

# Hadoop 3.X more examples

Big Data - 01/04/2020



# Hadoop 3 on AMAZON





# Hadoop 3 on AMAZON

The screenshot displays the AWS Management Console interface. At the top, there is a navigation bar with the AWS logo, 'Services', 'Resource Groups', and user information (Roberto, N. Virginia, Support). Below the navigation bar, the main content area is titled 'AWS services'. It includes a search bar with the placeholder text 'Find a service by name (for example, EC2, S3, Elastic Beanstalk)'. Underneath the search bar, there are sections for 'Recently visited services' and 'All services'. The 'All services' section is organized into a grid of categories, each with an icon and a list of services:

- Compute**: EC2, EC2 Container Service, Lightsail, Elastic Beanstalk, Lambda, Batch
- Storage**: S3, EFS, Glacier, Storage Gateway
- Database**: (partially visible)
- Developer Tools**: CodeCommit, CodeBuild, CodeDeploy, CodePipeline, X-Ray
- Management Tools**: CloudWatch, CloudFormation, CloudTrail, Config, OpsWorks, Service Catalog, Trusted Advisor
- Internet of Things**: AWS IoT
- Contact Center**: Amazon Connect
- Game Development**: Amazon GameLift
- Mobile Services**: Mobile Hub, Cognito, Device Farm, Mobile Analytics

On the right side of the console, there is a 'Featured next steps' section. It contains two recommendations:

- Manage your costs**: Get real-time billing alerts based on your cost and usage budgets. [Start now](#)
- Get best practices**: Use AWS Trusted Advisor for security, performance, cost and availability best practices. [Start now](#)

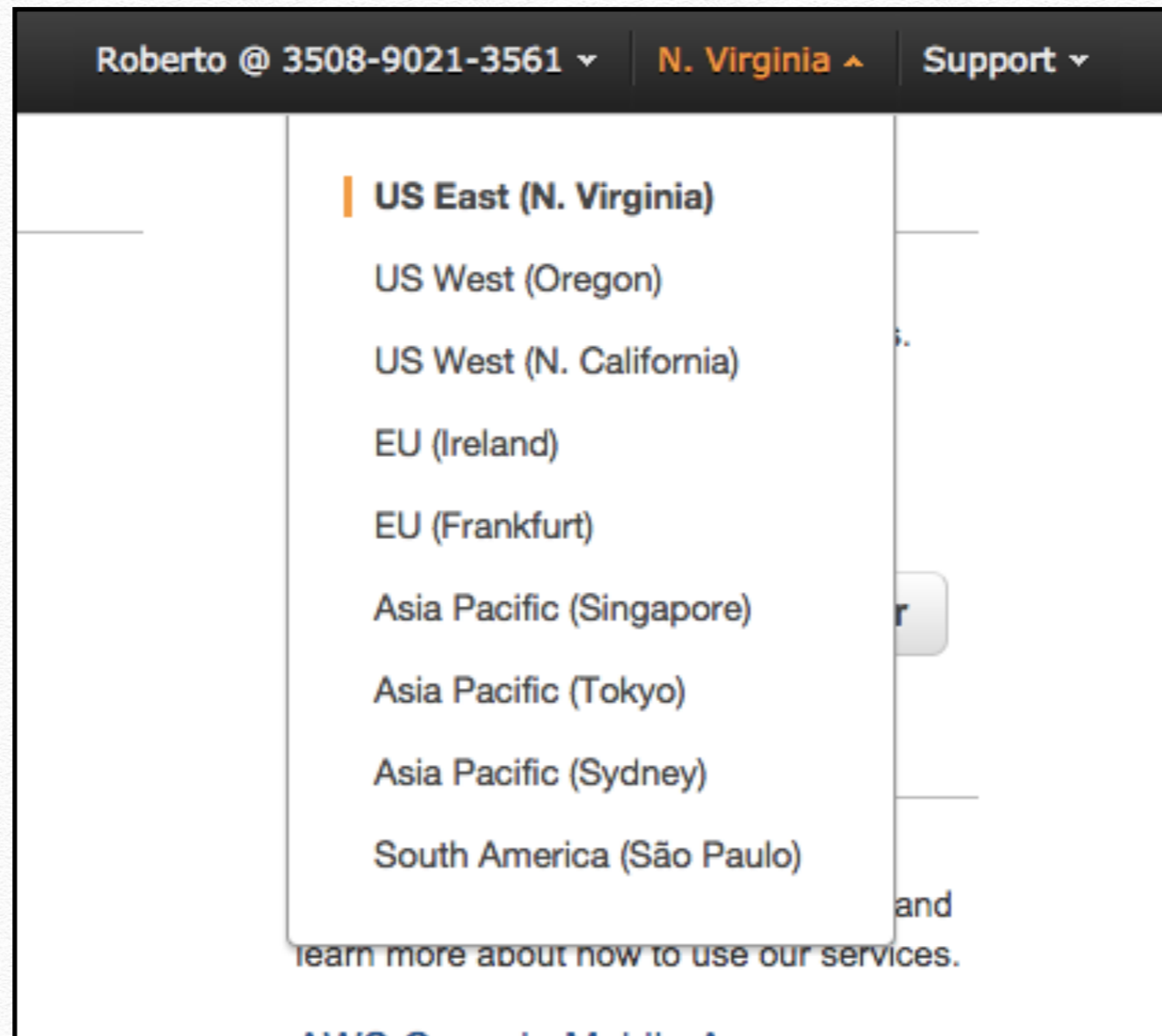
Below this section, there is a 'What's new?' section with two announcements:

- Announcing AWS Batch**: Now generally available, AWS Batch enables developers, scientists, and engineers to process large-scale batch jobs with ease. [Learn more](#)
- Announcing Amazon Lightsail**: See how this new service allows you to launch and manage your



# Hadoop 3 on AMAZON

## Regions



The screenshot shows the top navigation bar of the AWS Management Console. It includes the user's name and contact information 'Roberto @ 3508-9021-3561', the current region 'N. Virginia', and a 'Support' dropdown menu. The 'Regions' dropdown menu is open, displaying a list of available regions. The 'US East (N. Virginia)' region is highlighted with an orange bar on the left. Below the list, there is a link to learn more about how to use AWS services.

Roberto @ 3508-9021-3561 ▾ N. Virginia ▲ Support ▾

- US East (N. Virginia)**
- US West (Oregon)
- US West (N. California)
- EU (Ireland)
- EU (Frankfurt)
- Asia Pacific (Singapore)
- Asia Pacific (Tokyo)
- Asia Pacific (Sydney)
- South America (São Paulo)

and  
learn more about how to use our services.



# Hadoop 3 on AMAZON

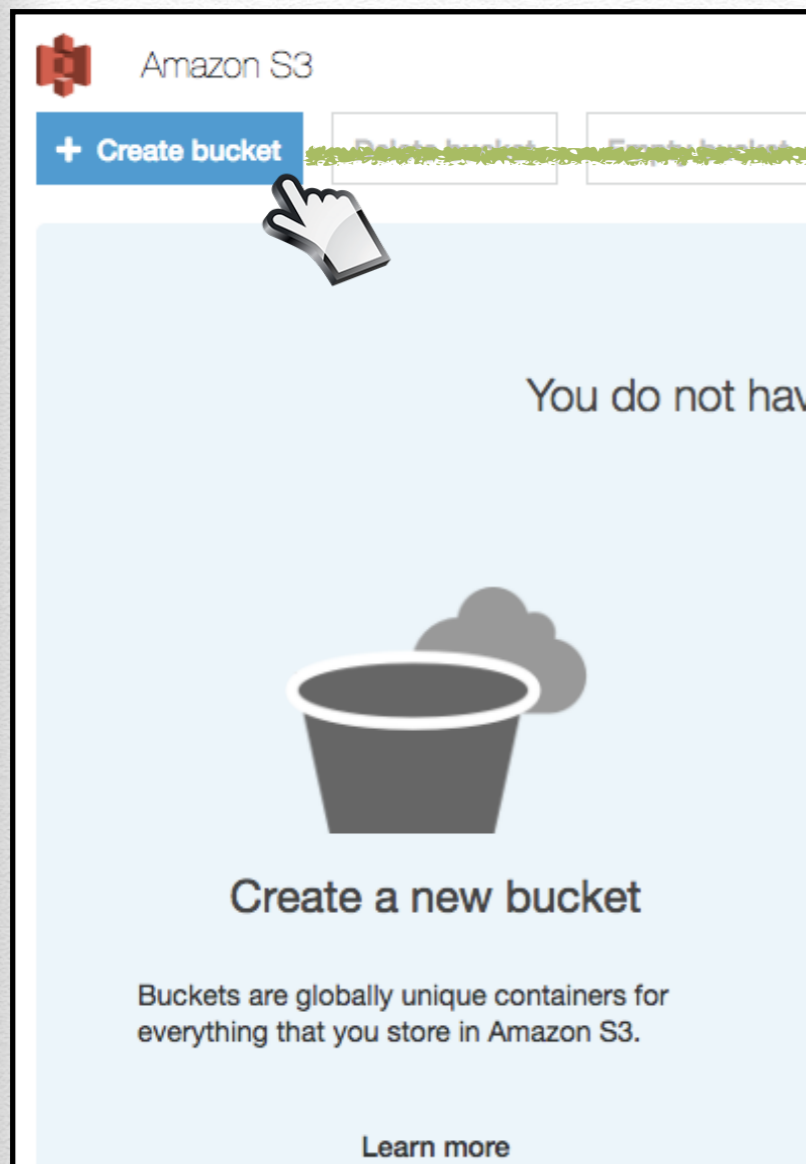
## S3 and buckets





# Hadoop 3 on AMAZON

## S3 and buckets



Create bucket

1 Name and region   2 Set properties   3 Set permissions   4 Review

Name and region

Bucket name ⓘ

robertobigdatabucket

Region

US East (N. Virginia) ▾

Copy settings from an existing bucket

You have no buckets 0 Buckets ▾

Create   Cancel   Next

This screenshot shows the 'Create bucket' modal window. The 'Name and region' step is active. The bucket name is 'robertobigdatabucket' and the region is 'US East (N. Virginia)'. The 'Copy settings from an existing bucket' dropdown shows 'You have no buckets'.



