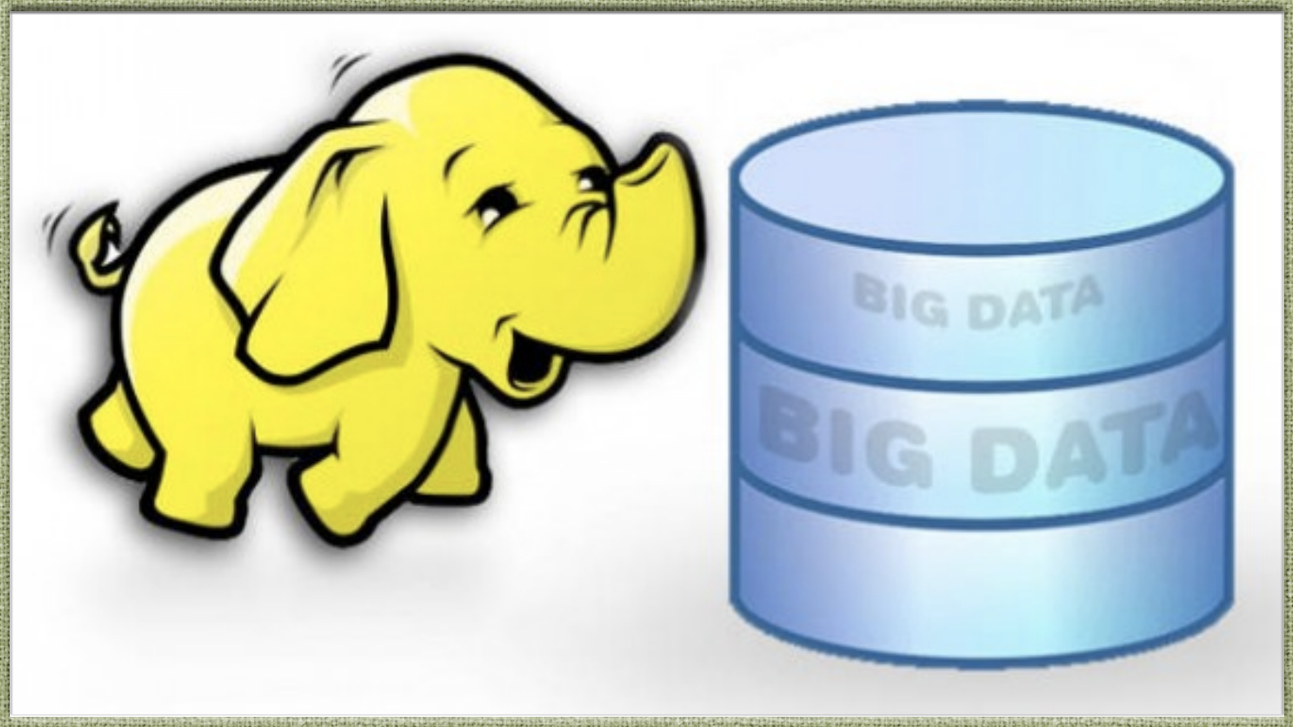
MapReduce

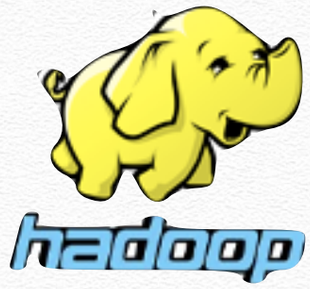


Hadoop 3 Configuration and First Examples

Big Data - 25/03/2020

Apache Hadoop & YARN

- **Apache Hadoop (1.X)**



  - De facto **Big Data open source platform**
  - Running for about 5 years in production at hundreds of companies like Yahoo, Ebay and Facebook

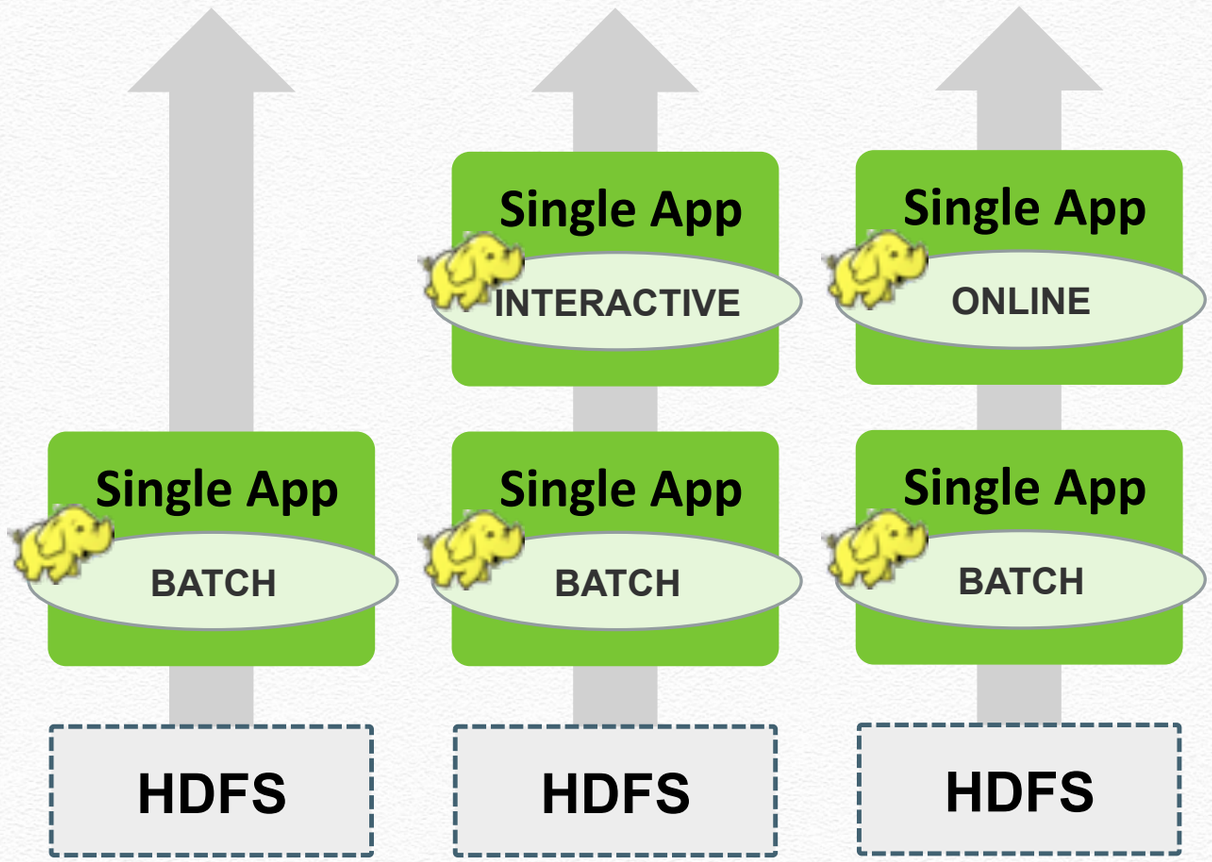
- **Hadoop 2.X**

  - Significant **improvements** in **HDFS** distributed storage layer. High Availability, NFS, Snapshots
  - **YARN** – next generation compute framework for Hadoop designed from the ground up based on experience gained from Hadoop 1
  - **YARN** running in production at Yahoo for about a year

1st Generation Hadoop: Batch Focus

HADOOP 1.0

Built for Web-Scale Batch Apps



All other usage patterns MUST leverage same infrastructure

Forces Creation of Silos to Manage Mixed Workloads

Hadoop 1 Architecture

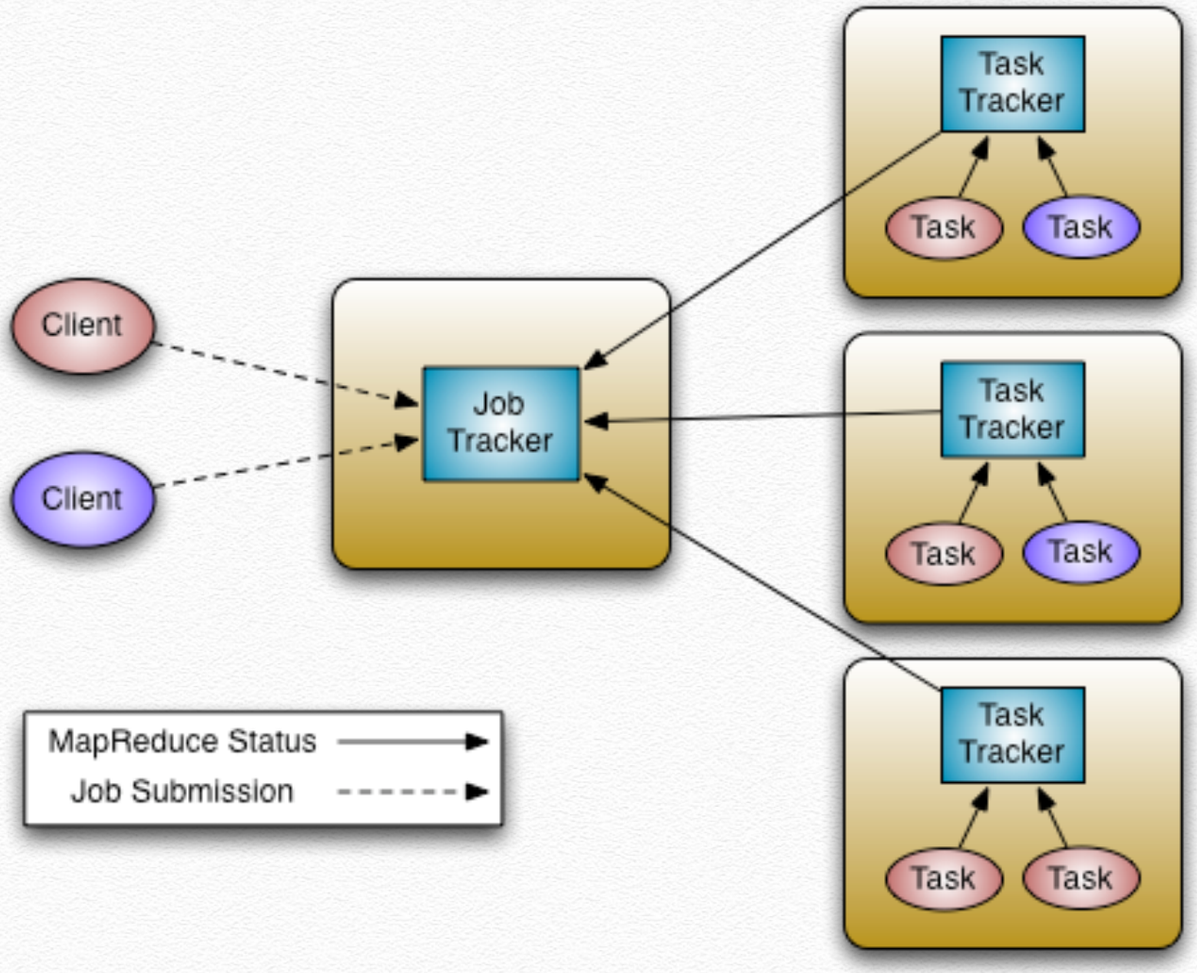
JobTracker

Manage Cluster Resources & Job Scheduling

TaskTracker

Per-node agent

Manage Tasks



Hadoop 1 Limitations

Lacks Support for Alternate Paradigms and Services

Force everything needs to look like Map Reduce

Iterative applications in MapReduce are 10x slower

Scalability

Max Cluster size ~5,000 nodes

Max concurrent tasks ~40,000

Availability

Failure Kills Queued & Running Jobs

Hard partition of resources into map and reduce slots

Non-optimal Resource Utilization

Hadoop as Next-Gen Platform

Single Use System

Batch Apps

HADOOP 1.0

MapReduce
(cluster resource management & data processing)

HDFS
(redundant, reliable storage)

Multi Purpose Platform

Batch, Interactive, Online, Streaming, …

HADOOP 2.0

MapReduce
(data processing)

Others

YARN
(cluster resource management)

HDFS2
(redundant, highly-available & reliable storage)

Hadoop 2 - YARN Architecture

ResourceManager (RM)

