it or query against the joined data
- Load DynamoDB data into HDFS and use it in your EMR job

# Use AWS Data Pipeline and EMR to transform data and load into Amazon Redshift



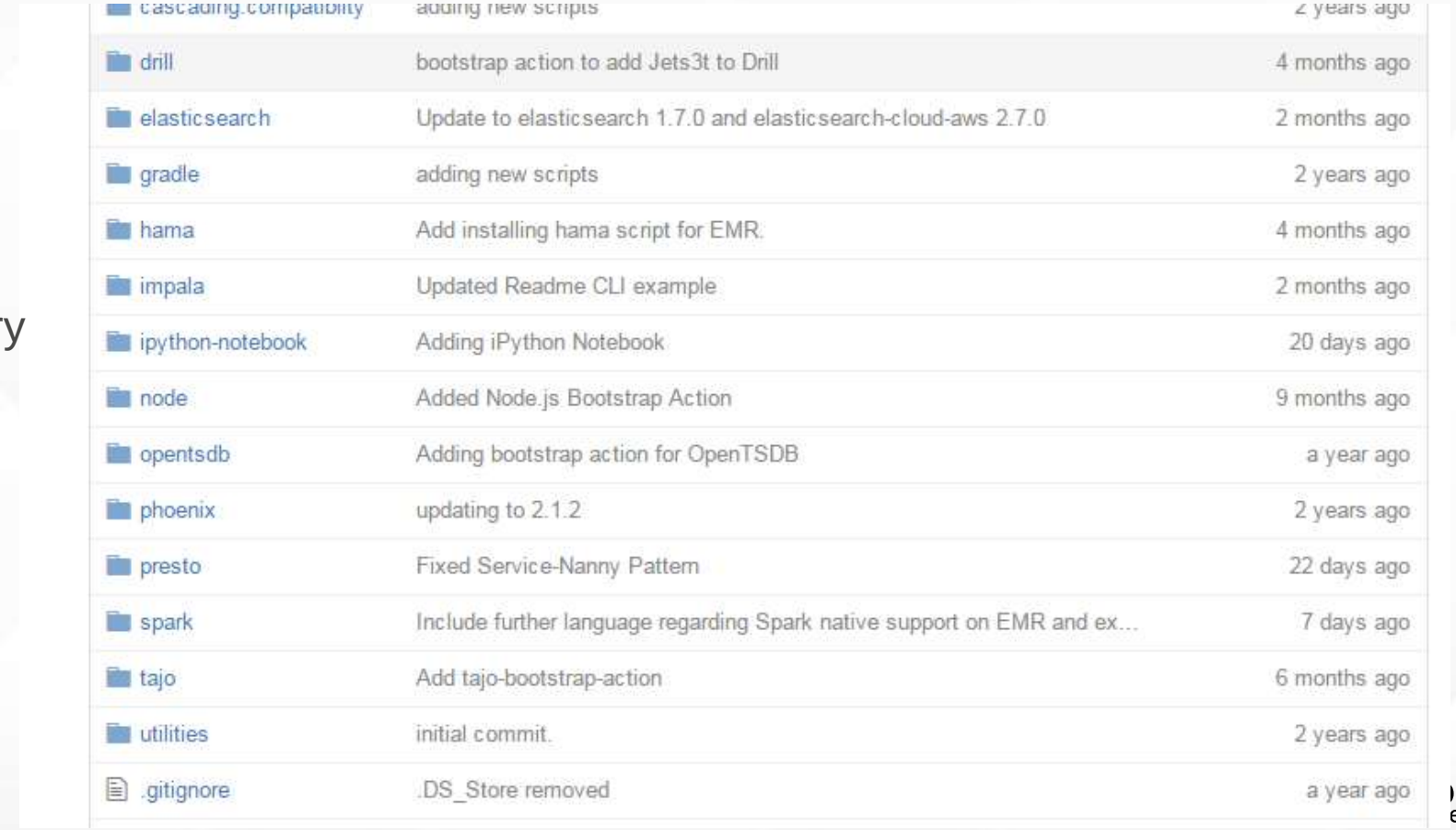

*Pipeline orchestrated and scheduled by AWS Data Pipeline*

# Install an iPython Notebook using Bootstrap

```
aws emr create-cluster --name iPythonNotebookEMR \ --ami-version 3.2.3 --instance-type
m3.xlarge --instance-count 3 \ --ec2-attributes KeyName=<<MYKEY>>> \ --use-default-roles \ --
bootstrap-actions Path=s3://elasticmapreduce.bootstrapactions/ipython-notebook/install-ipython-
notebook,Name=Install_iPython_NB \ --termination-protected
```

# Install Hadoop Applications with Bootstrap Actions

Github Repository

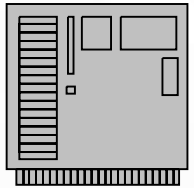


| | | |
|---|---|---|
| cascading.compatibility | adding new scripts | 2 years ago |
| drill | bootstrap action to add Jets3t to Drill | 4 months ago |
| elasticsearch | Update to elasticsearch 1.7.0 and elasticsearch-cloud-aws 2.7.0 | 2 months ago |
| gradle | adding new scripts | 2 years ago |
| hama | Add installing hama script for EMR. | 4 months ago |
| impala | Updated Readme CLI example | 2 months ago |
| ipython-notebook | Adding iPython Notebook | 20 days ago |
| node | Added Node.js Bootstrap Action | 9 months ago |
| opentsdb | Adding bootstrap action for OpenTSDB | a year ago |
| phoenix | updating to 2.1.2 | 2 years ago |
| presto | Fixed Service-Nanny Pattern | 22 days ago |
| spark | Include further language regarding Spark native support on EMR and ex... | 7 days ago |
| tajo | Add tajo-bootstrap-action | 6 months ago |
| utilities | initial commit. | 2 years ago |
| .gitignore | .DS_Store removed | a year ago |