# Hadoop 3 on AMAZON

## S3 and buckets

### Create bucket ✕

1 Name and region ✓ 2 Set properties ✓ 3 Set permissions 3 4 Review 4

▼ Manage users

User ID <span>?</span>	Objects <span>?</span>	Object permissions <span>?</span>	
rde79(Owner)	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write	<input checked="" type="checkbox"/> Read <input checked="" type="checkbox"/> Write	<span>✕</span>

▼ Manage public permissions

Group <span>?</span>	Objects <span>?</span>	Object permissions <span>?</span>	
Any authenticated AWS user	<input type="checkbox"/> Read <input type="checkbox"/> Write	<input type="checkbox"/> Read <input type="checkbox"/> Write	
Everyone	<input type="checkbox"/> Read <input type="checkbox"/> Write	<input type="checkbox"/> Read <input type="checkbox"/> Write	

[Previous](#) [Next](#)



# Hadoop 3 on AMAZON

## S3 and buckets

### Create bucket ✕

Name and region     Set properties     Set permissions    **4** Review

---

**Name and region** Edit

**Bucket name** robertobigatabucket    **Region** US East (N. Virginia)

---

**Properties** Edit

<b>Versioning</b>	Disabled
<b>Logging</b>	Disabled
<b>Tagging</b>	0 Tags

---

**Permissions** Edit

<b>Users</b>	1
<b>Public permissions</b>	Disabled

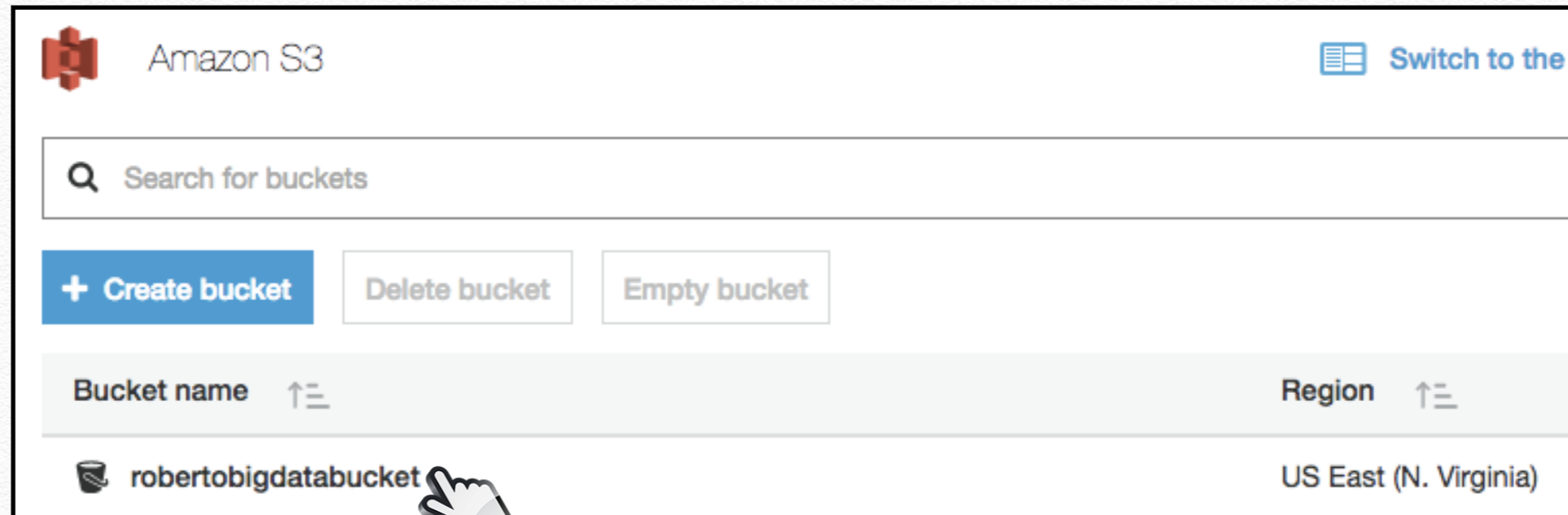
---

Previous Create bucket



# Hadoop 3 on AMAZON

## S3 and buckets



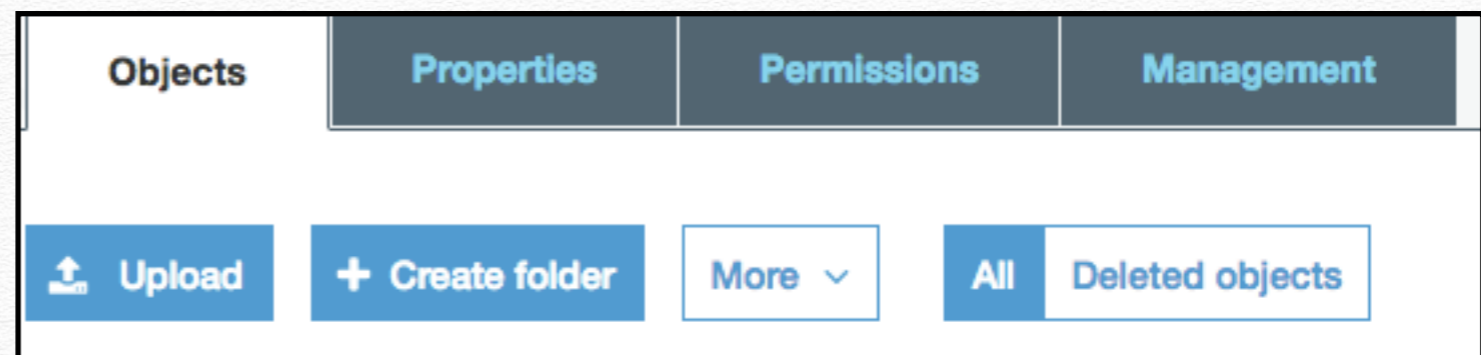
Amazon S3

Search for buckets

+ Create bucket Delete bucket Empty bucket

Bucket name	Region
robertobigdatabucket	US East (N. Virginia)

The screenshot shows the Amazon S3 console interface. At the top, there's the Amazon S3 logo and a search bar labeled "Search for buckets". Below the search bar are three buttons: "+ Create bucket" (highlighted in blue), "Delete bucket", and "Empty bucket". A table lists buckets with columns for "Bucket name" and "Region". One bucket, "robertobigdatabucket", is listed in the "US East (N. Virginia)" region. A hand cursor is pointing at the bucket name, and a green arrow points from this bucket to the detailed view below.