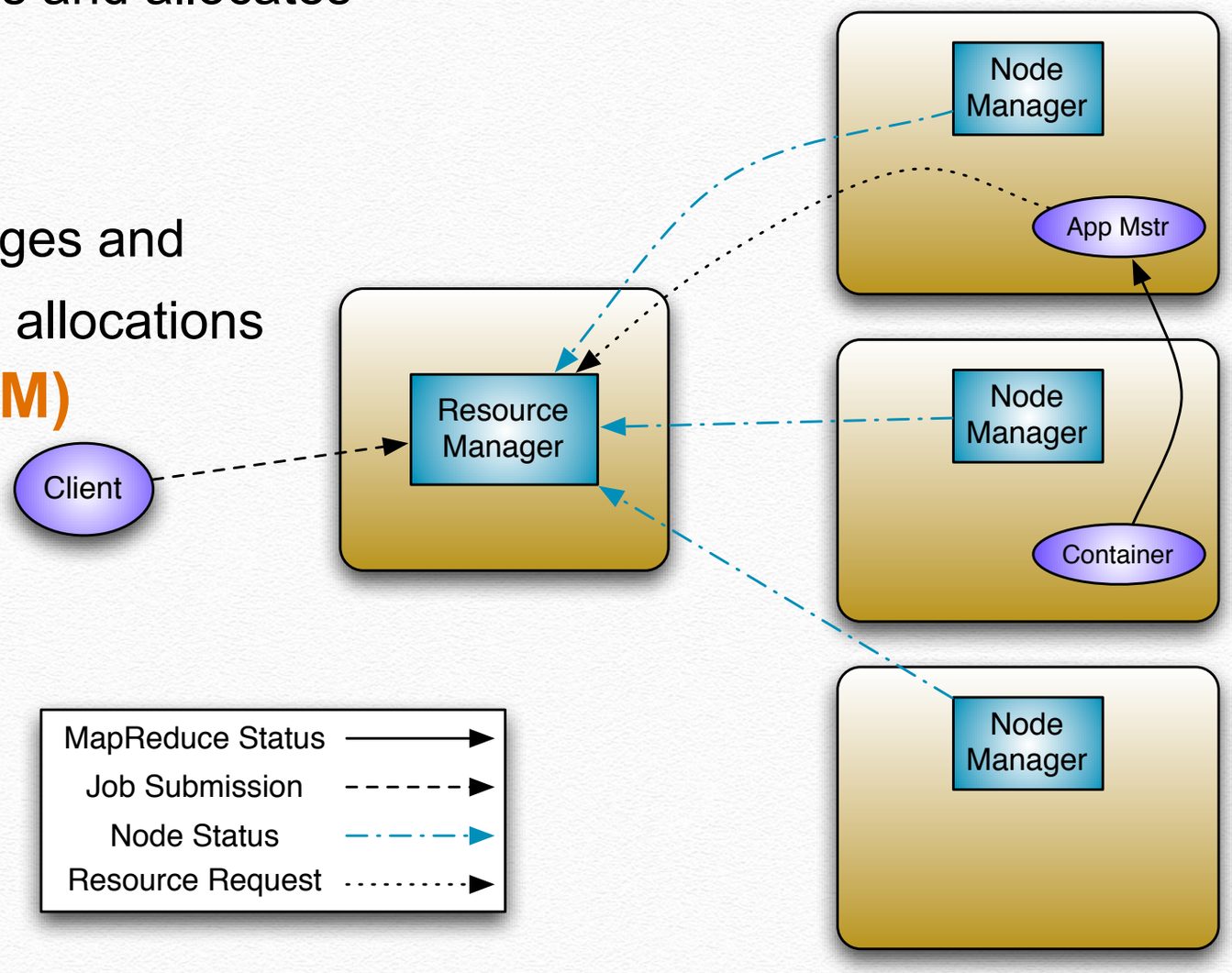
Central agent - Manages and allocates cluster resources

NodeManager (NM)

Per-Node agent - Manages and enforces node resource allocations

ApplicationMaster (AM)

Per-Application – Manages application lifecycle and task scheduling

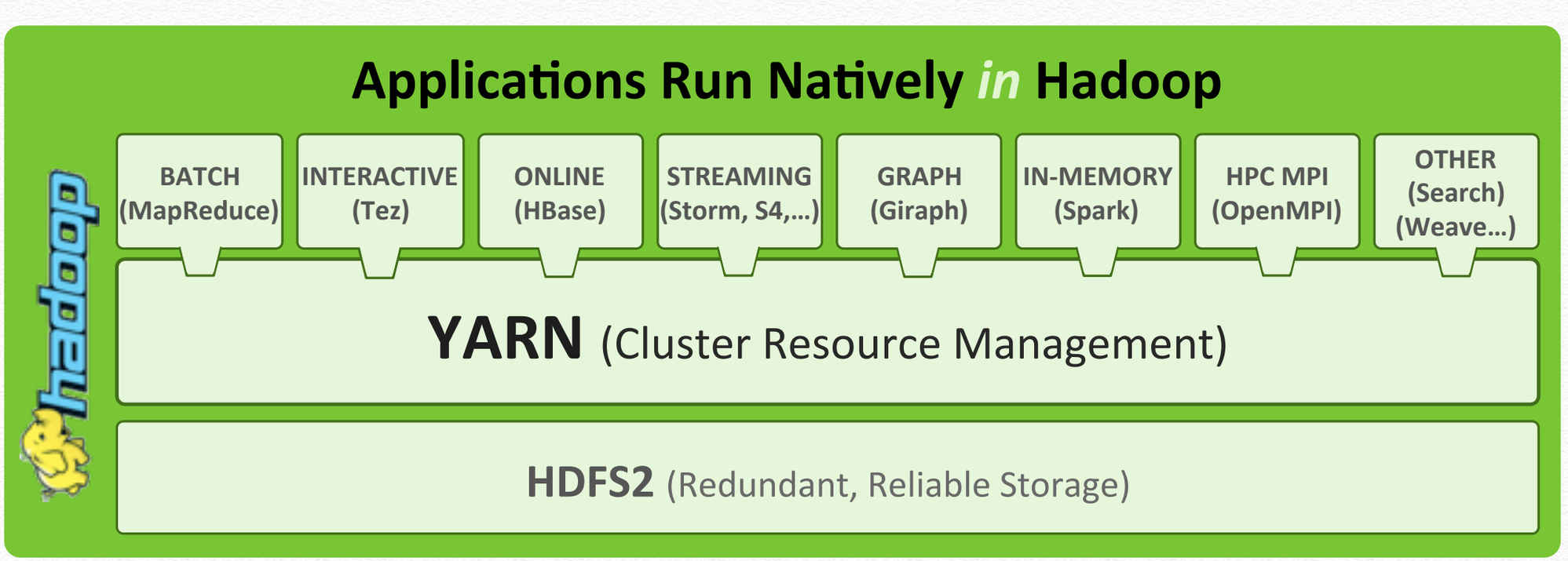


YARN: Taking Hadoop Beyond Batch

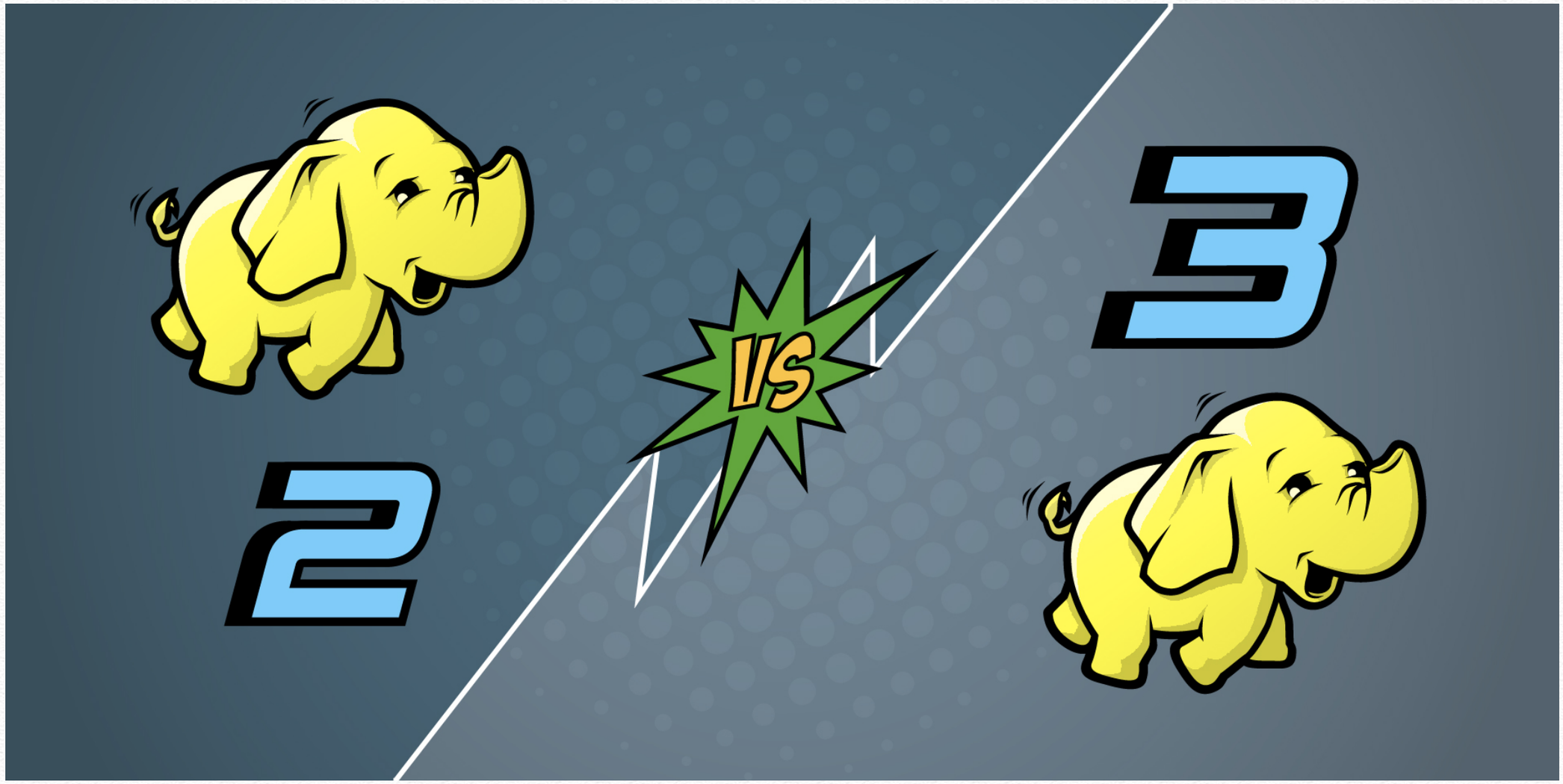
Store ALL DATA in one place…

Interact with that data in MULTIPLE WAYS

with Predictable Performance and Quality of Service



How Apache Hadoop 3 Adds Value Over Apache Hadoop 2



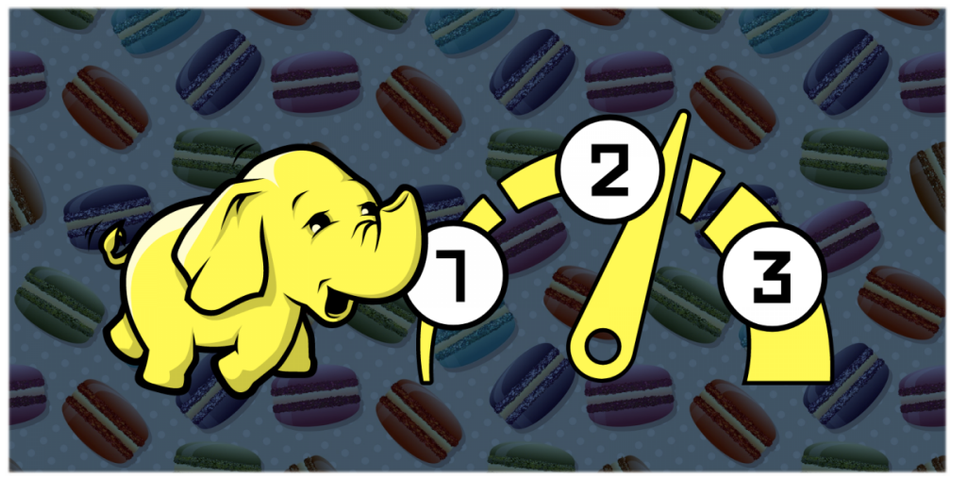
How Apache Hadoop 3 Adds Value Over Apache Hadoop 2

Attributes	Hadoop 2.x	Hadoop 3.x
Handling Fault-tolerance	Through replication	Through erasure coding
Storage	Consumes 200% in HDFS	Consumes just 50%
Scalability	Limited	Improved
File System	DFS, FTP and Amazon S3	All features plus Microsoft Azure Data Lake File System
Manual Intervention	Not needed	Not needed
Scalability	Up to 10,000 nodes in a cluster	Over 10,000 nodes in a cluster
Cluster Resource Management	Handled by YARN	Handled by YARN
Data Balancing	Uses HDFS balancer for this purpose	Uses Intra-data node balancer

How Apache Hadoop 3 Adds Value Over Apache Hadoop 2

Look at

- https://techvidvan.com/tutorials/hadoop-2-x-vs-hadoop-3-x/
- http://www.adaltas.com/en/2018/07/25/clusters-workloads-migration-hadoop-2-to-3/
- https://data-flair.training/blogs/hadoop-2-x-vs-hadoop-3-x-comparison/



Environment variables

- In the **bash_profile** export all needed **environment variables**

```
Air-di-Roberto:~ roberto$ cd
Air-di-Roberto:~ roberto$ nano .bash_profile
```

```
  GNU nano 2.0.6                File: .bash_profile

export PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
export JAVA_HOME=$(/usr/libexec/java_home)
export HADOOP_HOME=/Users/roberto/Documents/hadoop-3.2.1
export PATH=$PATH:$HADOOP_HOME/bin
```