# Amazon EMR – Design Patterns

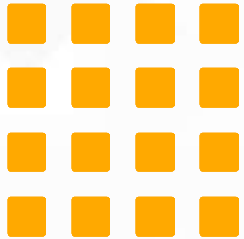
amazon web services

# EMR example #1: Batch Processing



GB of logs pushed to S3 hourly

Input and output stored in S3

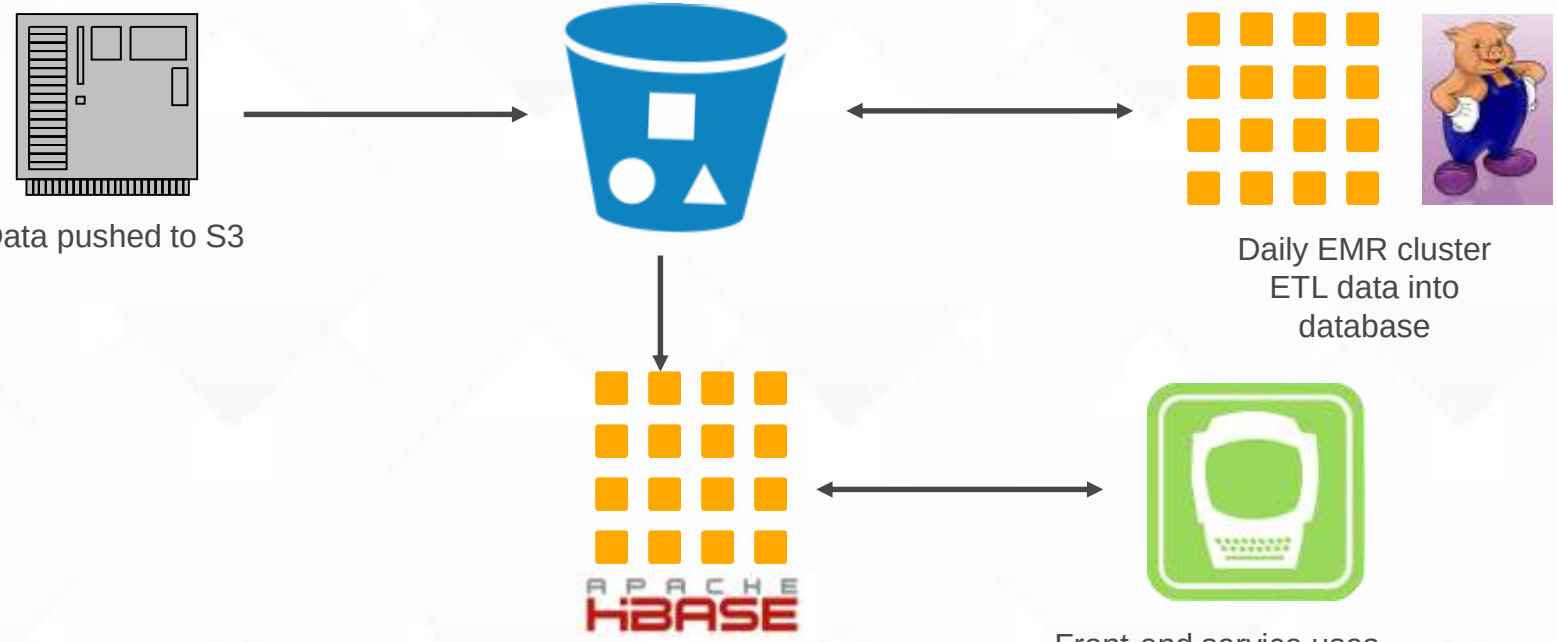



Daily EMR cluster using Hive to process data

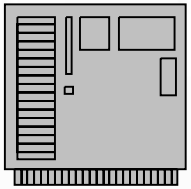

250 Amazon EMR jobs per day, processing 30 TB of data

http://aws.amazon.com/solutions/case-studies/yelp/

# EMR example #2: Long-running cluster

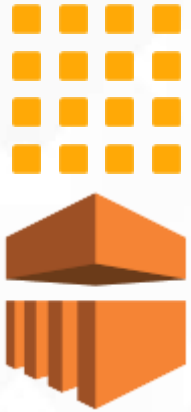


# EMR example #3: Interactive query



TBs of logs sent daily

Logs stored in Amazon S3



Hive Metastore on Amazon EMR

cloudera IMPALA

Spark

presto

HIVE TEZ

NETFLIX

Interactive query using Presto on multi-petabyte warehouse

http://nflx.it/1dO7Pnt

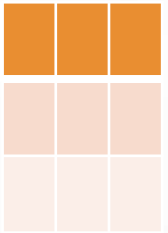
amazon web services

# Why Spark? Functional Programming Basics

```
for (int i = 0, i <= n, i++) {
   if (s[i].contains(“ERROR”) {
      messages[i] = split(s[i], ‘\t’)[2]
   }
}
```

Sequential processing

```
messages = textFile(...).filter(lambda s: s.contains(“ERROR”))
                        .map(lambda s: s.split(‘\t’)[2])
```