Objects Properties Permissions Management

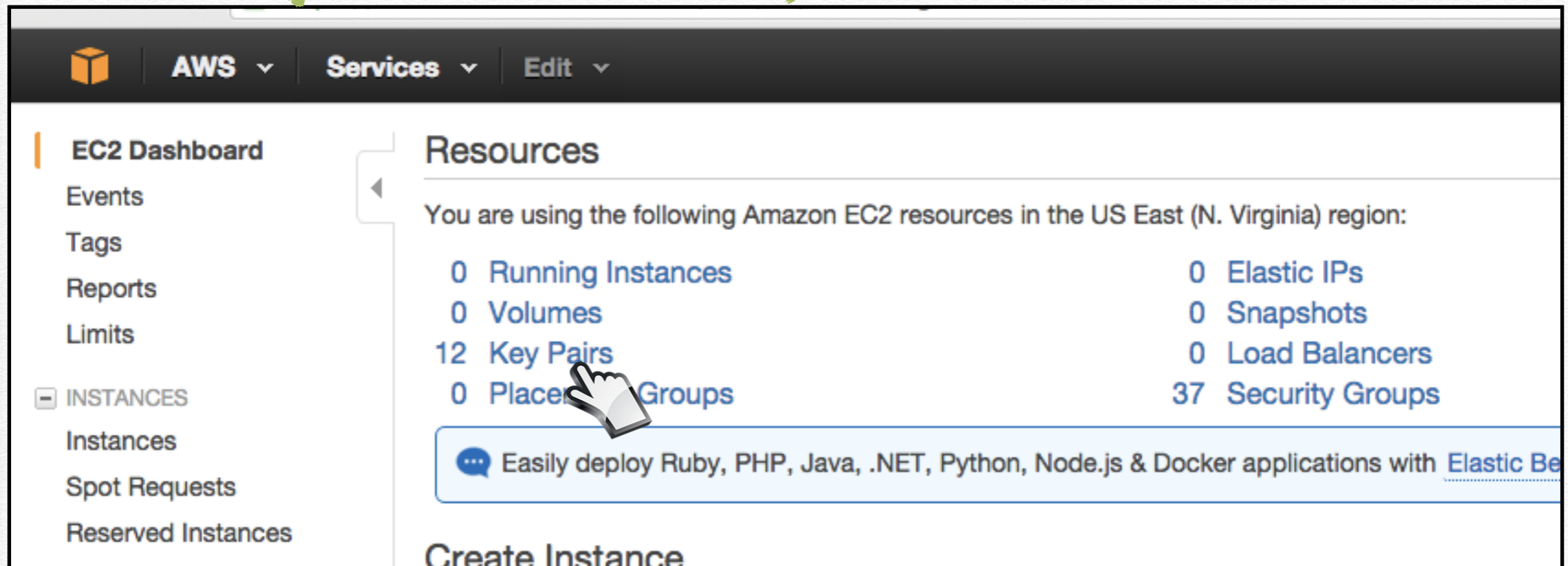
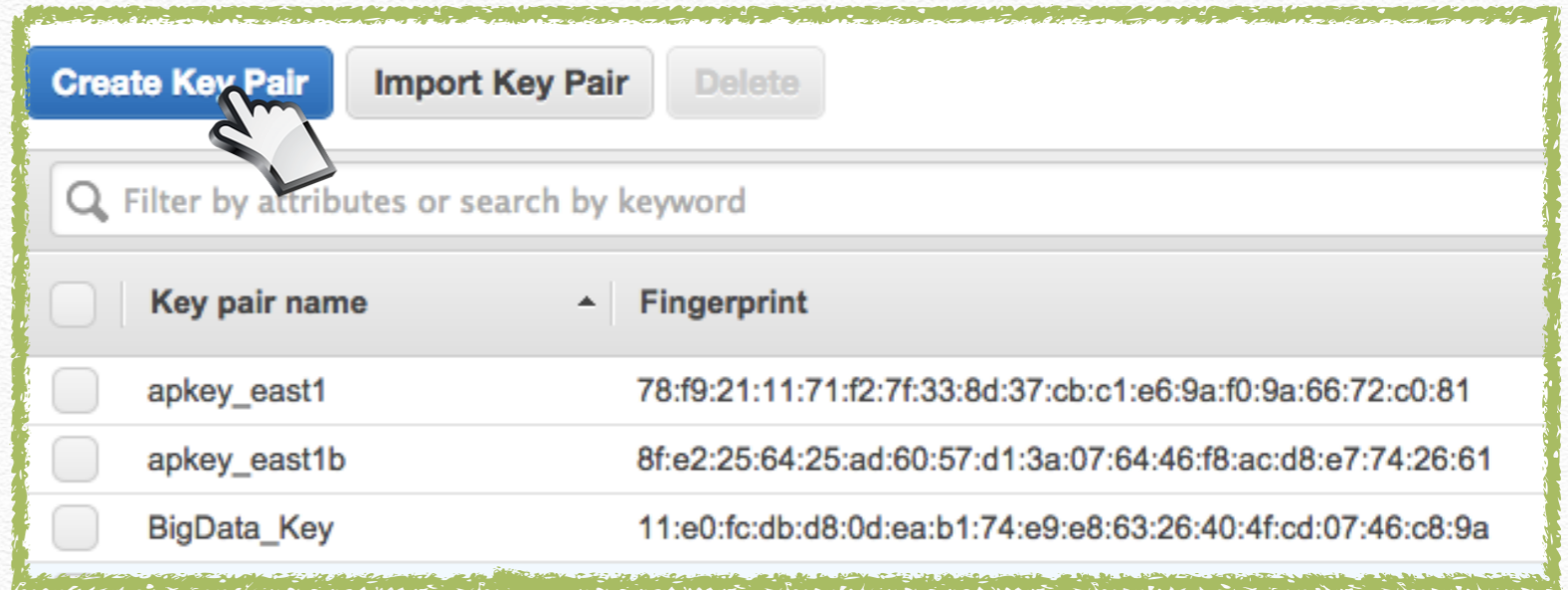
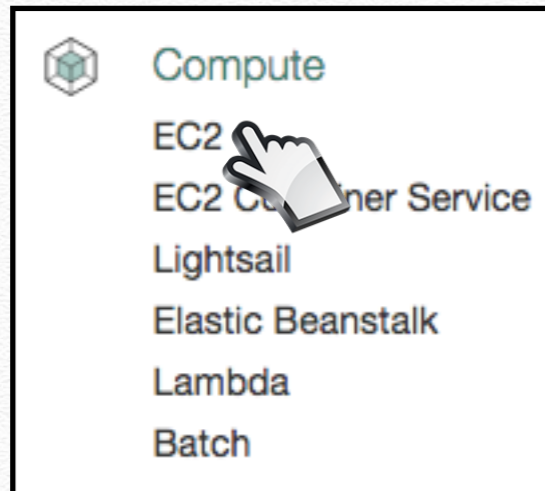
Upload + Create folder More All Deleted objects

The screenshot shows the detailed view of a bucket in the Amazon S3 console. At the top, there are four tabs: "Objects", "Properties" (highlighted in dark blue), "Permissions", and "Management". Below the tabs are several buttons: "Upload" (with an upload icon), "+ Create folder" (highlighted in blue), "More" (with a dropdown arrow), "All", and "Deleted objects".



# Hadoop 3 on AMAZON

## EC2 virtual servers

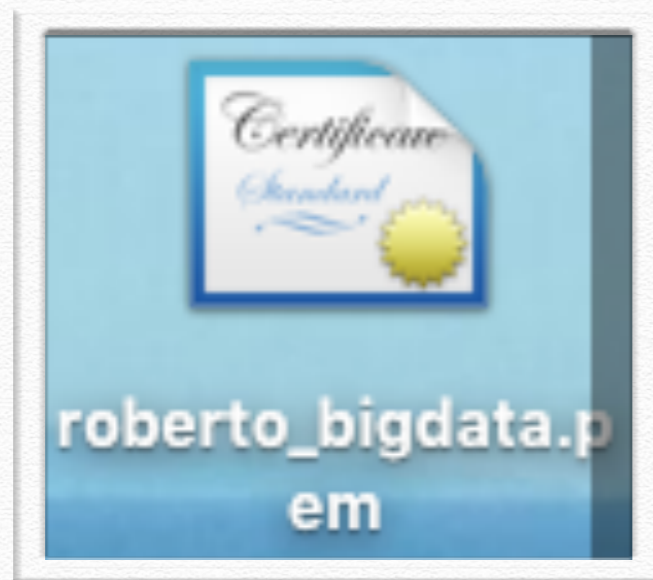




# Hadoop 3 on AMAZON

Key pair name	Fingerprint
<input checked="" type="checkbox"/> roberto_bigdata	40:1d:a1:c1:25:49:d7:74:20:16:6d:54:88:42:45:9d:85:7b:f6:48