Mac OS X

Setup passphraseless ssh

- check that you can **ssh** to the **localhost** without a **passphrase**:

```
$:~ ssh localhost
```

- If you cannot ssh to localhost without a passphrase, execute the following commands:

```
$:~ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$:~ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$:~ chmod 0600 ~/.ssh/authorized_keys
```

Setup passphraseless ssh

- Allow **remote login**

Mac OS X

Mac OS X

```
$:~ ssh-keygen -t rsa -P ""
$:~ cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

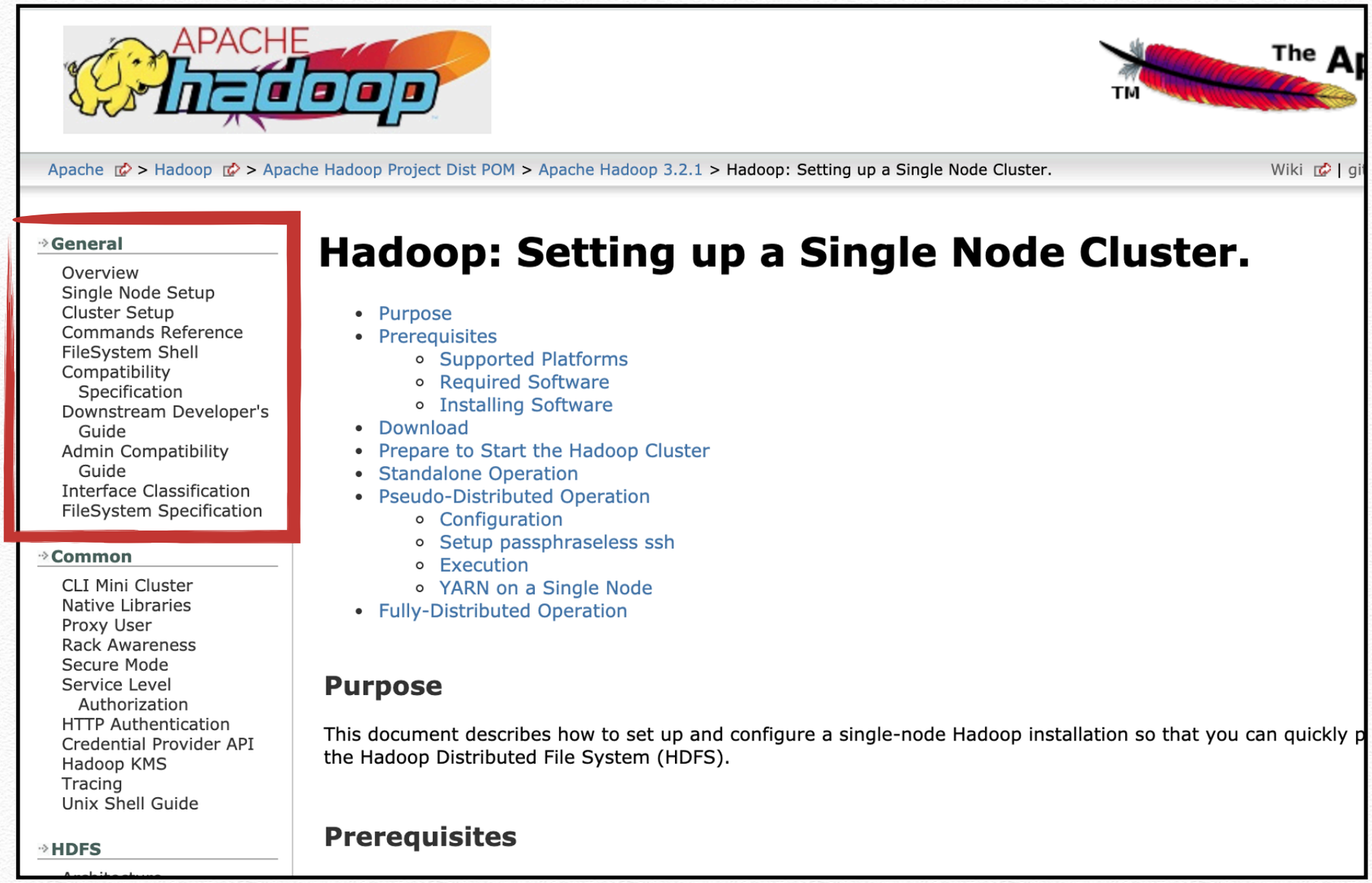
Hadoop 3 Configuration

- Download the **binary release** of **apache hadoop:**
- **hadoop-3.2.1.tar.gz**

Version	Release date	Source download	Binary download
2.10.0	2019 Oct 29	source (checksum signature)	binary (checksum signature)
3.1.3	2019 Oct 21	source (checksum signature)	binary (checksum signature)
3.2.1	2019 Sep 22	source (checksum signature)	binary (checksum signature)
3.1.2	2019 Feb 6	source (checksum signature)	binary (checksum signature)
2.9.2	2018 Nov 19	source (checksum signature)	binary (checksum signature)

Hadoop 3 Configuration

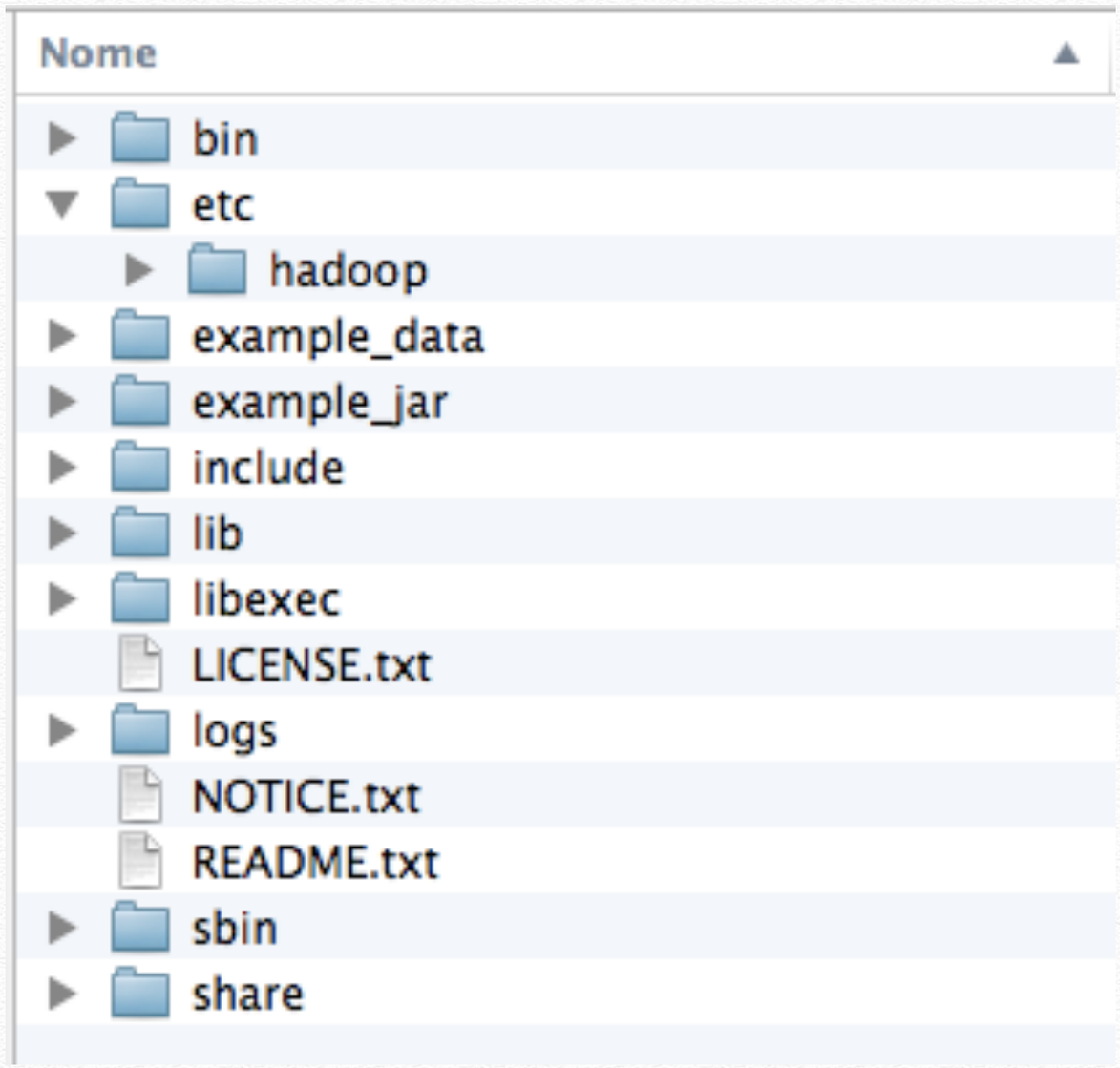
- At **https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/SingleCluster.html** you can find a **WIKI** about Hadoop 3



Hadoop 3 Configuration: Pseudo-Distributed Operation

- In the **etc/hadoop directory** of the **hadoop-home directory**, set the following files

  - **core-site.xml**
  - **hdfs-site.xml**



Hadoop 3 Configuration: Pseudo-Distributed Operation

core-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
   <property>
      <name>fs.defaultFS</name>
      <value>hdfs://localhost:9000</value>
   </property>
</configuration>
```

hdfs-site.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<configuration>
   <property>
      <name>dfs.replication</name>
      <value>1</value>
   </property>
</configuration>
```

Hadoop 3 Configuration & Running

- Final configuration:

```
$:~ hadoop-*/bin/hdfs namenode -format
```

- **Running** hadoop:

```
$:~ hadoop-*/sbin/start-dfs.sh
```

Hadoop 3 Configuration & Running

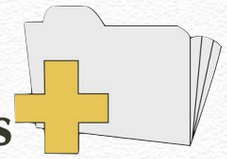
- Check all running daemons in Hadoop using the command **jps**

```
$:~ jps
```

```
20758 NameNode
20829 DataNode
20920 SecondaryNameNode
21001 Jps
```

Hadoop 3: **commands** on hdfs

```
$:~hadoop-*/bin/hdfs dfs <command> <parameters>
```

- create a directory in hdfs 

```
$:~hadoop-*/bin/hdfs dfs -mkdir input
```

- copy a local file in hdfs 

```
$:~hadoop-*/bin/hdfs dfs -put /tmp/example.txt input
```

- copy result files from hdfs to local file system

```
$:~hadoop-*/bin/hdfs dfs -get output/result localoutput
```

- delete a directory in hdfs 

```
$:~hadoop-*/bin/hdfs dfs -rm -r input
```

Hadoop 3 Configuration & Running

- Final configuration:

```
$:~ hadoop-*/bin/hdfs namenode -format
```

- **Running** hadoop:

```
$:~ hadoop-*/sbin/start-dfs.sh
```

- **Make** the HDFS **directories** required to execute **MapReduce jobs**:

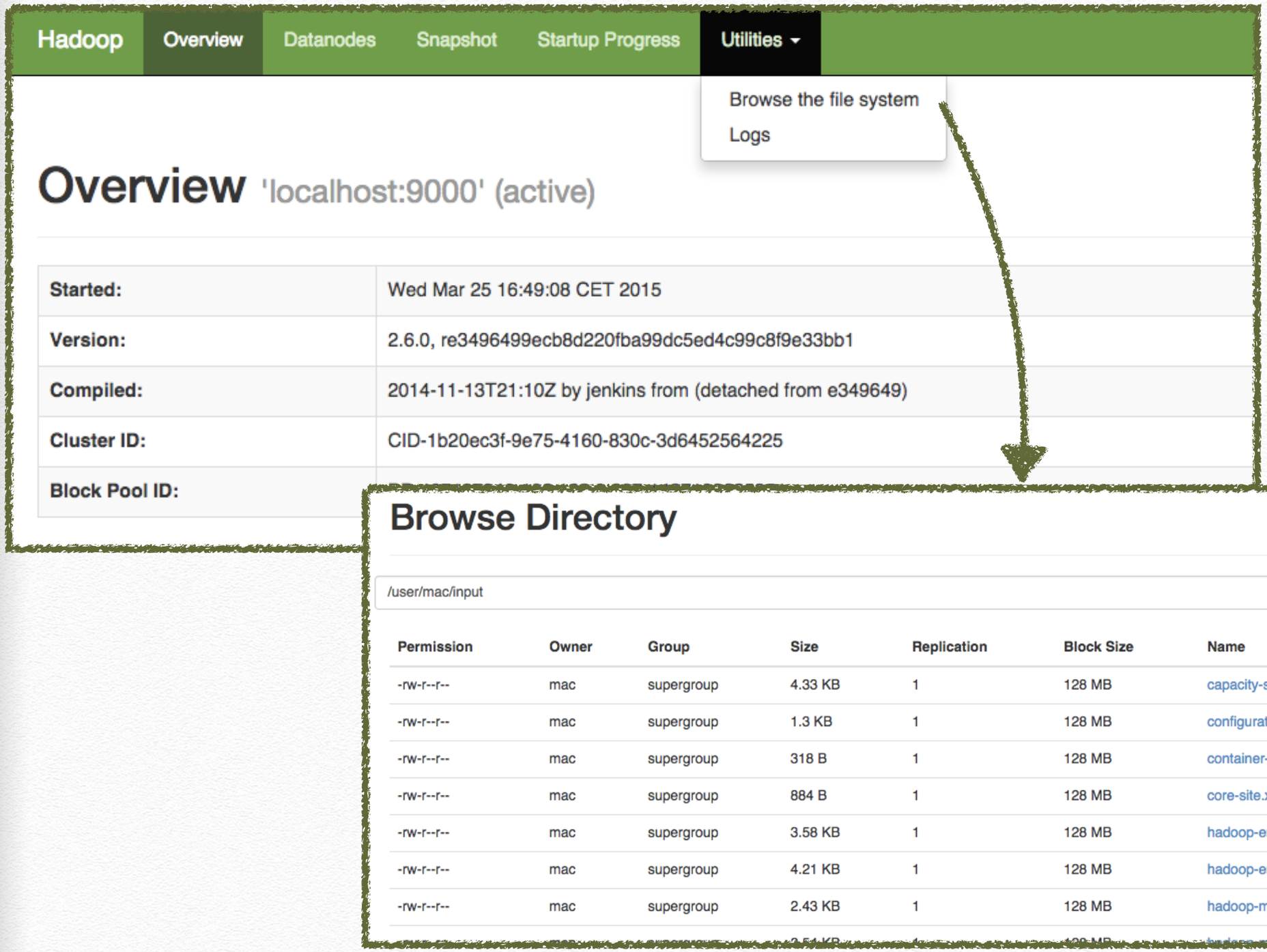
```
$:~ hadoop-*/bin/hdfs dfs -mkdir /user
$:~ hadoop-*/bin/hdfs dfs -mkdir /user/<username>
```

- **Copy** the input files into the distributed filesystem:

```
$:~ hadoop-*/bin/hdfs dfs -put etc/hadoop input
```

Hadoop3: browse **hdfs**

- **http://localhost:9870/**



Hadoop 3: **execute** *MR* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: Word Count in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/words.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/word.jar
                           WordCount input/words.txt output/result
```

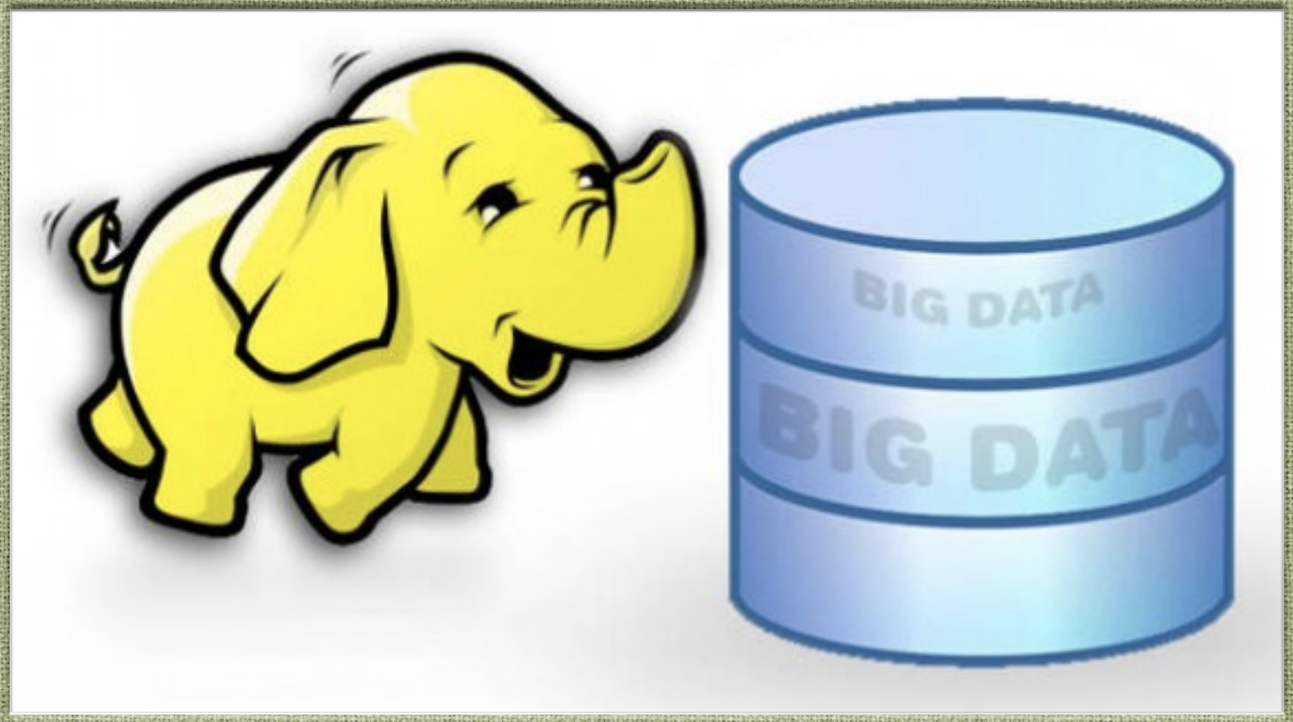
path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

https://youtu.be/BungOXP3q5Q

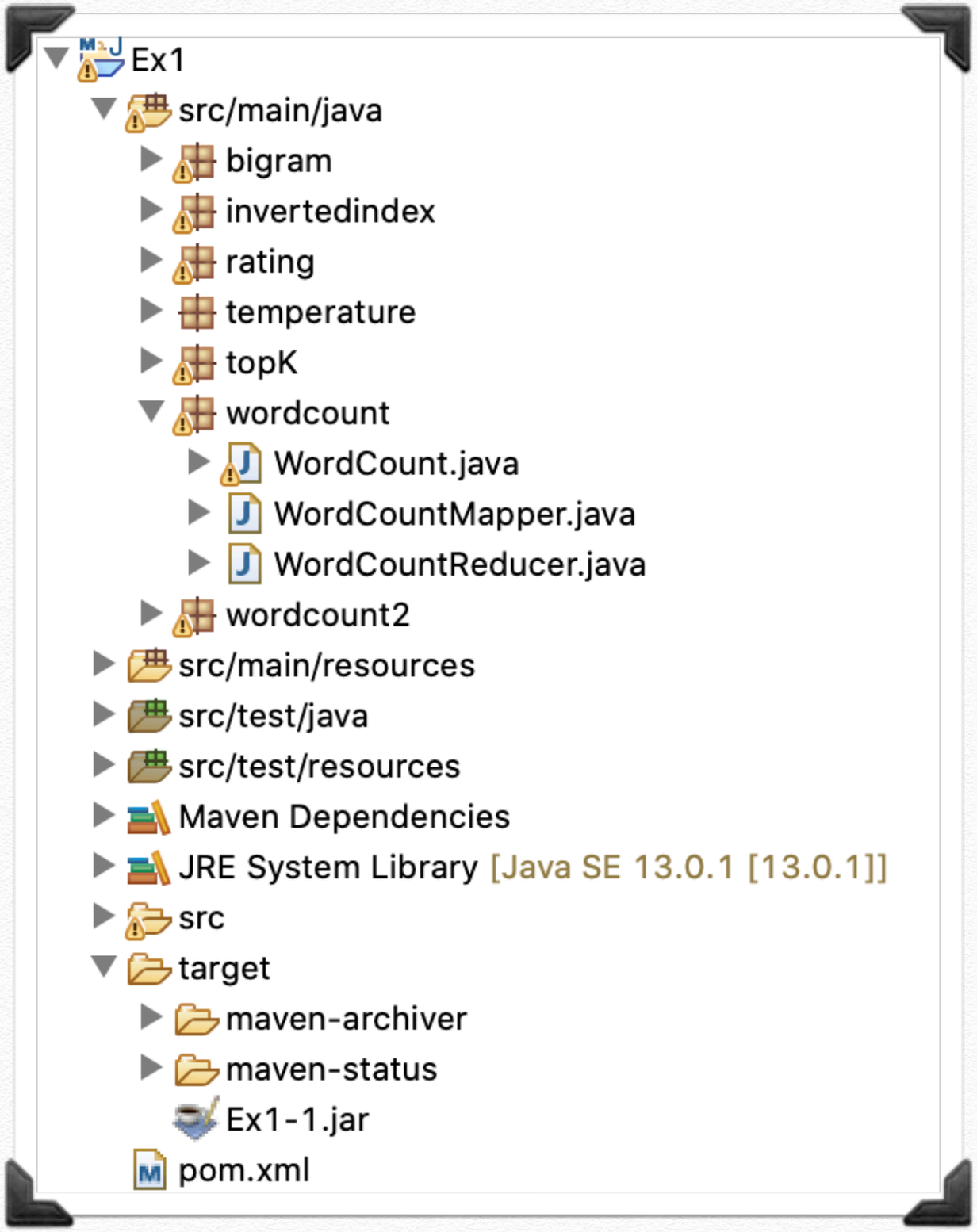
- Web video to **configure** Hadoop

MapReduce



Let's start with some examples!

Build a Maven Java Project



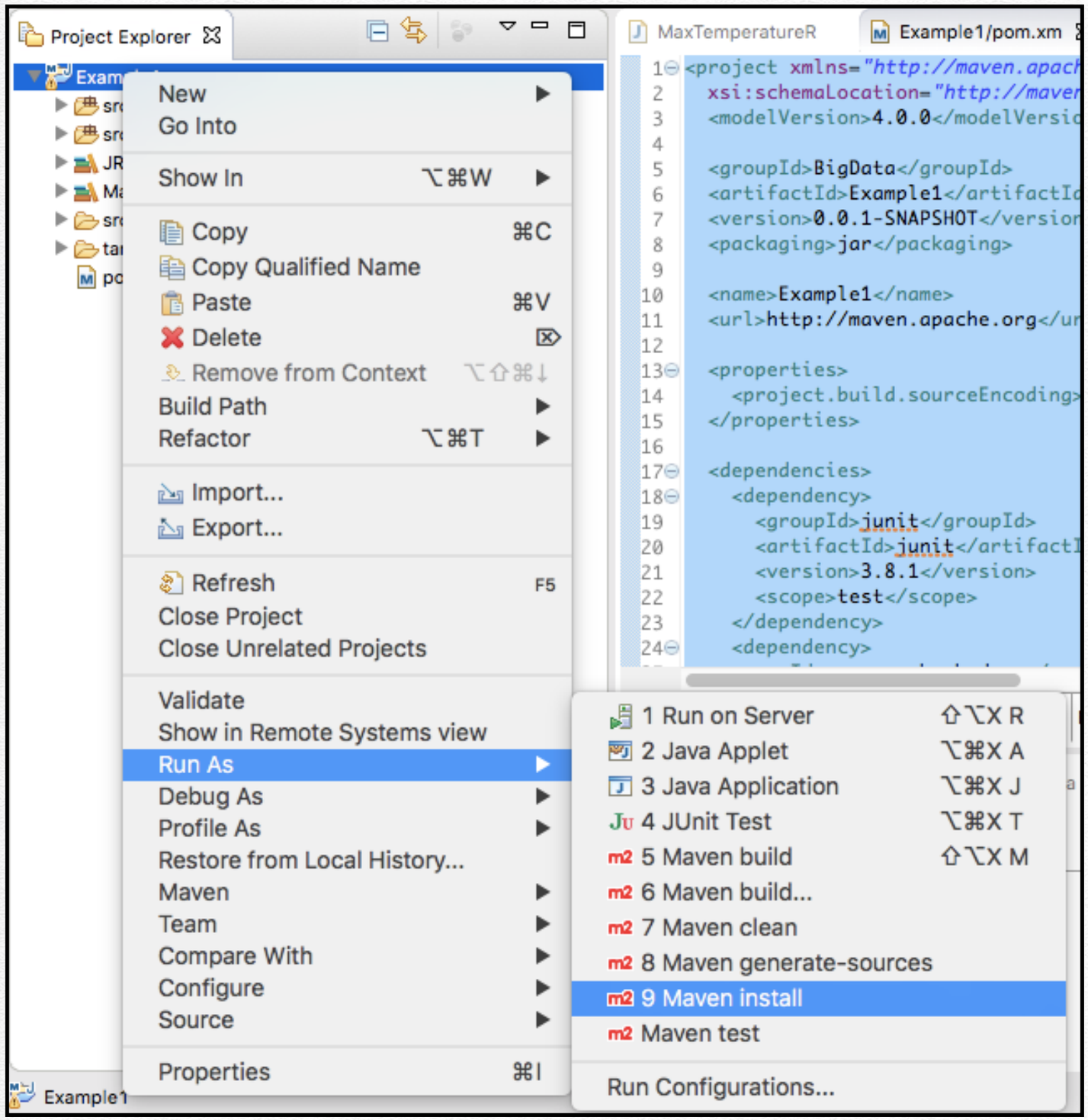
Build a Maven Java Project: .pom

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://
maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

  <groupId>BigData</groupId>
  <artifactId>Ex1</artifactId>
  <version>1</version>
  <packaging>jar</packaging>