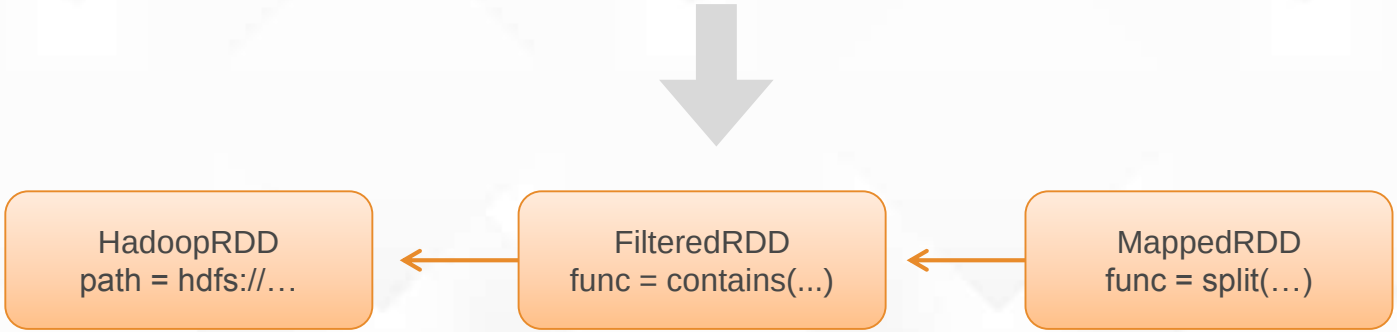


Easy to parallel

# RDDs (and now DataFrames) and Fault Tolerance

- RDDs track the transformations used to build them (their lineage) to recompute lost data
- E.g.:

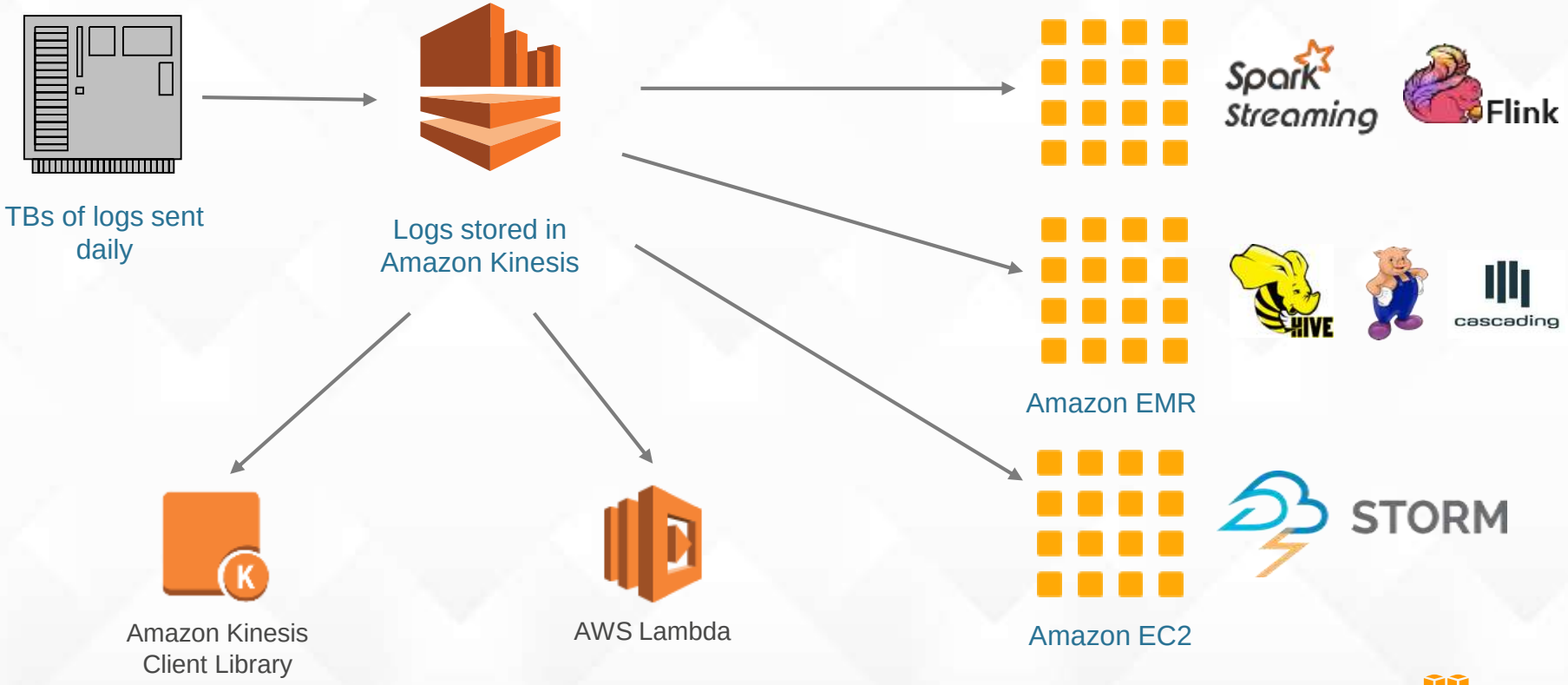
```
messages = textFile(...).filter(lambda s: s.contains(“ERROR”))
                        .map(lambda s: s.split(‘\t’)[2])
```