file .pem

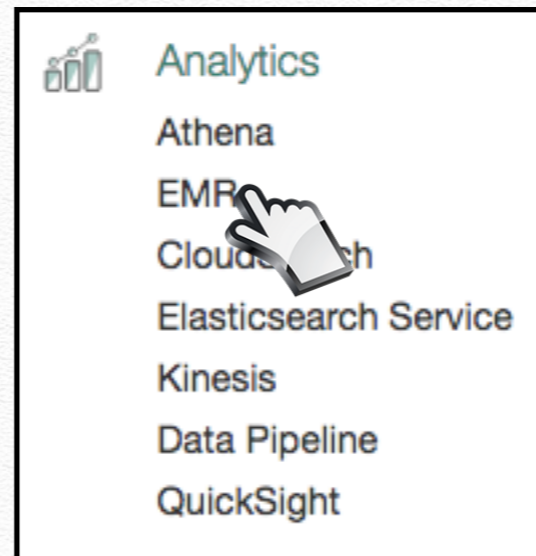


```
-----BEGIN RSA PRIVATE KEY-----
MIIeowIBAAKCAQEA1y1mGpoREC945ImOEqM01VxOX3ICShOOrZTzf1tC9Oj4ReqChRuuF3X7ndvo
Y8LLTWKc4p/qFHjya9jjvIkOTcu4RuB8h8cP/HZhOZu1tZy1/XobA+eIopKBuhQACATXbs0ERIDD
fC6sLwwurMJHU0OznElbEJsMN/nEK8IYv4uu2gqfzbgoxHibHd6jiHFEk17ymZIB8D0dQhFGwBII
B29grlmyUCC1miA1pp8L3eHBqHXs0Vlw94tLM+ktLgeA4mdP4jvADRLt0quozmp89dj2ZHO8npGq
DMkDUJsXJIBcfHT7dmGBIxATuLdAjwHt56nwchA9dGXCVI7YfIjmtwIDAQABAolBAHkVrAJGNInk
TOVa+c7VFnMF+XhSVRI4RS66xfch1ODahHNbjsz2kZXUJ55iVDhnMI4+osglcwHIOaqkpyq9+VWf
0PFzdVo2k3Fe8EEptSwYnnSgFLmyzdDTBrs+a/IXP1+zcLZXuymTXgMml+FRhi99xoGo6dzDUzHg
lBqHo8OM/wuFk3zStDIRJta1q6T12LyCyxQ5IDeWo5MJIBfiKuTQOE573UILHp2tyIEeYcQKHpBO
BW9jkafue/QramwsyZmyBtRMcZm4wDmRqm0YjfGRP+gNCYD/bXhofqzHlsjFsuxVIMhLIQrYZm9A
iFzdZBSavWw0rxlc2zUdgj3FAskCgYEA/aExJIQwvSdvHb0AedqpTvfbmj9iAKJ1xYwbz04grnph
Nw1eWt40vjVRZaVCaNk33QjIHZJBr5w7BFqcdLuoes2kGMnOFANK7kQ0HAnPDvQttt3gqlvcn6
Eq23C7Dhj+3qKSI2g4b7qSW43eys/FiYMvWwXNfo1L0PDmtGj+MCgYEA2TA1uMX4Vu3SPDDLXIV
cExRw5RY60dUwnfbqeaITf+VNui+9XZuXSBzQj0782xmeJwzbFHSQFzeZzMABWYL8tUufVutoGD
1SkC0uxcYq/5URD3zw5dgybwIjZMzcTpN3Ug7u6oeNPRYQ9FDcfMMMpEyBjFM2JzGn2KHmcSNh0C
gYADfaj1hDQ/hjlrSVymULif2h11pAWvSi0iaOIBEyWz+o83+MEhswk6zPUP1xRw6Pxx2ESQ/8/F
ff2D4kF6zPQH3zlr1tcemaMOnEtb+Z2zC0Xfv7FlavQZBpyggS+Rw593laklY8koSkVQcKp4s3c2
CeoEWY7xiT40JnqcuOGTQKBgEhXQ32RCz/BjSaBRHdt2iR2d0Gctv9fGf9QNu1naP2B+ie/pjDn
UQbxpl5rrIH1nHNANij9KvZJMnJOZLCegLtiqYzrD/5gM04bw+IHclo4rP1wfmOMKd+WZ6MjDN/4
94IOTTLocVsVioces+x8ISobJT/U6FJON3KaYBfyUuGNAoGBAKmZhyvN4M7Dkcj3nCRFahyrMnaz
okfHacmb3vQ3Q0wSP51k8YMM4qXlqGhCWm3wY8jsufUt+Lk5lcfNjTLyAMP98WY5LppZv/80IAZ5
Wltx8fAuMRtoR5ojqp3wo1ZUNzp+0saPPVQ7n6CBWjj3TbzU5w6bR4UI2kiPomt40rEx
-----END RSA PRIVATE KEY-----
```



# Hadoop 3 on AMAZON

## Elastic Map Reduce



A screenshot of the AWS Elastic MapReduce Cluster List interface. The interface shows the following elements:

- Header: Elastic MapReduce ▾ Cluster List
- Buttons: Create cluster (highlighted with a mouse cursor), View details, Clone, Terminate
- Filter: All clusters ▾ Filter clusters ... 1 cluster (all loaded)
- Table headers: Name, ID




# Hadoop 3 on AMAZON

## EMR cluster

### General Configuration

**Cluster name**

**Logging** ⓘ

**S3 folder**  

**Launch mode**  **Cluster** ⓘ  **Step execution** ⓘ



# Hadoop 3 on AMAZON

EMR cluster

cluster

Logging ⓘ

S3 folder

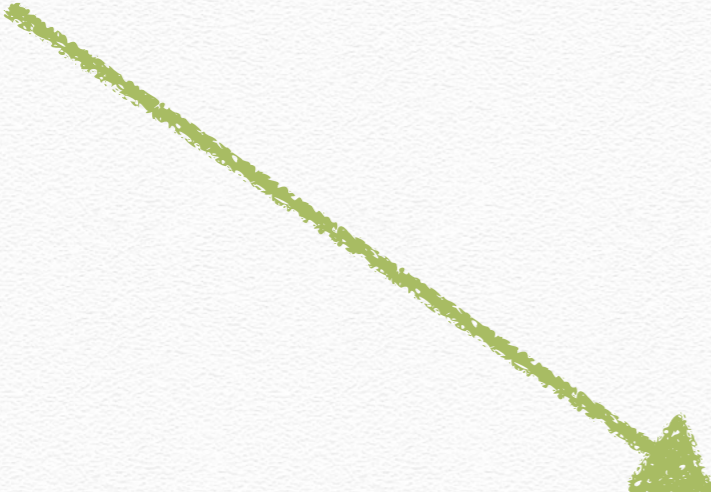
Cluster ⓘ  Step execution ⓘ

Select S3 Folder

< > URL:

robertobigdatabucket/

Cancel Select






# Hadoop 3 on AMAZON

## EMR cluster

### General Configuration

Cluster name

Logging 

S3 folder  

Launch mode  Cluster 

With Cluster, EMR creates a cluster with a set of specified applications.




# Hadoop 3 on AMAZON



## EMR cluster

### General Configuration

Cluster name

Logging 

S3 folder  

Launch mode  Cluster   Step execution 

With Step execution, EMR will create a cluster, execute added steps and terminate when done.



# Hadoop 3 on AMAZON

## EMR cluster

### Software configuration

**Vendor**  Amazon  MapR

**Release**   

- Applications**
- Core Hadoop: Hadoop 2.7.3 with Ganglia 3.7.2, Hive 2.1.1, Hue 3.11.0, Mahout 0.12.2, Pig 0.16.0, and Tez 0.8.4
  - HBase: HBase 1.3.0 with Ganglia 3.7.2, Hadoop 2.7.3, Hive 2.1.1, Hue 3.11.0, Phoenix 4.9.0, and ZooKeeper 3.4.9
  - Presto: Presto 0.166 with Hadoop 2.7.3 HDFS and Hive 2.1.1 Metastore
  - Spark: Spark 2.1.0 on Hadoop 2.7.3 YARN with Ganglia 3.7.2 and Zeppelin 0.7.0



# Hadoop 3 on AMAZON

## EMR cluster

### Software configuration

**Vendor**  Amazon  MapR

**Release**  ⌵ i

**Applications**  All applications: Hadoop 2.4.0, Hive 0.13.1, and Pig 0.12.0



<http://doc.mapr.com/display/MapR/MapR+Overview>

MapR 5.0 Documentation / Home ⚙ Tools ▾

## MapR Overview

MapR is a complete enterprise-grade distribution for Apache Hadoop. The MapR Distribution for Apache Hadoop has been engineered to improve Hadoop's reliability, performance, and ease of use. The MapR distribution provides a full Hadoop stack that includes the MapR File System (MapR-FS), MapReduce, a complete Hadoop ecosystem, and the MapR Control System user interface. You can use MapR with Apache Hadoop, HDFS, and MapReduce APIs.

The following image displays a high-level view of the MapR Distribution for Apache Hadoop:

```
graph LR; MCS[MapR Control System MCS]; subgraph Tools; direction TB; T1[Whirr]; T2[Sqoop]; T3[Oozie]; end; subgraph Ecosystem; direction TB; E1[Hive]; E2[Pig]; E3[Flume]; E4[HCatalog]; E5[Mahout]; E6[Cascading]; end; subgraph Core; direction TB; C1[MapReduce]; C2[Apache HBase]; end; subgraph Storage; direction TB; S1[MapR-FS]; end; MCS --- Tools; MCS --- Ecosystem; MCS --- Core; MCS --- Storage; Storage --- Hbase[Hbase API]; Storage --- NFS[NFS Interface]; Storage --- HDFS[HDFS API];
```

The MapR distribution provides several unique features that address common concerns with Apache Hadoop:



# Hadoop 3 on AMAZON

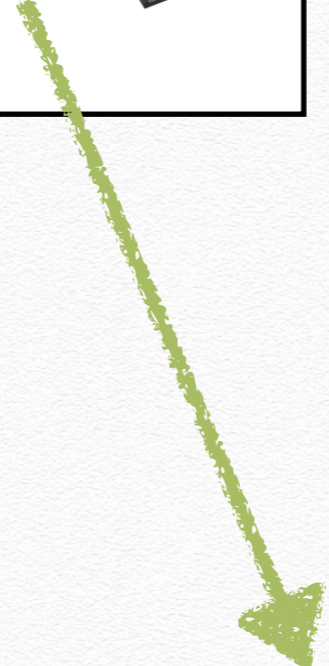
## EMR cluster

**Hardware configuration**

**Instance type**  

**Number of instances**  (1 master and 2 core nodes)

- m2.xlarge
- m2.2xlarge
- m2.4xlarge
- Storage Optimized
- d2.xlarge
- d2.2xlarge
- d2.4xlarge
- d2.8xlarge
- hs1.8xlarge
- i2.xlarge
- i2.2xlarge
- i2.4xlarge
- i2.8xlarge
- GPU Instances
- g2.2xlarge
- Storage Optimized (Previous Generation)
- hi1.4xlarge
- General Purpose (Previous Generation)
- m1.medium
- m1.large
- m1.xlarge
- General Purpose
- ✓ m3.xlarge
- m3.2xlarge
- m4.large
- m4.xlarge
- m4.2xlarge
- m4.4xlarge
- m4.10xlarge
- m4.16xlarge
- Memory Optimized
- r3.xlarge
- r3.2xlarge
- r3.4xlarge
- r3.8xlarge
- r4.xlarge
- r4.2xlarge
- r4.4xlarge
- r4.8xlarge
- r4.16xlarge






# Hadoop 3 on AMAZON

## EMR cluster

### Security and access

EC2 key pair    [Learn how to create an EC2 key pair.](#)

Permissions  Default  Custom

Use default IAM roles. If roles are not present, they will be automatically created for you with managed policies for automatic policy updates.