  <name>Ex1</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>


    …
</project>
```

Build a Maven Java Project: .pom

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

    ...
<dependencies>

    ...

    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-common</artifactId>
      <version>3.2.1</version>
    </dependency>

    <dependency>
      <groupId>org.apache.hadoop</groupId>
      <artifactId>hadoop-mapreduce-client-core</artifactId>
      <version>3.2.1</version>
    </dependency>

    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.16</version>
    </dependency>
  </dependencies>

    ...
</project>
```

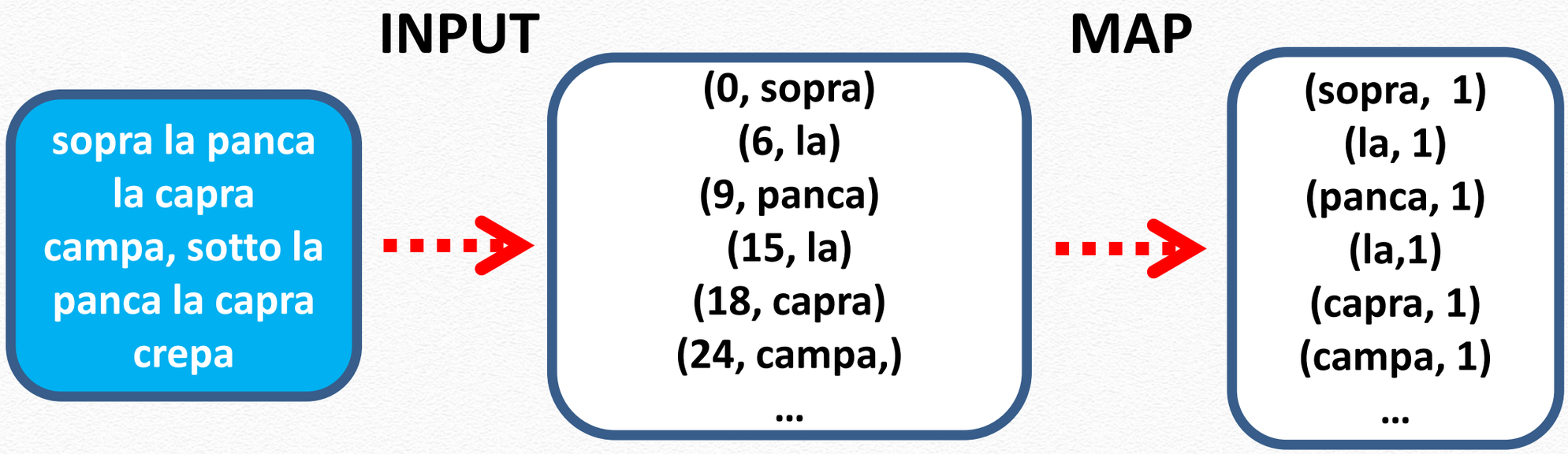
Build a Maven Java Project



https://youtu.be/M2a5gTNgeqo

- Web video to **build** a Maven Project

WordCount: logical data flow



WordCount: MAPPER

```
public class WordCountMapper extends
        Mapper<LongWritable, Text, Text, IntWritable> {

    private static final IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);

        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }

    }

}
```

WordCount: REDUCER

```
public class WordCountReducer extends
        Reducer<Text, IntWritable, Text, IntWritable> {

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

        int sum = 0;

        for (IntWritable value : values) {
            sum += value.get();
        }

        context.write(key, new IntWritable(sum));
    }
}
```

WordCount: JOB

```
public class WordCount {

    public static void main(String[] args) throws Exception {

        Job job = new Job(new Configuration(), "WordCount");

        job.setJarByClass(WordCount.class);

        job.setMapperClass(WordCountMapper.class);
        // combiner use
        // job.setCombinerClass(WordCountReducer.class);
        job.setReducerClass(WordCountReducer.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.waitForCompletion(true);
    }
}
```

Hadoop 3: **execute** *WordCount* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: WordCount in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/words.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/Example1.jar
                    wordcount/WordCount input/words.txt output/result_words
```

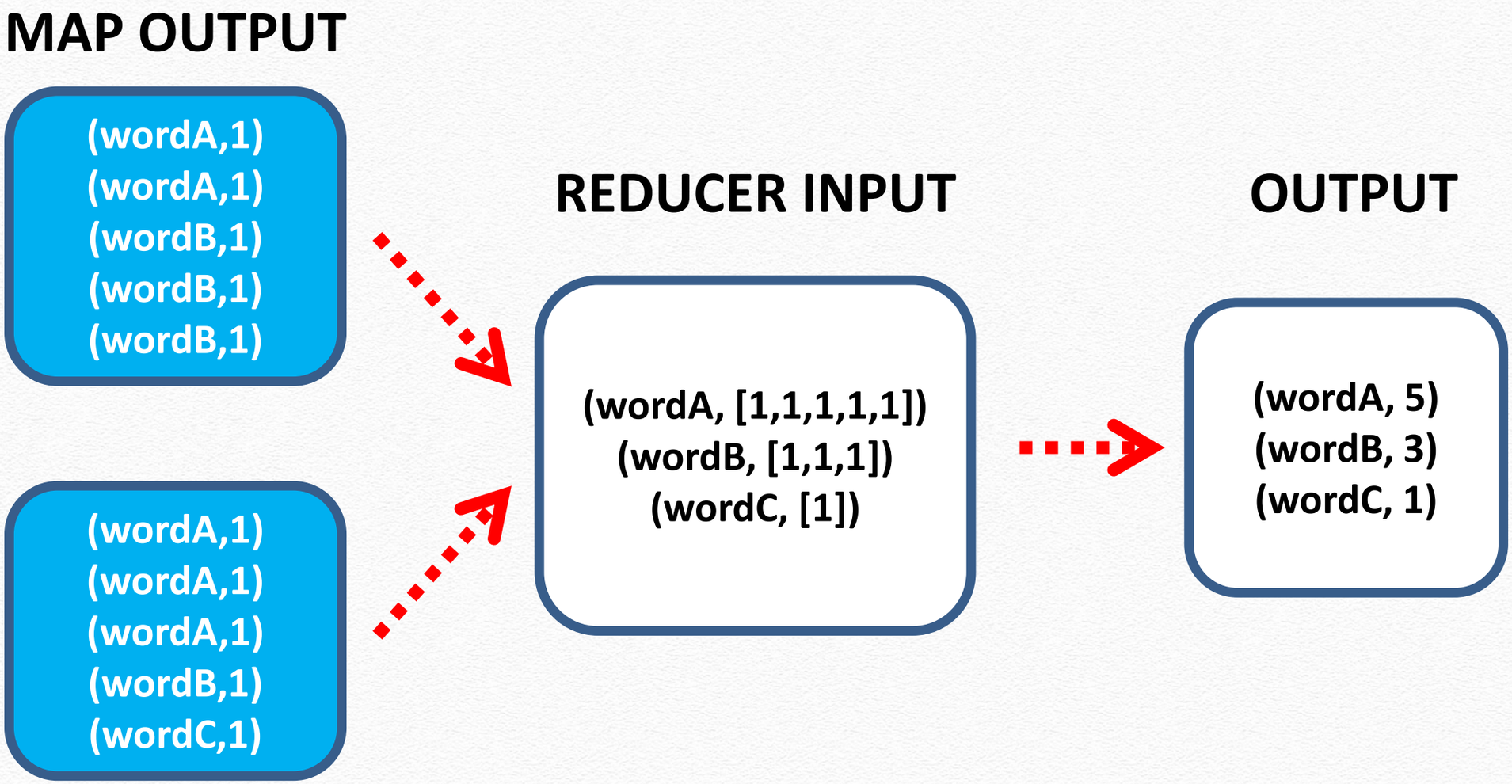
path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

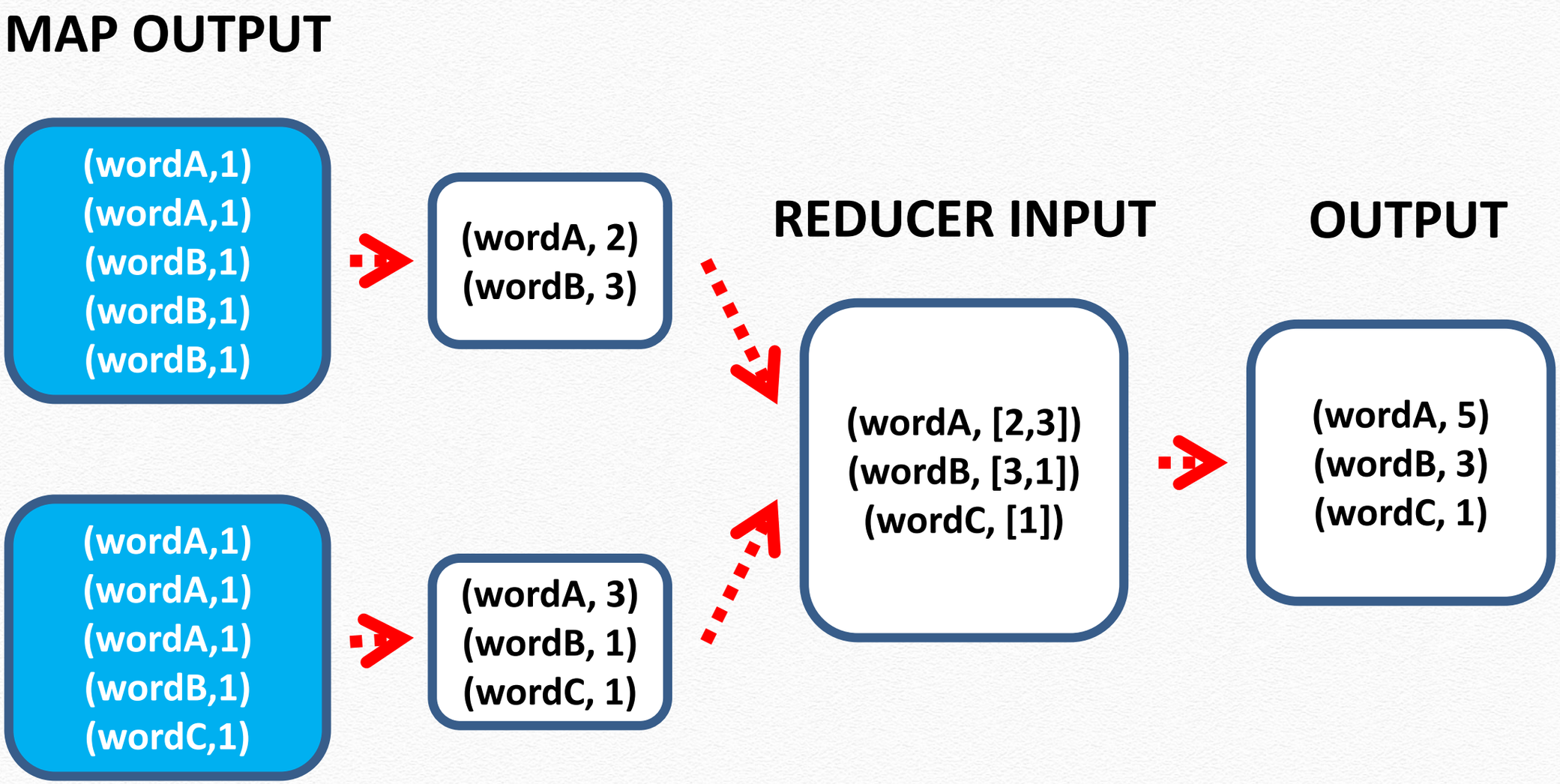
https://youtu.be/q3HflEl1bLQ

- Web video to **execute** a **Java** MapReduce job

WordCount: without Combiner



WordCount: with Combiner



WordCount: with Combiner

In the job configuration: ...