# Why Presto? SQL on Unstructured and Unlimited Data

- Dynamic Catalog
- Rich ANSI SQL
- Connectors as Plugins
- High Concurrency
- Batch (ETL)/Interactive Clusters

# EMR example #4: Streaming data processing

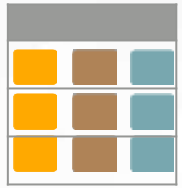


# EMR Best Practices

amazon web services

# File formats

- Row oriented
  - Text files
  - Sequence files
    - Writable object
  - Avro data files
    - Described by schema
- Columnar format
  - Object Record Columnar (ORC)
  - Parquet



Logical table

Row oriented

Column oriented

# S3 File Format:  Parquet

- Parquet file format:  http://parquet.apache.org
- Self-describing columnar file format
- Supports nested structures (Dremel “record shredding” algo)
- Emerging standard data format for Hadoop
  - Supported by:  Presto, Spark, Drill, Hive, Impala, etc.

# Parquet vs ORC

- Evaluated Parquet and ORC (competing open columnar formats)

- ORC encrypted performance is currently a problem
  - 15x slower vs. unencrypted (94% slower)
  - 8 CPUs on 2 nodes: ~900 MB/sec vs. ~60 MB/sec encrypted
  - Encrypted Parquet is ~27% slower vs. unencrypted
  - Parquet: ~100MB/sec from S3 per CPU core (encrypted)

# Factors to consider

- Processing and query tools
  - Hive, Impala and Presto
- Evolution of schema
  - Avro for schema and Presto for storage
- File format “splittability”
  - Avoid JSON/XML Files. Use them as records

# File sizes

- Avoid small files
  - Anything smaller than 100MB
- Each mapper is a single JVM
  - CPU time is required to spawn JVMs/mappers
- Fewer files, matching closely to block size
  - fewer calls to S3
  - fewer network/HDFS requests

# Dealing with small files

- Reduce HDFS block size, e.g. 1MB (default is 128MB)

```
--bootstrap-action s3://elasticmapreduce/bootstrap-
actions/configure-hadoop --args "-
m,dfs.block.size=1048576"
```

- Better: Use `S3DistCp` to combine smaller files together
  - `S3DistCp` takes a pattern and target path to combine smaller input files to larger ones
  - Supply a target size and compression codec

# Compression

- Always compress data files On Amazon S3
  - Reduces network traffic between Amazon S3 and Amazon EMR
  - Speeds Up Your Job
- Compress mappers and reducer output
- Amazon EMR compresses inter-node traffic with LZO with Hadoop 1, and Snappy with Hadoop 2

# Choosing the right compression

- Time sensitive, faster compressions are a better choice
- Large amount of data, use space efficient compressions
- Combined Workload, use gzip

| Algorithm | Splittable? | Compression ratio | Compress + decompress speed |
|---|---|---|---|
| Gzip (DEFLATE) | No | High | Medium |
| bzip2 | Yes | Very high | Slow |
| LZO | Yes | Low | Fast |
| Snappy | No | Low | Very fast |

# Cost saving tips for Amazon EMR

- Use S3 as your persistent data store – query it using Presto, Hive, Spark, etc.
- Only pay for compute when you need it
- Use Amazon EC2 Spot instances to save >80%
- Use Amazon EC2 Reserved instances for steady workloads
- Use CloudWatch alerts to notify you if a cluster is underutilized, then shut it down. E.g. 0 mappers running for >N hours

# Holy Grail Question

What if I have small file issues?