EMR role [EMR\\_DefaultRole](#) 

EC2 instance profile [EMR\\_EC2\\_DefaultRole](#) 

✓ Choose an option

Proceed without an EC2 key pair

roberto\_bigdata



# Hadoop 3 on AMAZON

## EMR cluster


**Cluster: My cluster** **Starting** Configuring cluster software

**Connections:** [Enable Web Connection](#) – Hue, Ganglia, Resource Manager ... (View All)

**Master public DNS:** ec2-54-159-62-237.compute-1.amazonaws.com [SSH](#)

**Tags:** -- [View All / Edit](#)

---

Summary	Configuration Details
<b>ID:</b> j-47LWBSL8AKPS	<b>Release label:</b> emr-5.4.0
<b>Creation date:</b> 2017-04-04 11:55 (UTC+2)	<b>Hadoop distribution:</b> Amazon 2.7.3
<b>Elapsed time:</b> 2 minutes	<b>Applications:</b> Ganglia 3.7.2, Hive 2.1.1, Hue 3.11.0, Mahout 0.12.2, Pig 0.16.0, Tez 0.8.4
<b>Auto-terminate:</b> No	<b>Log URI:</b> s3://robertobigdatabucket/ 
<b>Termination protection:</b> Off <a href="#">Change</a>	<b>EMRFS consistent view:</b> Disabled

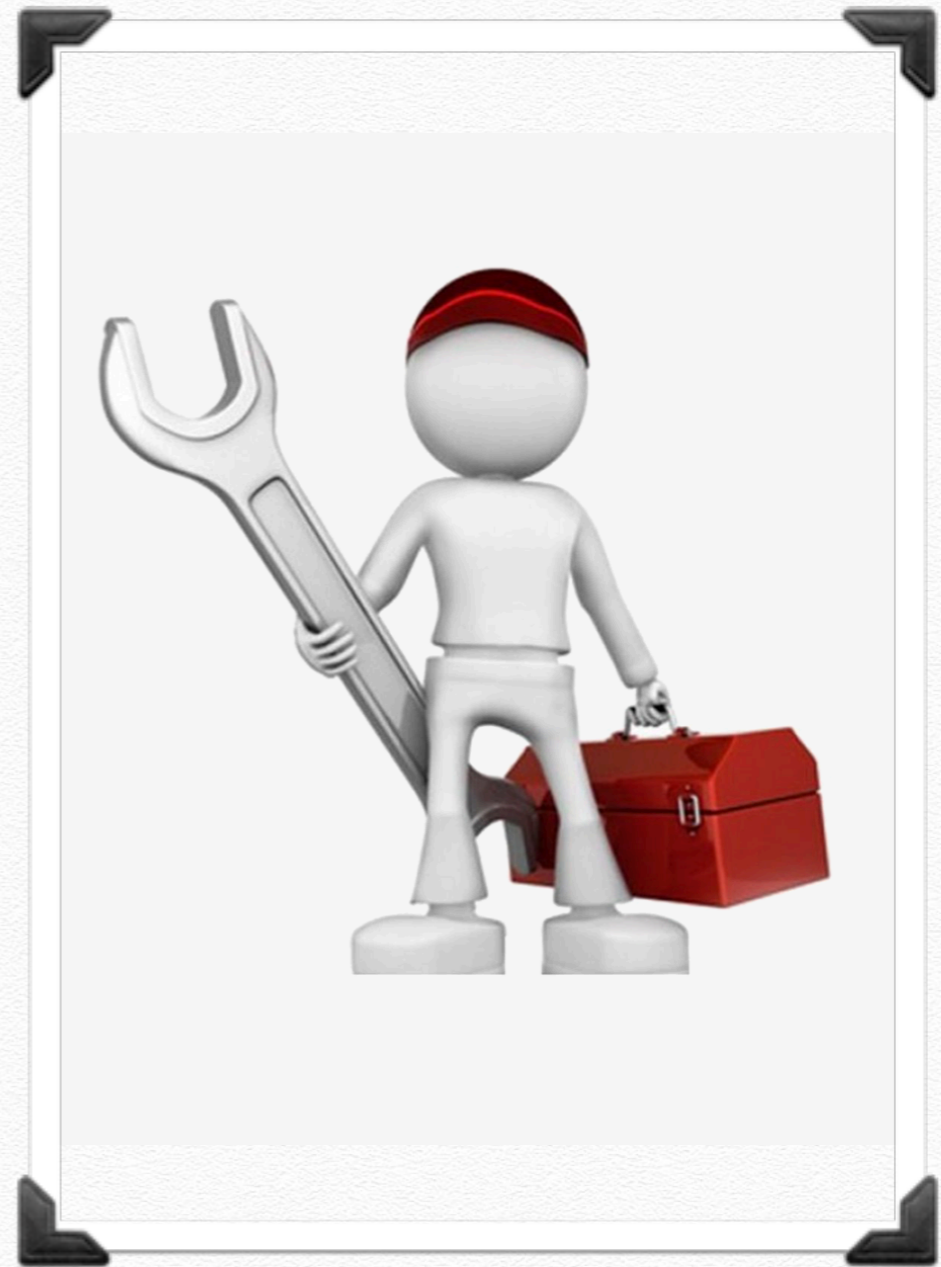
---

Network and Hardware	Security and Access
<b>Availability zone:</b> us-east-1c	<b>Key name:</b> roberto_bigdata
<b>Subnet ID:</b> --	<b>EC2 instance profile:</b> EMR_EC2_DefaultRole
<b>Master:</b> <b>Bootstrapping</b> 1 m3.xlarge	<b>EMR role:</b> EMR_DefaultRole
<b>Core:</b> <b>Provisioning</b> 2 m3.xlarge	<b>Visible to all users:</b> All <a href="#">Change</a>
<b>Task:</b> --	<b>Security groups for</b> <a href="#">sg-98da73f3</a> (ElasticMapReduce-Master: master)



[https://youtu.be/Z1je94H\\_jSg](https://youtu.be/Z1je94H_jSg)

- ❖ Web video to **configure** AWS cluster





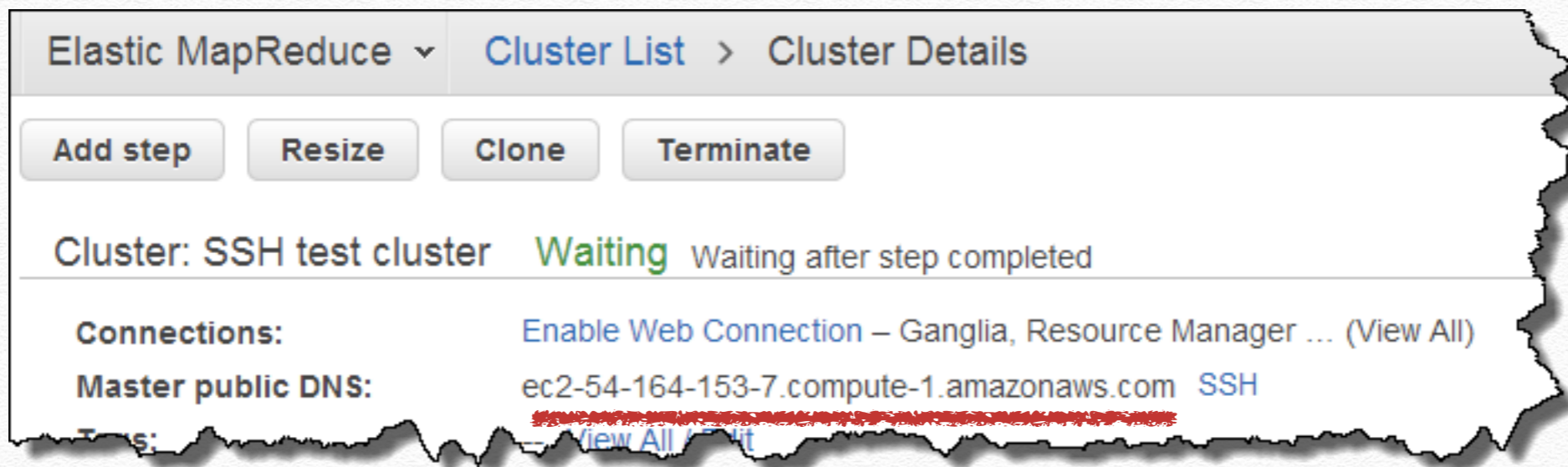
# Hadoop 3 Running on AWS

- ❖ Authorization of the .pem file

```
$:~ chmod 400 roberto_bigdata.pem
```

- ❖ Upload files (data and your personal jars) on hadoop of EMR cluster:

```
$:~ scp -i <file .pem> <file> hadoop@<DNS_EMR_CLUSTER>:~
```