```
job.setMapperClass(WordCountMapper.class);
job.setCombinerClass(WordCountReducer.class);
job.setReducerClass(WordCountReducer.class);
...
```

Example: Temperature

We collected information from

https://www.ncdc.noaa.gov/

(National Climatic Data Center) about the atmosphere.

For each year (starting from 1763), for each month, for each day, for each hour, for each existing meteorological stations in the world we have an entry in a file that specifies:

data, *time*, *id*, *air temperature*, *pressure*, *elevation*, *latitude*, *longitude*...

Temperature: data format

…

position: *15-19*

0057332130999991958010103004+51317+028783FM-12+017199999V0203201N00721004501CN0100001N9-00211-01391102681

87-92: *position*

value 9999 for missing air temperature

char + at position 87 is optional

0057332130999991960010103004+51317+028783FM-12+017199999V0203201N00721004501CN0100001N9+00541-01391102681

…

Year

Air Temperature in Degrees Celsius x10

Example: Temperature

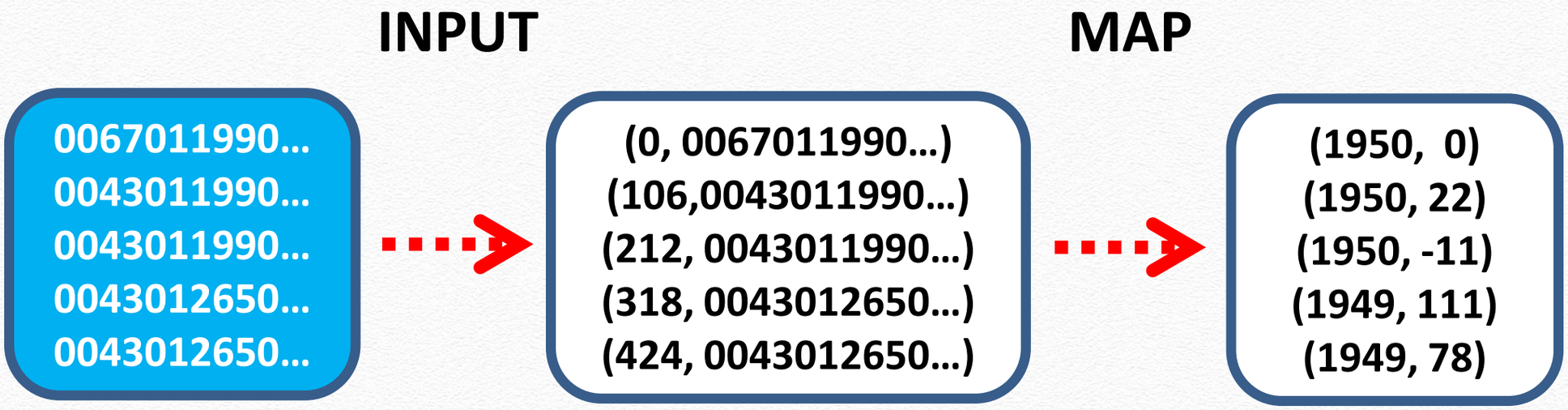
Return for each Year (*incrementally*) the max Air Temperature:

1949 -> 111

1950 -> 22

• • •

Temperature: logical data flow



Temperature: MAPPER

map (k1, v1) -> [(k2, v2)]

```
public class MaxTemperatureMapper
    extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable>
{

    private static final int MISSING = 9999;

    public void map(LongWritable key, Text value, OutputCollector<Text,
                    IntWritable> output, Reporter reporter)
            throws IOException {

        String line = value.toString();
        String year = line.substring(15, 19);

        int airTemperature;

        if (line.charAt(87) == '+') {
            airTemperature = Integer.parseInt(line.substring(88, 92));
        } else {
            airTemperature = Integer.parseInt(line.substring(87, 92));
        }

        if ( airTemperature != MISSING ) {
            output.collect(new Text(year), new IntWritable(airTemperature));
        }
    }

}
```

Temperature: REDUCER

reduce (k2, [v2]) -> [(k3, v3)]

```
public class MaxTemperatureReducer
    extends MapReduceBase implements Reducer<Text, IntWritable, Text,
                                              DoubleWritable> {

    public void reduce(Text key, Iterator<IntWritable> values,
                       OutputCollector<Text, DoubleWritable> output,
                       Reporter reporter)
            throws IOException {

        int maxValue = Integer.MIN_VALUE;

        while (values.hasNext()) {
            maxValue = Math.max(maxValue, values.next().get());
        }

        output.collect(key, new DoubleWritable(((double)maxValue)/10));
    }

}
```

Temperature: JOB

```
public class MaxTemperature {

    public static void main(String[] args) throws IOException {

        JobConf conf = new JobConf(MaxTemperature.class);
        conf.setJobName("Max temperature");

        FileInputFormat.addInputPath(conf, new Path(args[0]));
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));

        conf.setMapperClass(MaxTemperatureMapper.class);
        conf.setReducerClass(MaxTemperatureReducer.class);

        conf.setMapOutputKeyClass(Text.class);
        conf.setMapOutputValueClass(IntWritable.class);

        conf.setOutputKeyClass(Text.class);
        conf.setOutputValueClass(DoubleWritable.class);

        JobClient.runJob(conf);

    }
}
```

Hadoop 3: **execute** *MaxTemperature* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: MaxTemperature in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/temperature.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/Example1.jar
   temperature/MaxTemperature input/temperature.txt output/result_temperature
```

path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

Bigram: count

"A bigram is a couple of two consecutive words"

In this sentence there are 8 bigrams:

- *A bigram*
- *bigram is*
- *is a*
- *a couple*
- ...

Bigram: count

.

sopra la panca la capra campa, sotto la panca la capra crepa

campa, sotto 1
capra campa, 1
capra crepa 1
la capra 2
la panca 2
panca la 2
sopra la 1
sotto la 1

To represent a Bigram we want to use a custom Writable type (BigramWritable)

Bigram: count

```
public class BigramWritable implements WritableComparable<BigramWritable> {

    private Text leftBigram;
    private Text rightBigram;

    public BigramWritable() {
    }

    public BigramWritable(Text left, Text right) {
        this.leftBigram = left;
        this.rightBigram = right;
    }

    public void readFields(DataInput in) throws IOException {
        leftBigram = new Text(in.readUTF());
        rightBigram = new Text(in.readUTF());
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(leftBigram.toString());
        out.writeUTF(rightBigram.toString());
    }

    public void set(Text prev, Text cur) {
        leftBigram = prev;
        rightBigram = cur;
    }
```

Bigram: count

```
@Override
public String toString() {
    return leftBigram.toString() + " " + rightBigram.toString();
}

@Override
public int hashCode() {
    return leftBigram.hashCode() + rightBigram.hashCode();
}

@Override
public boolean equals(Object o) {
    if (o instanceof BigramWritable) {
        BigramWritable bigram = (BigramWritable) o;
        return leftBigram.equals(bigram.leftBigram)
                && rightBigram.equals(bigram.rightBigram);
    }
    return false;
}

public int compareTo(BigramWritable tp) {
    int cmp = leftBigram.compareTo(tp.leftBigram);
    if (cmp != 0) {
        return cmp;
    }
    return rightBigram.compareTo(tp.rightBigram);
}

}
```

Bigram: MAPPER

```
public class BigramCountMapper extends Mapper<LongWritable, Text, BigramWritable,
IntWritable> {
    private static final IntWritable ONE = new IntWritable(1);
    private static final BigramWritable BIGRAM = new BigramWritable();

    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String line = value.toString();
        String prev = null;
            StringTokenizer itr = new StringTokenizer(line);
            while (itr.hasMoreTokens()) {
                String cur = itr.nextToken();
            if (prev != null) {
                BIGRAM.set(new Text(prev),new Text(cur));
                context.write(BIGRAM, ONE);
            }
            prev = cur;
        }
    }
}
```

Bigram: REDUCER

```
public class BigramCountReducer extends Reducer<BigramWritable,
IntWritable, Text, IntWritable> {

    private final static IntWritable SUM = new IntWritable();

    @Override
    public void reduce(BigramWritable key, Iterable<IntWritable> values,
Context context) throws IOException, InterruptedException {
        int sum = 0;
        Iterator<IntWritable> iter = values.iterator();
        while (iter.hasNext()) {
            sum += iter.next().get();
        }
        SUM.set(sum);
        context.write(new Text(key.toString()), SUM);
    }

}
```

Bigram: JOB

```
public class BigramCount extends Configured implements Tool {

    private static final Logger LOG = Logger.getLogger(BigramCount.class);

    private BigramCount() {
    }

    private static final String INPUT = "input";
    private static final String OUTPUT = "output";
    private static final String NUM_REDUCERS = "numReducers";

    @SuppressWarnings({ "static-access" })
    public int run(String[] args) throws Exception {
        Options options = new Options();

        options.addOption(OptionBuilder.withArgName("path").hasArg()
                .withDescription("input path").create(INPUT));
        options.addOption(OptionBuilder.withArgName("path").hasArg()
                .withDescription("output path").create(OUTPUT));
        options.addOption(OptionBuilder.withArgName("num").hasArg()
                .withDescription("number of reducers").create(NUM_REDUCERS));
```

Bigram: JOB

```
CommandLine cmdline;
 CommandLineParser parser = new GnuParser();

 try {
     cmdline = parser.parse(options, args);
 } catch (ParseException exp) {
     System.err.println("Error parsing command line: "
             + exp.getMessage());
     return -1;
 }

 if (!cmdline.hasOption(INPUT) || !cmdline.hasOption(OUTPUT)) {
     System.out.println("args: " + Arrays.toString(args));
     HelpFormatter formatter = new HelpFormatter();
     formatter.setWidth(120);
     formatter.printHelp(this.getClass().getName(), options);
     ToolRunner.printGenericCommandUsage(System.out);
     return -1;
 }

 String inputPath = cmdline.getOptionValue(INPUT);
 String outputPath = cmdline.getOptionValue(OUTPUT);
 int reduceTasks = cmdline.hasOption(NUM_REDUCERS) ? Integer
         .parseInt(cmdline.getOptionValue(NUM_REDUCERS)) : 1;

 LOG.info("Tool name: " + BigramCount.class.getSimpleName());
 LOG.info(" - input path: " + inputPath);
 LOG.info(" - output path: " + outputPath);
 LOG.info(" - num reducers: " + reduceTasks);
```

Bigram: JOB

```
Job job = new Job(getConf());
        job.setJobName(BigramCount.class.getSimpleName());
        job.setJarByClass(BigramCount.class);

        job.setNumReduceTasks(reduceTasks);

        FileInputFormat.setInputPaths(job, new Path(inputPath));
        FileOutputFormat.setOutputPath(job, new Path(outputPath));

//        FileInputFormat.addInputPath(job, new Path(args[0]));
//       FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setMapOutputKeyClass(BigramWritable.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        job.setMapperClass(BigramCountMapper.class);
        job.setReducerClass(BigramCountReducer.class);

        Path outputDir = new Path(outputPath);
        FileSystem.get(getConf()).delete(outputDir, true);

        long startTime = System.currentTimeMillis();
        job.waitForCompletion(true);

        System.out.println("Job Finished in "
                + (System.currentTimeMillis() - startTime) / 1000.0
                + " seconds");

        return 0;
    }

    public static void main(String[] args) throws Exception {
        ToolRunner.run(new BigramCount(), args);
    }
}
```

Hadoop 3: **execute** *BigramCount* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: BigramCount in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/words.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/Example1.jar
                    bigram/BigramCount input/words.txt output/result_bigram 1
```

path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

Average Word Length by initial letter

sopra la panca
la capra campa,
sotto la panca la
capra crepa

INPUT

(0, **s**opra)
(6, **l**a)
(9, **p**anca)
(15, **l**a)
(18, **c**apra)
(24, **c**ampa,)
…

MAP

(**s**, 5)
(**l**, 2)
(**p**, 5)
(**l**, 2)
(**c**, 5)
(**c**, 6)
…

(**s**, 5)
(**l**, 2)
(**p**, 5)
(**l**, 2)
(**c**, 5)
(**c**, 5)
…

SHUFFLE & SORT

(**c**, [5,6])
(**l**, [2,2])
(**p**, [5])
(**s**, [5])
…

REDUCE

(**c**, 5.25)
(**l**, 2)
(**p**, 5)
(**s**, 5)
…

Hadoop 3: **execute** *AverageWordLength* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: AverageWordLength in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/words.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/Example1.jar
        avgword/AverageWordLength input/words.txt output/result_avg_word
```

path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

Inverted Index

there is a tab (\t)

```
1  if you prick us do we not bleed,
2  if you tickle us do we not laugh,
3  if you poison us do we not die and,
4  if you wrong us shall we not revenge
```

```
and,     3
bleed,   1
die      3
do       1,2,3
if       1,4,3,2
laugh,   2
not      3,4,1,2
poison   3
prick    1
revenge  4
shall    4
tickle   2
us       2,4,3,1
we       2,4,1,3
wrong    4
you      1,3,4,2
```

Inverted Index

1 if you prick us do we not bleed,
2 if you tickle us do we not laugh,
3 if you poison us do we not die and,
4 if you wrong us shall we not revenge

(**1**, if you prick …)
(**2**, if you tickle …)
(**3**, if you poison …)
(**4**, if you wrong …)

(**if**, 1)
(**you**, 1)
(**prick**, 1)
…
(**if**, 2)
(**you**, 2)
(**tickle**, 2)
…

(**if**, [1,2,…])
(**you**, [1,2,…])
(**prick**, [1])
(**tickle**, [2])
…

Hadoop 3: **execute** *InvertedIndex* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: InvertedIndex in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/words2.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/Example.jar
          invertedindex/InvertedIndex input/words2.txt output/result_ii
```

path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

Top K records (citations)

A,B
C,B
F,B
A,A
C,A
R,F
R,C
B,B

(**B**, 4)
(**A**, 2)
(**F**, 1)
(**C**, 1)

The objective is to find the **top-10** most frequently cited patents in descending order.

The format of the input data is **CITING_PATENT,CITED_PATENT**.

Top K records (citations)

A,B
C,B
F,B
A,A
C,A
R,F
R,C
B,B

MAP1

(CITEDPATENT, 1)

(**B**, 1)
(**B**, 1)
(**B**, 1)
(**A**, 1)
(**A**, 1)
(**F**, 1)
…

REDUCE1

(CITEDPATENT, N)

(**B**, 4)
(**A**, 2)
(**F**, 1)
…

(CITEDPATENT, N)

(**B**, 4)
(**A**, 2)
(**F**, 1)
…

MAP2

COPY

(CITEDPATENT, N)

(**B**, 4)
(**A**, 2)
(**F**, 1)
…

REDUCE2

FILTER

(CITEDPATENT, N)

(**B**, 4)
(**A**, 2)
(**F**, 1)
…

Hadoop 3: **execute** *TopKRecords* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

- **Example: TopKRecords in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/citations.txt input

$:~hadoop-*/bin/hadoop jar /example_jar/Example1.jar
                    topK/TopKRecords input/citations.txt output/result_topk
```

path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

Hadoop 3: **clean** *DisjointSelection* application

```
$:~hadoop-*/bin/hadoop jar <path-jar> <jar-MainClass> <jar-parameters>
```