# S3DistCP Options

- Most Important Options
- --src
- --srcPattern
- --dest
- --groupBy
- --outputCodec

| Option |
| --- |
| --src,LOCATION |
| --dest,LOCATION |
| --srcPattern,PATTERN |
| --groupBy,PATTERN |
| --targetSize,SIZE |
| --appendToLastFile |
| --outputCodec,CODEC |
| --s3ServerSideEncryption |
| --deleteOnSuccess |
| --disableMultipartUpload |
| --multipartUploadChunkSize,SIZE |
| --numberFiles |
| --startingIndex,INDEX |
| --outputManifest,FILENAME |
| --previousManifest,PATH |
| --requirePreviousManifest |
| --copyFromManifest |
| --s3Endpoint ENDPOINT |
| --storageClass CLASS |

# Persistent vs. Transient Clusters

# Persistent Clusters

- Very similar to traditional Hadoop deployments
- Cluster stays around after the job is done
- Data persistence model:
  - Amazon S3
  - Amazon S3 Copy To HDFS
  - HDFS, with Amazon S3 as backup


amazon web services

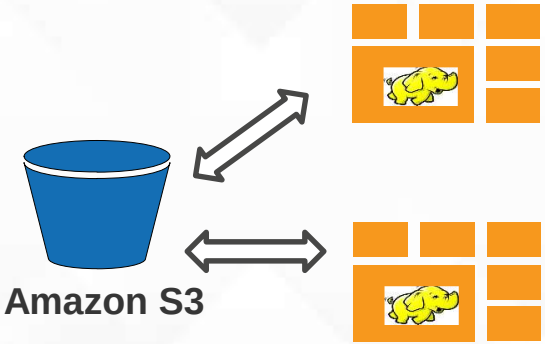
# Persistent Clusters

- Always keep data safe on Amazon S3 even if you're using HDFS for primary storage
- Get in the habit of shutting down your cluster and start a new one, once a week or month
  - Design your data processing workflow to account for failure
- You can use workflow managements such as AWS Data Pipeline