The screenshot shows the AWS EMR console interface. At the top, there are navigation tabs: 'Elastic MapReduce' (with a dropdown arrow), 'Cluster List', and 'Cluster Details'. Below the tabs are four buttons: 'Add step', 'Resize', 'Clone', and 'Terminate'. The main content area displays the cluster name 'SSH test cluster' followed by the state 'Waiting' in green text and the reason 'Waiting after step completed'. Underneath, there is a section for 'Connections' with a link 'Enable Web Connection - Ganglia, Resource Manager ... (View All)'. Below that, the 'Master public DNS' is listed as 'ec2-54-164-153-7.compute-1.amazonaws.com' with 'SSH' next to it. At the bottom, there is a link 'View All / Edit'.

```
$:~ scp -i roberto_bigdata.pem ./example_data/words.txt  
hadoop@ec2-54-164-153-7.compute-1.amazonaws.com:~
```

```
$:~ scp -i roberto_bigdata.pem ./jar/Ex1-1.jar  
hadoop@ec2-54-164-153-7.compute-1.amazonaws.com:~
```



# Hadoop 3 Running on AWS

- ❖ **Connection to** hadoop of EMR cluster:

```
$:~ ssh hadoop@<DNS_EMR_CLUSTER> -i <file .pem>
```

```
$:~ ssh hadoop@ec2-54-164-153-7.compute-1.amazonaws.com  
-i roberto_bigdata.pem
```

- ❖ Then you can execute MR jobs as in your local machine





# Hadoop 3 Running on AWS

❖ **Execution** hadoop of EMR cluster:

```
[hadoop@ip-172-31-45-32 ~]$ ls
```

```
Ex1-1.jar words.txt
```

```
[hadoop@ip-172-31-45-32 ~]$ hdfs dfs -mkdir /input
```

```
[hadoop@ip-172-31-45-32 ~]$ hdfs dfs -mkdir /output
```

```
[hadoop@ip-172-31-45-32 ~]$ hadoop jar Ex1-1.jar wordcount/WordCount  
/input/words.txt /output/results_words
```



# Hadoop 3 Running on AWS

❖ **View** hadoop execution results of EMR cluster:

```
[hadoop@ip-172-31-45-32 ~]$ hdfs dfs -ls /output/results_words
```

```
/output/results_words/_SUCCESS  
/output/results_words/part-r-00000  
/output/results_words/part-r-00001  
/output/results_words/part-r-00002
```

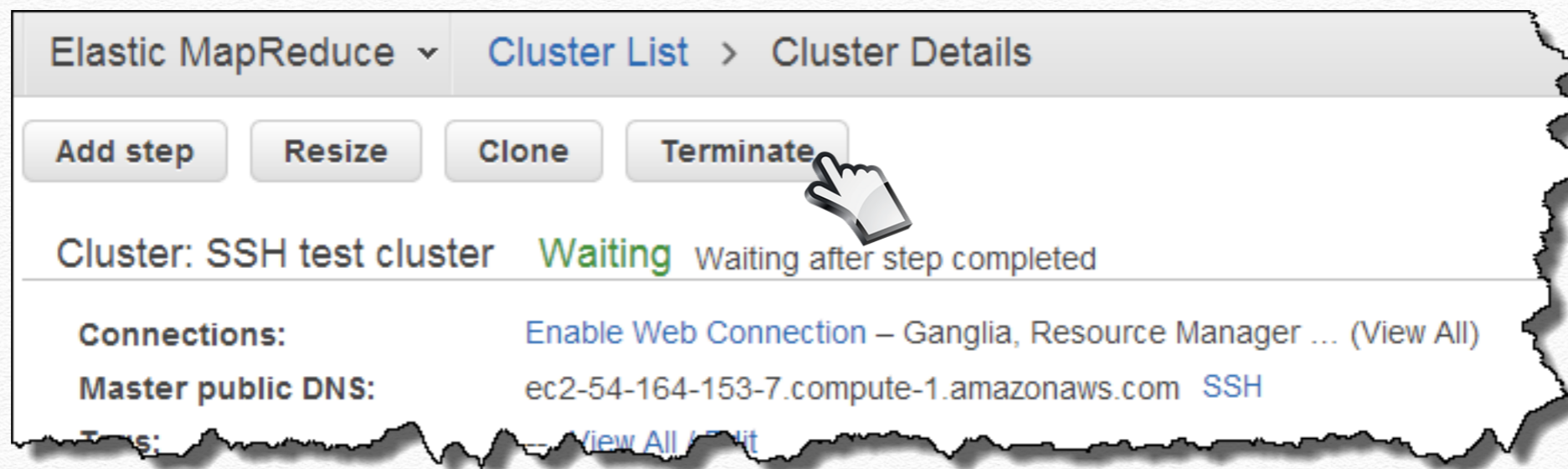
```
[hadoop@ip-172-31-45-32 ~]$ hdfs dfs -cat /output/results_words/part-r-00000
```

```
panca 2  
otto 1
```



# Hadoop 3 Running on AWS

- ❖ Finally **TERMINATE** your cluster





<https://youtu.be/JZD7MjIzwCQ>

- ❖ Web video to **execute** a **Java** MapReduce job on AWS Cluster





# View Web Interfaces Hosted on Amazon EMR Clusters

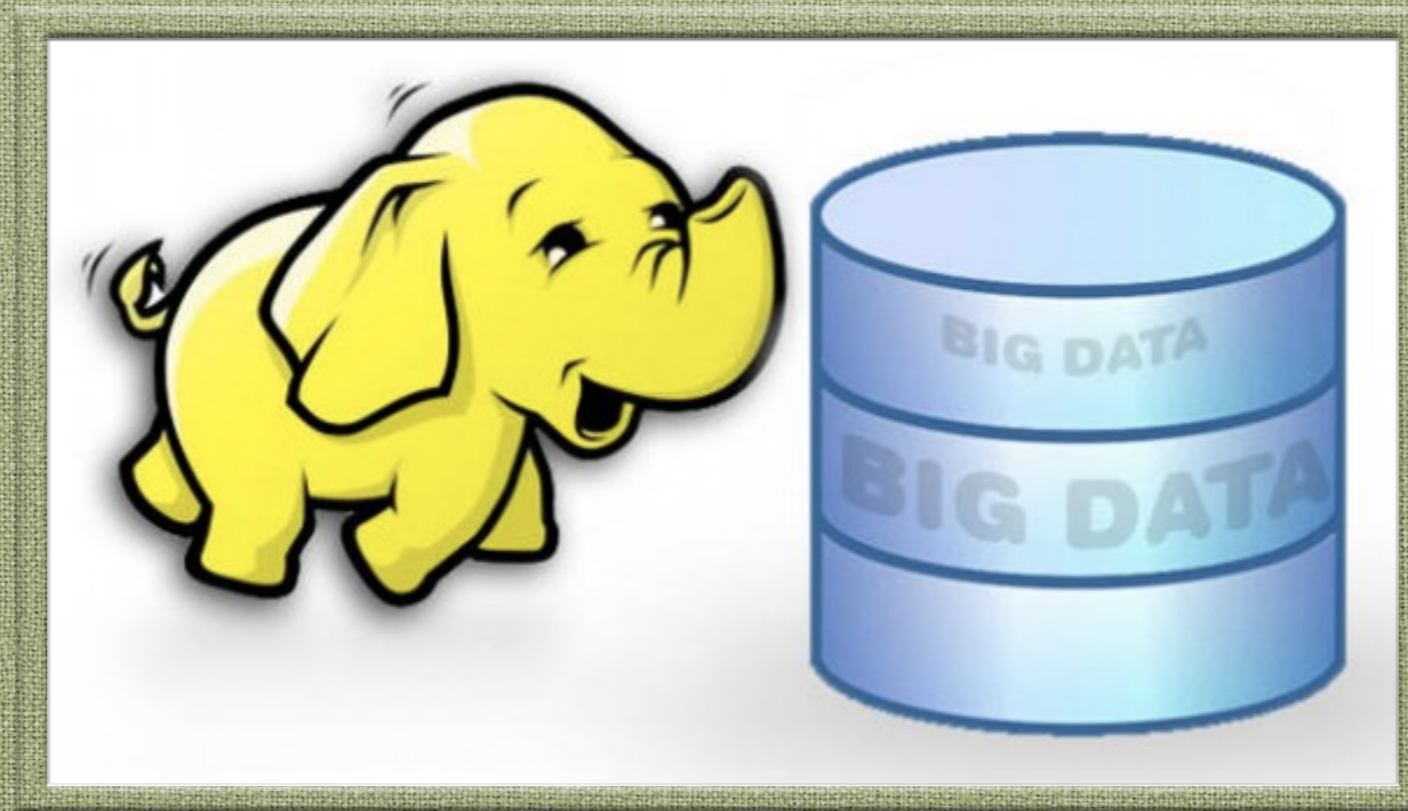
- ❖ To access **web interfaces**, you must edit the **security groups** associated with master and core instances so that they have an inbound rule that allows SSH traffic (port 22) from trusted clients, such as your computer's IP address.
- ❖ For more information about modifying security group rules, see [Adding Rules to a Security Group](#) in the *Amazon EC2 User Guide for Linux Instances*.