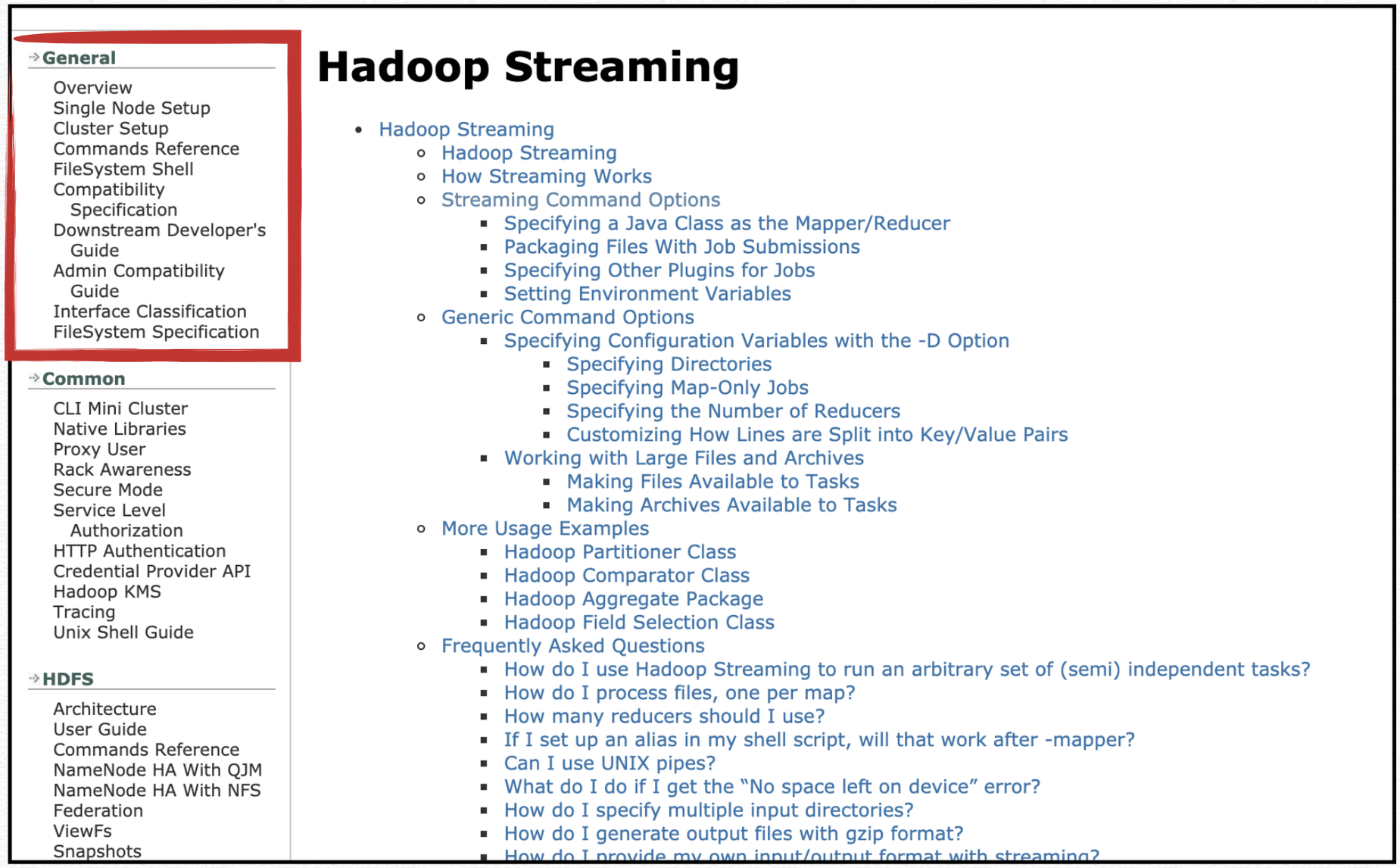
- **Example: DisjointSelection in MapReduce**

```
$:~hadoop-*/bin/hdfs dfs -rm -r output
```

Build a Python Project

Hadoop 3 streaming

- At **https://hadoop.apache.org/docs/current/hadoop-streaming/HadoopStreaming.html#Streaming_Command_Options** you can find a **WIKI** about Hadoop 3 streaming



General

- Overview
- Single Node Setup
- Cluster Setup
- Commands Reference
- FileSystem Shell
- Compatibility Specification
- Downstream Developer's Guide
- Admin Compatibility Guide
- Interface Classification
- FileSystem Specification

Common

- CLI Mini Cluster
- Native Libraries
- Proxy User
- Rack Awareness
- Secure Mode
- Service Level Authorization
- HTTP Authentication
- Credential Provider API
- Hadoop KMS
- Tracing
- Unix Shell Guide

HDFS

- Architecture
- User Guide
- Commands Reference
- NameNode HA With QJM
- NameNode HA With NFS
- Federation
- ViewFs
- Snapshots

Hadoop Streaming

- Hadoop Streaming
  - Hadoop Streaming
  - How Streaming Works
  - Streaming Command Options
    - Specifying a Java Class as the Mapper/Reducer
    - Packaging Files With Job Submissions
    - Specifying Other Plugins for Jobs
    - Setting Environment Variables
  - Generic Command Options
    - Specifying Configuration Variables with the -D Option
      - Specifying Directories
      - Specifying Map-Only Jobs
      - Specifying the Number of Reducers
      - Customizing How Lines are Split into Key/Value Pairs
    - Working with Large Files and Archives
      - Making Files Available to Tasks
      - Making Archives Available to Tasks
  - More Usage Examples
    - Hadoop Partitioner Class
    - Hadoop Comparator Class
    - Hadoop Aggregate Package
    - Hadoop Field Selection Class
  - Frequently Asked Questions
    - How do I use Hadoop Streaming to run an arbitrary set of (semi) independent tasks?
    - How do I process files, one per map?
    - How many reducers should I use?
    - If I set up an alias in my shell script, will that work after -mapper?
    - Can I use UNIX pipes?
    - What do I do if I get the "No space left on device" error?
    - How do I specify multiple input directories?
    - How do I generate output files with gzip format?
    - How do I provide my own input/output format with streaming?

Hadoop 3 streaming

- Download **hadoop-streaming-3.2.1.jar** at **https://jar-download.com/artifacts/org.apache.hadoop/hadoop-streaming/3.2.1/source-code**

Download hadoop-streaming JAR 3.2.1 with all dependencies

These are the files of the artifact **hadoop-streaming** version **3.2.1** from the group **org.apache.hadoop**.

Apache Hadoop MapReduce Streaming

Download hadoop-streaming (3.2.1)

Artifact hadoop-streaming
Group org.apache.hadoop
Version 3.2.1
Last update 10. September 2019
Newest version No
Tags: streaming hadoop apache mapreduce
Organization not specified
URL Not specified
License not specified
Dependencies amount 0
Dependencies No dependencies
There are maybe transitive dependencies!

The newest version!

Maven | Gradle | Ivy | SBT

```
<dependency>
    <groupId>org.apache.hadoop</groupId>
    <artifactId>hadoop-streaming</artifactId>
    <version>3.2.1</version>
</dependency>
<!-- Thanks for using https://jar-download.com -->
```

Show more of this group | Show more artifacts with this name
Show all versions of hadoop-streaming | Show documentation
Add to **Project**

Python: WordCount Mapper

- **mapper.py**

```
#!/usr/bin/env python
"""mapper.py"""

import sys

# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # write the results to STDOUT (standard output);
        # what we output here will be the input for the
        # Reduce step, i.e. the input for reducer.py
        #
        # tab-delimited; the trivial word count is 1
        print '%s\t%s' % (word, 1)
```

Python: WordCount Reducer

- reducer**.py**

```python
#!/usr/bin/env python
"""reducer.py"""

from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

# input comes from STDIN
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()

    # parse the input we got from mapper.py
    word, count = line.split('\t', 1)

    # convert count (currently a string) to int
    try:
        count = int(count)
    except ValueError:
        # count was not a number, so silently
        # ignore/discard this line
        continue

    # this IF-switch only works because Hadoop sorts map output
    # by key (here: word) before it is passed to the reducer
    if current_word == word:
        current_count += count
    else:
        if current_word:
            # write result to STDOUT
            print '%s\t%s' % (current_word, current_count)
        current_count = count
        current_word = word

# do not forget to output the last word if needed!
if current_word == word:
    print '%s\t%s' % (current_word, current_count)
```

Hadoop 3: **execute** *WordCount* application

```
$:~hadoop-*/bin/hadoop jar <path-streaming-jar> <jar-parameters>
```

- **Example: (Python) WordCount in MapReduce**

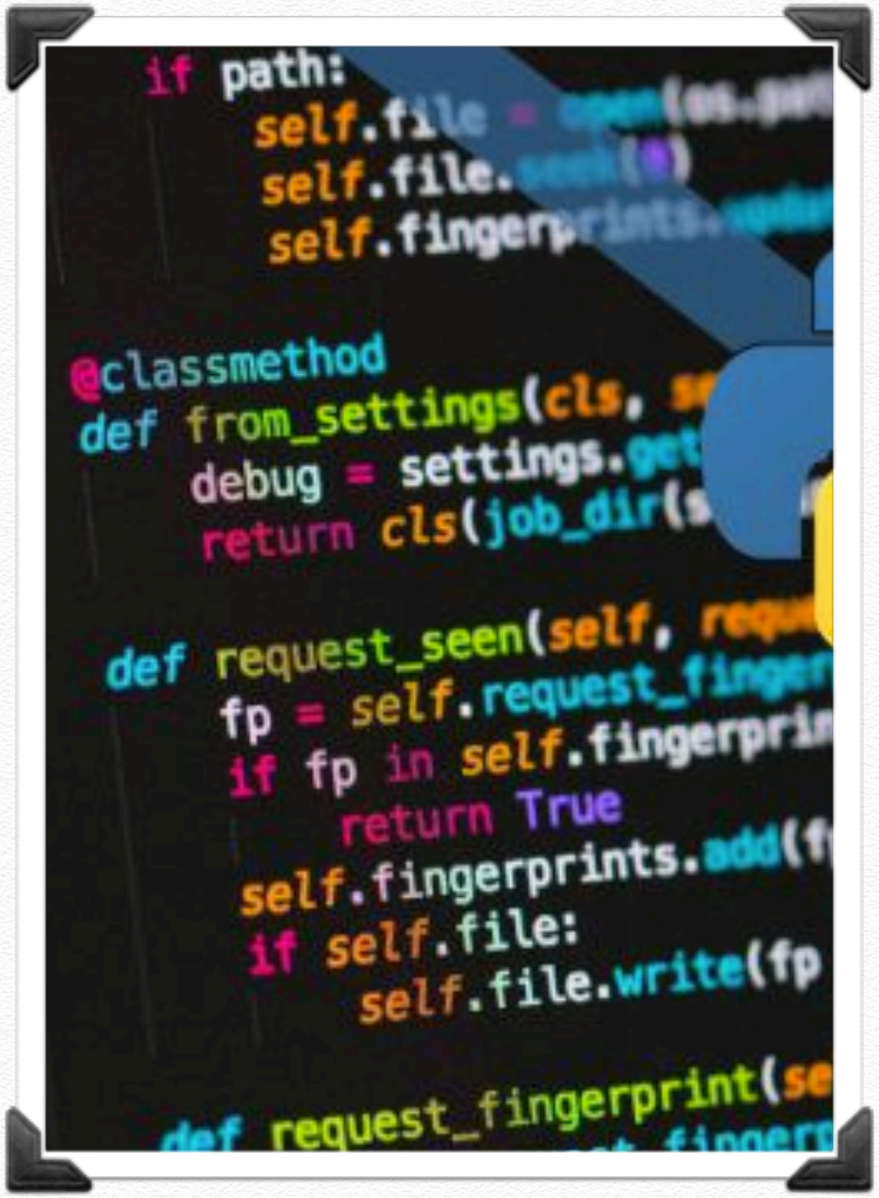
```
$:~hadoop-*/bin/hdfs dfs -mkdir output

$:~hadoop-*/bin/hdfs dfs -put /example_data/words.txt input

$:~hadoop-*/bin/hadoop jar hadoop-streaming-3.2.1.jar
                    -mapper mapper.py -reducer reducer.py
                      -input  /user/roberto/input/words.txt
                      -output /user/roberto/output/result_words
```