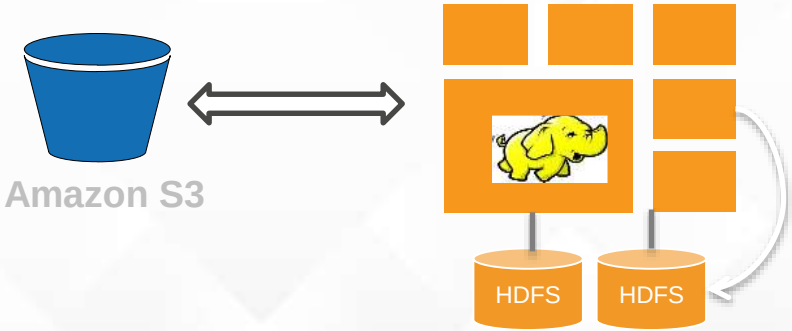
# Benefits of Persistent Clusters

- Ability to share data between multiple jobs
- Always On for Analyst Access

Transient cluster

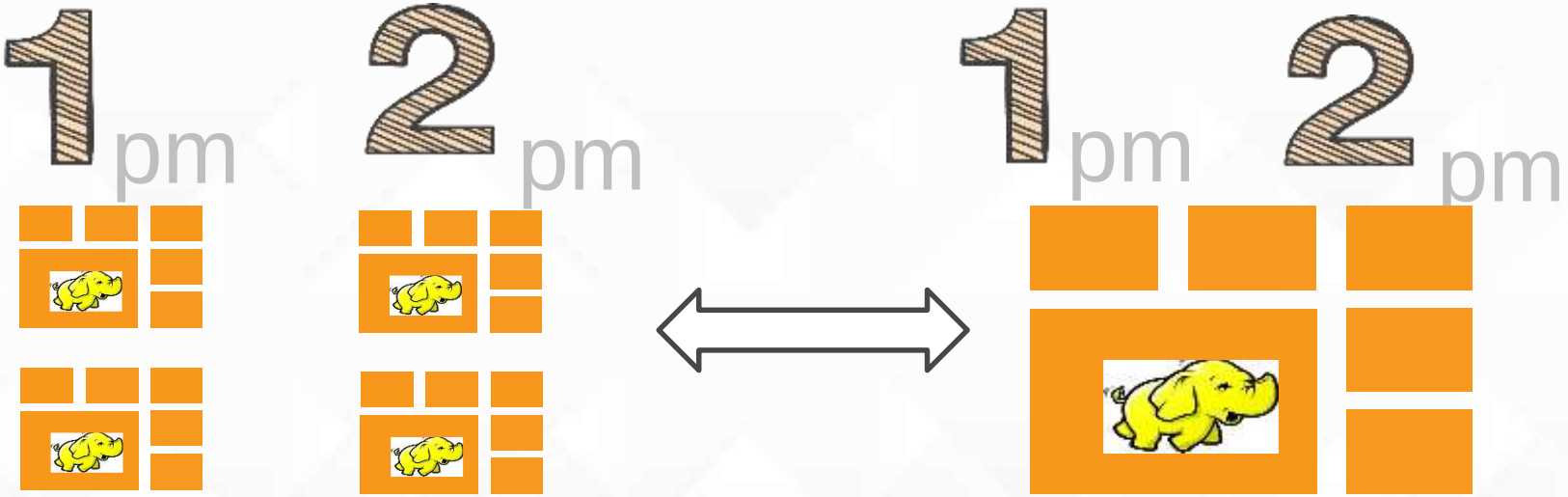



Long running clusters



# Benefit of Persistent Clusters

- Cost effective for repetitive jobs



# When to use Persistent clusters?

```
If   ( Data Load Time + Processing Time) x Number Of Jobs > 24

      Use Persistent Clusters

Else

      Use Transient Clusters
```

# When to use Persistent clusters?

*e.g.*

*( 20min data load + 1 hour Processing time ) x  20 jobs*

*= 26  hours*

*Is > 24 hour,  therefore use Persistent Clusters*

# Transient Clusters

- Cluster lives only for the duration of the job
  - Shut down the cluster when the job is done

- Data persisted on Amazon S3
  - Input & output

Data on Amazon S3

amazon web services

# Benefits of Transient Clusters

1. Control your cost
2. Minimum maintenance
   - Cluster goes away when job is done
3. Best Flexibility of Tools
4. Practice cloud architecture
   - Pay for what you use
   - Data processing as a workflow