# View Web Interfaces Hosted on Amazon EMR Clusters

Name of interface	URI
Ganglia	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;/ganglia/">http://<i>master-public-dns-name</i>/ganglia/</a>
Hadoop HDFS NameNode	<a href="https://&lt;i&gt;master-public-dns-name&lt;/i&gt;:50470/">https://<i>master-public-dns-name</i>:50470/</a>
Hadoop HDFS DataNode	<a href="https://&lt;i&gt;coretask-public-dns-name&lt;/i&gt;:50475/">https://<i>coretask-public-dns-name</i>:50475/</a>
HBase	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:16010/">http://<i>master-public-dns-name</i>:16010/</a>
Hue	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:8888/">http://<i>master-public-dns-name</i>:8888/</a>
JupyterHub	<a href="https://&lt;i&gt;master-public-dns-name&lt;/i&gt;:9443/">https://<i>master-public-dns-name</i>:9443/</a>
Livy	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:8998/">http://<i>master-public-dns-name</i>:8998/</a>
Spark HistoryServer	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:18080/">http://<i>master-public-dns-name</i>:18080/</a>
Tez	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:8080/tez-ui">http://<i>master-public-dns-name</i>:8080/tez-ui</a>
YARN NodeManager	<a href="http://&lt;i&gt;coretask-public-dns-name&lt;/i&gt;:8042/">http://<i>coretask-public-dns-name</i>:8042/</a>
YARN ResourceManager	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:8088/">http://<i>master-public-dns-name</i>:8088/</a>
Zeppelin	<a href="http://&lt;i&gt;master-public-dns-name&lt;/i&gt;:8890/">http://<i>master-public-dns-name</i>:8890/</a>





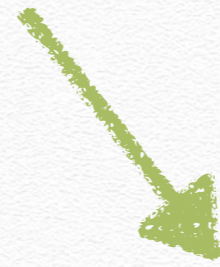
**Let's start with some examples!**



# Example: LastFM Listeners per Track

Consider the following log file

UserId	TrackId	Shared	Radio	Skip
111115	222	0	1	0
111113	225	1	0	0
111117	223	0	1	1
111115	225	1	0	0



- Number of unique listeners
- Number of times the track was shared with others
- Number of times the track was listened to on the radio
- Number of times the track was listened to in total
- Number of times the track was skipped on the radio

taken from XYZ.com that is an online music website where users listen to various tracks



# Example: LastFM Listeners per Track

Write a MapReduce code to find out unique listeners per track

111115 | 222 | 0 | 1 | 0

111113 | 225 | 1 | 0 | 0

111117 | 223 | 0 | 1 | 1

111115 | 225 | 1 | 0 | 0



222 1

223 1

225 2



# Example: LastFM Listeners per Track

To make it simple to remember the data sequence, let's create a constants class as shown below

```
public class LastFMConstants {  
  
    public static final int USER_ID = 0;  
    public static final int TRACK_ID = 1;  
    public static final int IS_SHARED = 2;  
    public static final int RADIO = 3;  
    public static final int IS_SKIPPED = 4;  
  
}
```



# Example: LastFM Listeners per Track

lets create the mapper class which would emit intermediate key value pairs as (TrackId, UserId) as shown below

```
public static class UniqueListenersMapper extends
Mapper< Object , Text, IntWritable, IntWritable > {
    IntWritable trackId = new IntWritable();
    IntWritable userId = new IntWritable();

public void map(Object key, Text value,
    Mapper< Object , Text, IntWritable, IntWritable > .Context context)
    throws IOException, InterruptedException {

    String[] parts = value.toString().split("[|]");
    trackId.set(Integer.parseInt(parts[LastFMConstants.TRACK_ID]));
    userId.set(Integer.parseInt(parts[LastFMConstants.USER_ID]));
    if (parts.length == 5) {
        context.write(trackId, userId);
    } else {
        // add counter for invalid records
        context.getCounter(COUNTERS.INVALID_RECORD_COUNT).increment(1L);
    }
}
}
```



# Example: LastFM Listeners per Track

You would have also noticed that we are using a counter here named **INVALID\_RECORD\_COUNT**, to count if there are any invalid records which are not coming the expected format. Remember, if we don't do this then in case of invalid records, our program might fail.



# Example: LastFM Listeners per Track

let's write a Reducer class to aggregate the results.

```
public static class UniqueListenersReducer extends
    Reducer< IntWritable , IntWritable, IntWritable, IntWritable> {
    public void reduce(IntWritable trackId, Iterable< IntWritable > userIds,
        Reducer< IntWritable , IntWritable, IntWritable, IntWritable>.Context
context) throws IOException, InterruptedException {

        Set< Integer > userIdSet = new HashSet< Integer >();
        for (IntWritable userId : userIds) {
            userIdSet.add(userId.get());
        }
        IntWritable size = new IntWritable(userIdSet.size());
        context.write(trackId, size);
    }
}
```



# Example: LastFM Listeners per Track

Now we can take look at the Driver class

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    if (args.length != 2) {
        System.err.println("Usage: uniquelisteners < in > < out >");
        System.exit(2);
    }
    Job job = new Job(conf, "Unique listeners per track");
    job.setJarByClass(UniqueListeners.class);
    job.setMapperClass(UniqueListenersMapper.class);
    job.setReducerClass(UniqueListenersReducer.class);
    job.setOutputKeyClass(IntWritable.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
    org.apache.hadoop.mapreduce.Counters counters = job.getCounters();
    System.out.println("No. of Invalid Records : "
        + counters.findCounter(COUNTERS.INVALID_RECORD_COUNT)
            .getValue());
}
```



# Example: Call Data Record (CDR) Analytics

Consider the following log file

```
FromPhoneNumber | ToPhoneNumber | CallStartTime | CallEndTime | STDFlag
9665128505 | 8983006310 | 2015-03-01 09:08:10 | 2015-03-01 10:12:15 | 1
9665128505 | 8983006310 | 2015-03-01 07:08:10 | 2015-03-01 08:12:15 | 0
9665128505 | 8983006310 | 2015-03-01 09:08:10 | 2015-03-01 09:12:15 | 1
9665128505 | 8983006310 | 2015-03-01 09:08:10 | 2015-03-01 10:12:15 | 0
9665128506 | 8983006310 | 2015-03-01 09:08:10 | 2015-03-01 10:12:15 | 1
9665128507 | 8983006310 | 2015-03-01 09:08:10 | 2015-03-01 10:12:15 | 1
```

taken a Telecom company keeping records for its subscribers in specific format.



# Example: Call Data Record (CDR) Analytics

- \* Now we have to write a map reduce code to find out all phone numbers who are making more than 60 mins of STD calls.
- \* Here if STD Flag is 1 that means it was as STD Call. STD is call which is made outside of your state or long distance calls.
- \* By identifying such subscribers, telecom company wants to offer them STD(Long Distance) Pack which would efficient for them instead spending more money without that package.

FromPhoneNumber	duration
9665128505	68
9665128506	64
9665128507	64



# Example: Call Data Record (CDR) Analytics

To make it simple to remember the data sequence, let's create a constants class as shown below

```
public class CDRConstants {  
  
    public static int fromPhoneNumber = 0;  
    public static int toPhoneNumber = 1;  
    public static int callStartTime = 2;  
    public static int callEndTime = 3;  
    public static int STDFlag = 4;  
  
}
```



# Example: Call Data Record (CDR) Analytics

lets create the mapper class which would emit intermediate key value pairs as (FromPhoneNumber, Duration), here we would also need to use our Java skills to calculate duration ( `CallEndTime - CallStartTime`). We are also making some manipulations to get duration in minutes

```
public static class TokenizerMapper extends
    Mapper< Object , Text, Text, LongWritable> {

    Text phoneNumber = new Text();
    LongWritable durationInMinutes = new LongWritable();

    public void map(Object key, Text value,
        Mapper< Object , Text, Text, LongWritable>.Context context)
        throws IOException, InterruptedException {
        String[] parts = value.toString().split("[|]");
        if (parts[CDRConstants.STDFlag].equalsIgnoreCase("1")) {

            phoneNumber.set(parts[CDRConstants.fromPhoneNumber]);
            String callEndTime = parts[CDRConstants.callEndTime];
            String callStartTime = parts[CDRConstants.callStartTime];
            long duration = toMillis(callEndTime) - toMillis(callStartTime);
            durationInMinutes.set(duration / (1000 * 60));
            context.write(phoneNumber, durationInMinutes);
        }
    }

    private long toMillis(String date) { ... }
}
```



# Example: Call Data Record (CDR) Analytics

lets create the mapper class which would emit intermediate key value pairs as (FromPhoneNumber, Duration), here we would also need to use our Java skills to calculate duration ( `CallEndTime - CallStartTime`). We are also making some manipulations to get duration in minutes

```
public static class TokenizerMapper extends
    Mapper< Object , Text, Text, LongWritable> {

    ...
    private long toMillis(String date) {

        SimpleDateFormat format = new SimpleDateFormat(
            "yyyy-MM-dd HH:mm:ss");
        Date dateFrm = null;
        try {
            dateFrm = format.parse(date);

        } catch (ParseException e) {

            e.printStackTrace();
        }
        return dateFrm.getTime();
    }
}
```



# Example: Call Data Record (CDR) Analytics

Now that we have already done majority of things in Mapper Class itself, here a reduce would be a simple Sum Reducer. Here is how the code would look like

```
public static class SumReducer extends
    Reducer< Text , LongWritable, Text, LongWritable> {
    private LongWritable result = new LongWritable();

    public void reduce(Text key, Iterable< LongWritable> values,
        Reducer< Text , LongWritable, Text, LongWritable>.Context context)
        throws IOException, InterruptedException {
        long sum = 0;
        for (LongWritable val : values) {
            sum += val.get();
        }
        this.result.set(sum);
        if (sum >= 60) {
            context.write(key, this.result);
        }
    }
}
```



# Example: topN

We want to find the top-20 used words of a text file.

## **Input Data:**

The text of the book "Flatland" By E. Abbott.

*The Project Gutenberg eBook of Flatland: A Romance of Many Dimensions (Illustrated), by Edwin A. Abbot*

*This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at [www.gutenberg.net](http://www.gutenberg.net)*

*Title: Flatland: A Romance of Many Dimensions (Illustrated)*

*Author: Edwin A. Abbot*

*Release Date: March 10, 2008 [eBook #201]*

*Language: English*

*\*\*\* START OF THIS PROJECT GUTENBERG EBOOK FLATLAND \*\*\**



# Example: topN

- Now we have to write a map reduce code to find out the top-20 used word.

word	occurrences
the	2286
of	1634
and	1098
to	1088
a	936
i	735
in	713
that	499
is	429
you	419
my	334
it	330
as	322
by	317
not	317
or	299
but	279
. . .	



# Example: topN

lets create the mapper class which would emit intermediate key value pairs as (word, 1).

```
public static class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {  
  
    private final static IntWritable one = new IntWritable(1);  
    private Text word = new Text();  
    private String tokens = "[_!$#<>\\^=\\[\\]\\\\*^\\\\\\\\,;,.\\-:()?!\\\"'"]";  
  
    @Override  
    public void map(Object key, Text value, Context context) throws  
        IOException, InterruptedException {  
        String cleanLine =  
            value.toString().toLowerCase().replaceAll(tokens, " ");  
        StringTokenizer itr = new StringTokenizer(cleanLine);  
        while (itr.hasMoreTokens()) {  
            word.set(itr.nextToken().trim());  
            context.write(word, one);  
        }  
    }  
}
```



# Example: topN

lets create the reducer that retrieves every word and puts it into a Map: if the word already exists in the map, increments its value, otherwise sets it to 1.

```
public static class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
  
    private Map<Text, IntWritable> countMap = new HashMap<Text, IntWritable>();  
  
    @Override  
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws  
        IOException, InterruptedException {  
  
        // computes the number of occurrences of a single word  
        int sum = 0;  
        for (IntWritable val : values) {  
            sum += val.get();  
        }  
  
        countMap.put(new Text(key), new IntWritable(sum));  
    }  
  
    @Override  
    protected void cleanup(Context context) throws IOException, InterruptedException {  
        ...  
    }  
}
```



# Example: topN

lets create the reducer that retrieves every word and puts it into a Map: if the word already exists in the map, increments its value, otherwise sets it to 1.

```
@Override
protected void cleanup(Context context) throws IOException, InterruptedException {

    Map<Text, IntWritable> sortedMap = sortByValues(countMap);

    int counter = 0;
    for (Text key : sortedMap.keySet()) {
        if (counter++ == 20) {
            break;
        }
        context.write(key, sortedMap.get(key));
    }
}
```



# Example: Mean

We want to find the mean max temperature for every month.

## Input Data:

<b>DDMMYYYY</b>	<b>MIN</b>	<b>MAX</b>
01012000	-4.0	5.0
02012000	-5.0	5.1
03012000	-5.0	7.7
...		
29122013	3.0	9.0
30122013	0.0	9.8
31122013	0.0	9.0



# Example: Mean

- Now we have to write a map reduce code to find out the **mean** **max** **temperature** **for** **every** **month**.

<b>MMYYYY</b>	<b>AVG</b>
022012	7.230769230769231
022013	7.2
022010	7.851851851851852
022011	9.785714285714286
032013	10.741935483870968
032010	13.133333333333333
032012	18.548387096774192
032011	13.741935483870968
. . .	



# Example: Mean

lets create the mapper class which would emit intermediate key value pairs as (month, list of max temperatures).

```
public static class MeanMapper extends Mapper<Object, Text, Text, SumCount> {

    private final int DATE = 0;
    private final int MIN = 1;
    private final int MAX = 2;

    private Map<Text, List<Double>> maxMap = new HashMap<Text, List<Double>>();

    @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {

        // gets the fields of the CSV line
        String[] values = value.toString().split(",");

        // defensive check
        if (values.length != 3) {
            return;
        }

        // gets date and max temperature
        String date = values[DATE];
        Text month = new Text(date.substring(2));
        Double max = Double.parseDouble(values[MAX]);
    }
}
```



# Example: Mean

lets create the mapper class which would emit intermediate key value pairs as (month, list of max temperatures).

```
    // if not present, put this month into the map
    if (!maxMap.containsKey(month)) {
        maxMap.put(month, new ArrayList<Double>());
    }

    // adds the max temperature for this day to the list of temperatures
    maxMap.get(month).add(max);
}

@Override
protected void cleanup(Context context) throws IOException, InterruptedException {

    --- // code

}

}
```



# Example: Mean

Let's create **SumCount** that is a *utility wrapper* around two values for computing the mean of a dataset.

The two values are:

- \* the **sum** of the values
- \* the **number of values** summed

so that **dividing** the **first** by the **second** will give us the mean.



# Example: Mean

Let's create **SumCount** that is a *utility wrapper* around two values for computing the mean of a dataset.

```
public class SumCount implements WritableComparable<SumCount> {  
  
    DoubleWritable sum;  
    IntWritable count;  
  
    public SumCount() {  
        set(new DoubleWritable(0), new IntWritable(0));  
    }  
  
    --- // code  
  
    public void addSumCount(SumCount sumCount) {  
        set(new DoubleWritable(this.sum.get() + sumCount.getSum().get()),  
            new IntWritable(this.count.get() + sumCount.getCount().get()));  
    }  
  
    --- // code  
  
}
```



# Example: Mean

lets create the mapper class which would emit intermediate key value pairs as (month, SumCount).

```
@Override
protected void cleanup(Context context) throws IOException, InterruptedException {

    // loops over the months collected in the map() method
    for (Text month: maxMap.keySet()) {

        List<Double> temperatures = maxMap.get(month);

        // computes the sum of the max temperatures for this month
        Double sum = 0d;
        for (Double max: temperatures) {
            sum += max;
        }

        // emits the month as the key and a SumCount as the value
        context.write(month, new SumCount(sum, temperatures.size()));
    }
}
```



# Example: Mean

lets create the reducer class which would emit key value pairs as (month, avg max temperatures).

```
public static class MeanReducer extends Reducer<Text, SumCount, Text, DoubleWritable> {

    private Map<Text, SumCount> sumCountMap = new HashMap<Text, SumCount>();

    @Override
    public void reduce(Text key, Iterable<SumCount> values, Context context)
        throws IOException, InterruptedException {

        SumCount totalSumCount = new SumCount();

        for (SumCount sumCount : values) {

            // sums all of them
            totalSumCount.addSumCount(sumCount);
        }

        // puts the resulting SumCount into a map
        sumCountMap.put(new Text(key), totalSumCount);
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {

        --- // code
    }
}
```



# Example: Mean

lets create the reducer class which would emit key value pairs as (month, mean).

```
@Override
protected void cleanup(Context context) throws IOException, InterruptedException {

    // loops over the months collected in the reduce() method
    for (Text month: sumCountMap.keySet()) {

        double sum = sumCountMap.get(month).getSum().get();
        int count = sumCountMap.get(month).getCount().get();

        // emits the month and the mean of the max temperatures for the month
        context.write(month, new DoubleWritable(sum/count));
    }
}
```





Hadoop 3.X more examples

Big Data - 01/04/2020