# When to use Transient cluster?

If ( Data Load Time + Processing Time) x Number Of Jobs < 24

Use Transient Clusters

Else

Use Persistent Clusters

# When to use Transient clusters?

*e.g.*

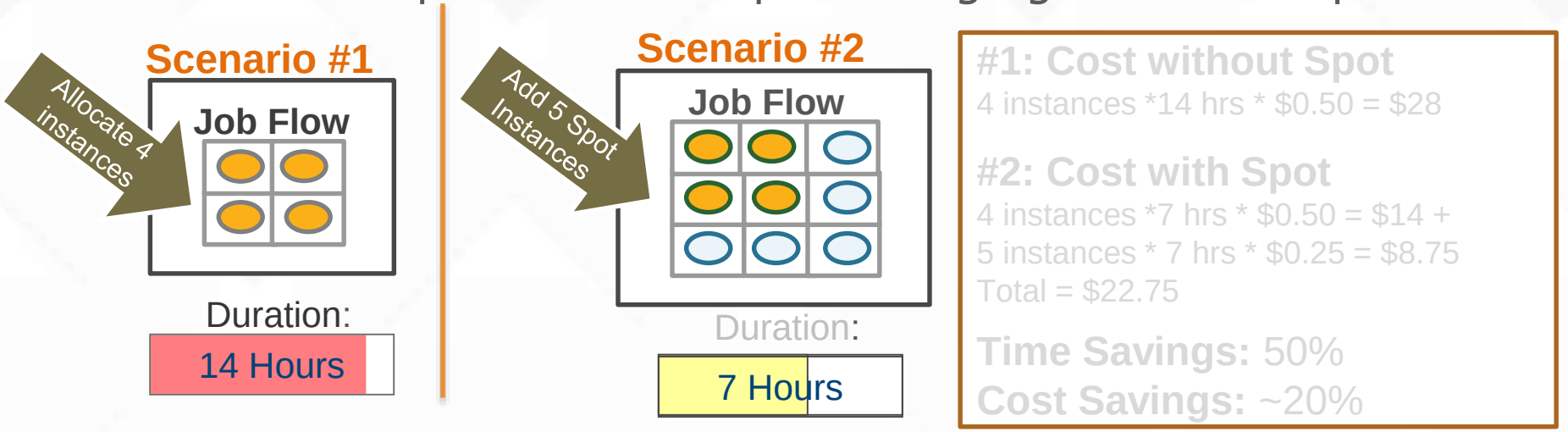
*( 20min data load + 1 hour Processing time ) x 10 jobs*

*= 13 hours*

*< 24 hour -> Use Transient Clusters*

# Reducing Costs with Spot Instances

Mix Spot and On-Demand instances to reduce cost and accelerate computation while protecting against interruption




Other EMR + Spot Use Cases

- Run entire cluster on Spot for biggest cost savings
- Reduce the cost of application testing

# Amazon EMR Nodes and Sizes

- Use m1 and c1 family for functional testing
- Use m3 and c3 xlarge and larger nodes for production workloads
- Use cc2/c3 for memory and CPU intensive jobs
- hs1, hi1, i2 instances for HDFS workloads
- Prefer a smaller cluster of larger nodes

amazon web services

# Holy Grail Question

How many nodes do I need?

# Cluster Sizing Calculation

1. Estimate the number of mappers your job requires.

# Cluster Sizing Calculation

2. Pick an EC2 instance and note down the number of mappers it can run in parallel

e.g. m1.xlarge = 8 mappers in parallel

# Cluster Sizing Calculation

3. We need to pick some sample data files to run a test workload. The number of sample files should be the same number from step #2.

# Cluster Sizing Calculation

4. Run an Amazon EMR cluster with a single core node and process your sample files from #3. Note down the amount of time taken to process your sample files.

# Cluster Sizing Calculation

$$\text{Estimated Number Of Nodes} = \frac{\textit{Total Mappers} * \textit{Time To Process Sample Files}}{\textit{Instance Mapper Capacity} * \textit{Desired Processing Time}}$$

# Example: Cluster Sizing Calculation

1. Estimate the number of mappers your job requires

150

2. Pick an instance and note down the number of mappers it can run in parallel

m1.xlarge with 8 mapper capacity per instance

# Example: Cluster Sizing Calculation

3. We need to pick some sample data files to run a test workload. The number of sample files should be the same number from step #2.

8 files selected for our sample test

# Example: Cluster Sizing Calculation

4. Run an Amazon EMR cluster with a single core node and process your sample files from #3. Note down the amount of time taken to process your sample files.

3 min to process 8 files

# Example: Cluster Sizing Calculation

Estimated number of nodes =

$$\frac{\textit{Total Mappers For Your Job} * \textit{Time To Process Sample Files}}{\textit{Per Instance Mapper Capacity} * \textit{Desired Processing Time}}$$

$$\frac{150 * 3\ \text{min}}{8 * 5\ \text{min}} = 11\ \text{m1.xlarge}$$

# Thank You

amazon web services