path on hdfs to reach the file

path on hdfs of the directory to generate to store the result

https://youtu.be/dFM-aSjJ86E

- Web video to **execute** a **Python** MapReduce job



Stopping the Hadoop 3 DFS

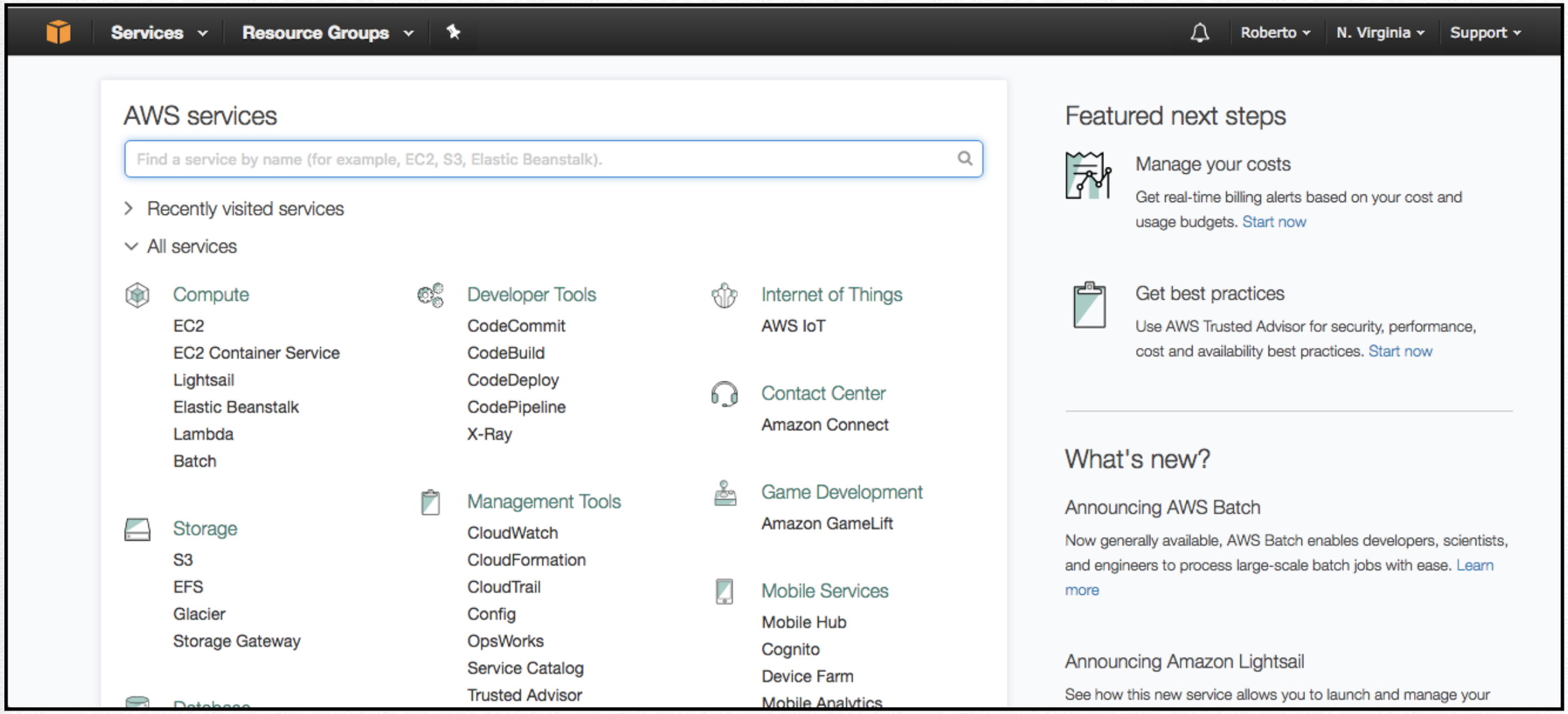
- **Stop** hadoop:

```
$:~ hadoop-*/sbin/stop-dfs.sh
```

That's all! Happy Map Reducing!

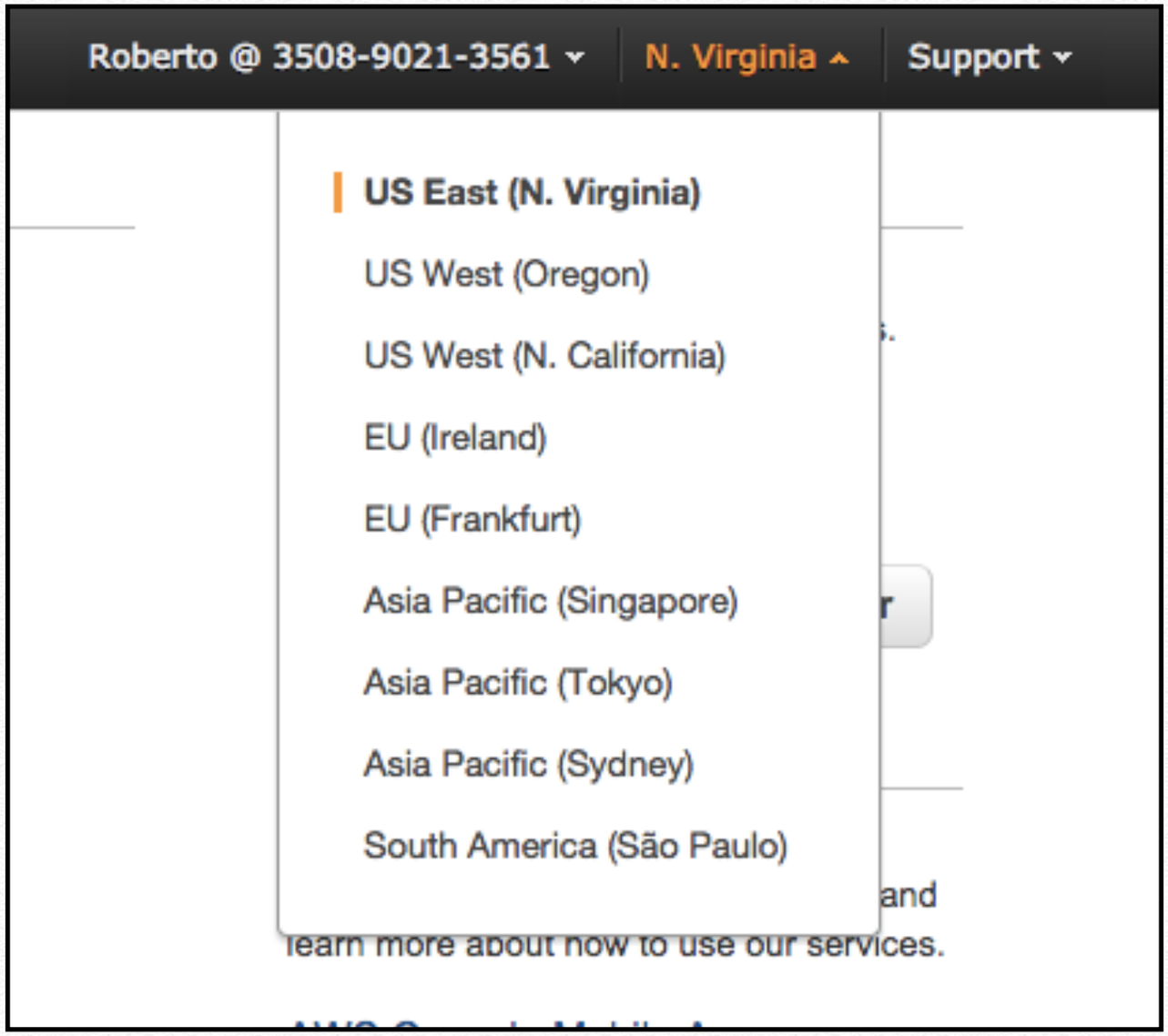
Hadoop 3 on AMAZON

Hadoop 3 on AMAZON



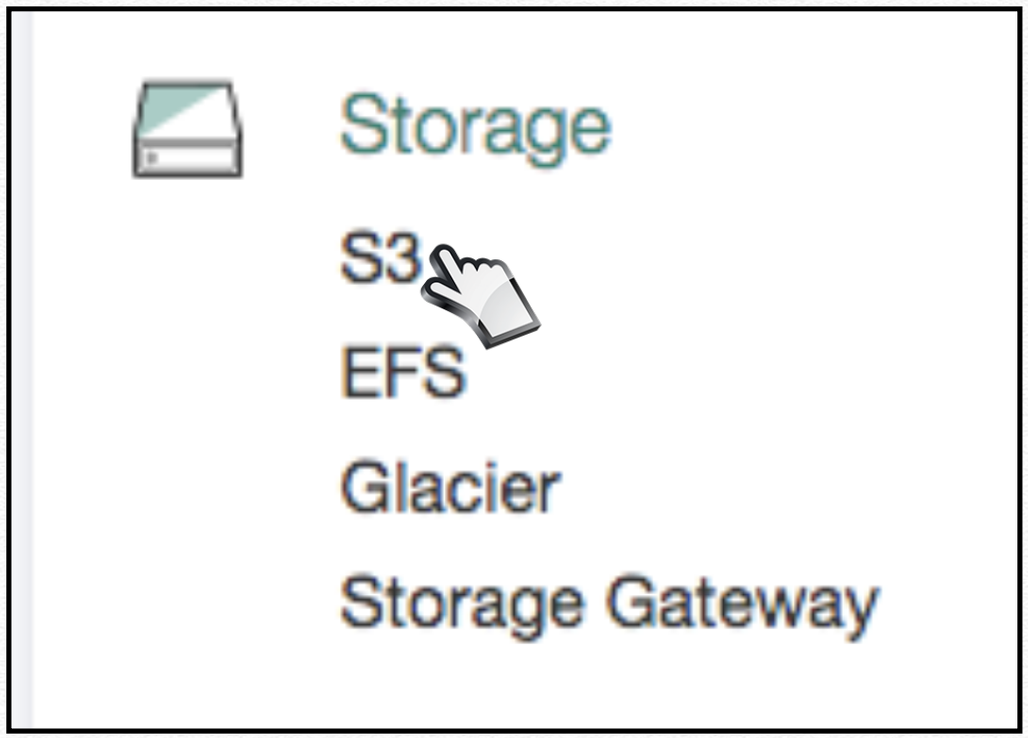
Hadoop 3 on AMAZON

Regions



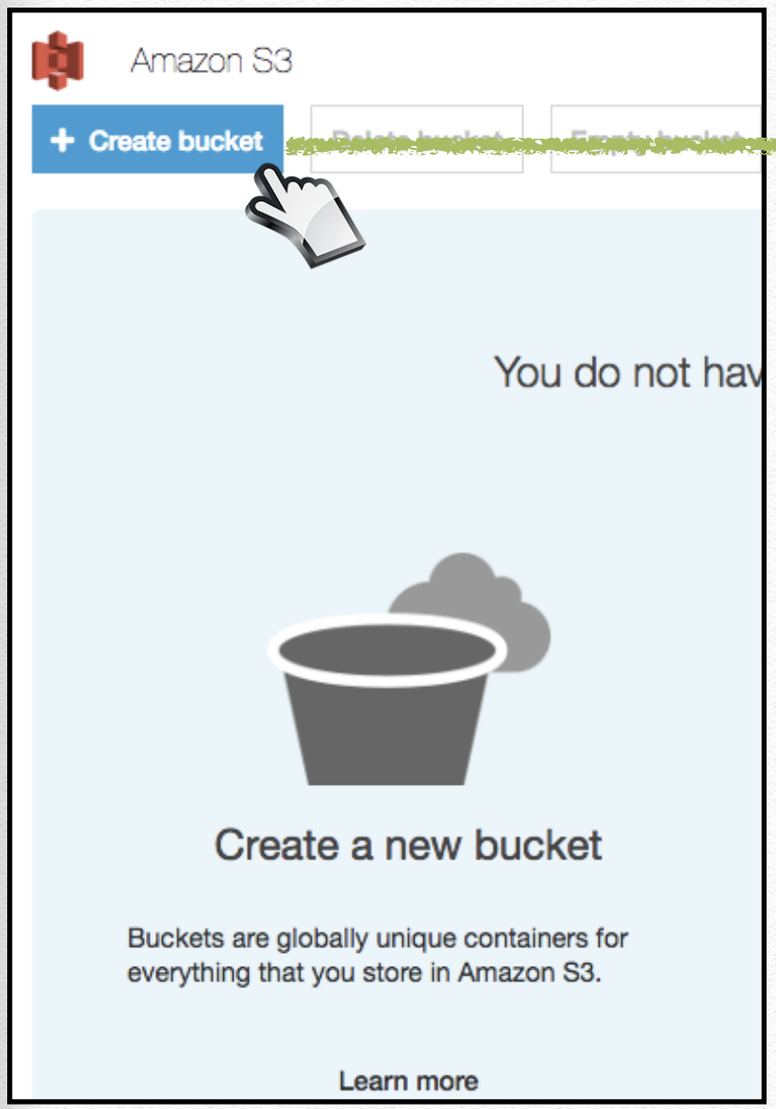
Hadoop 3 on AMAZON

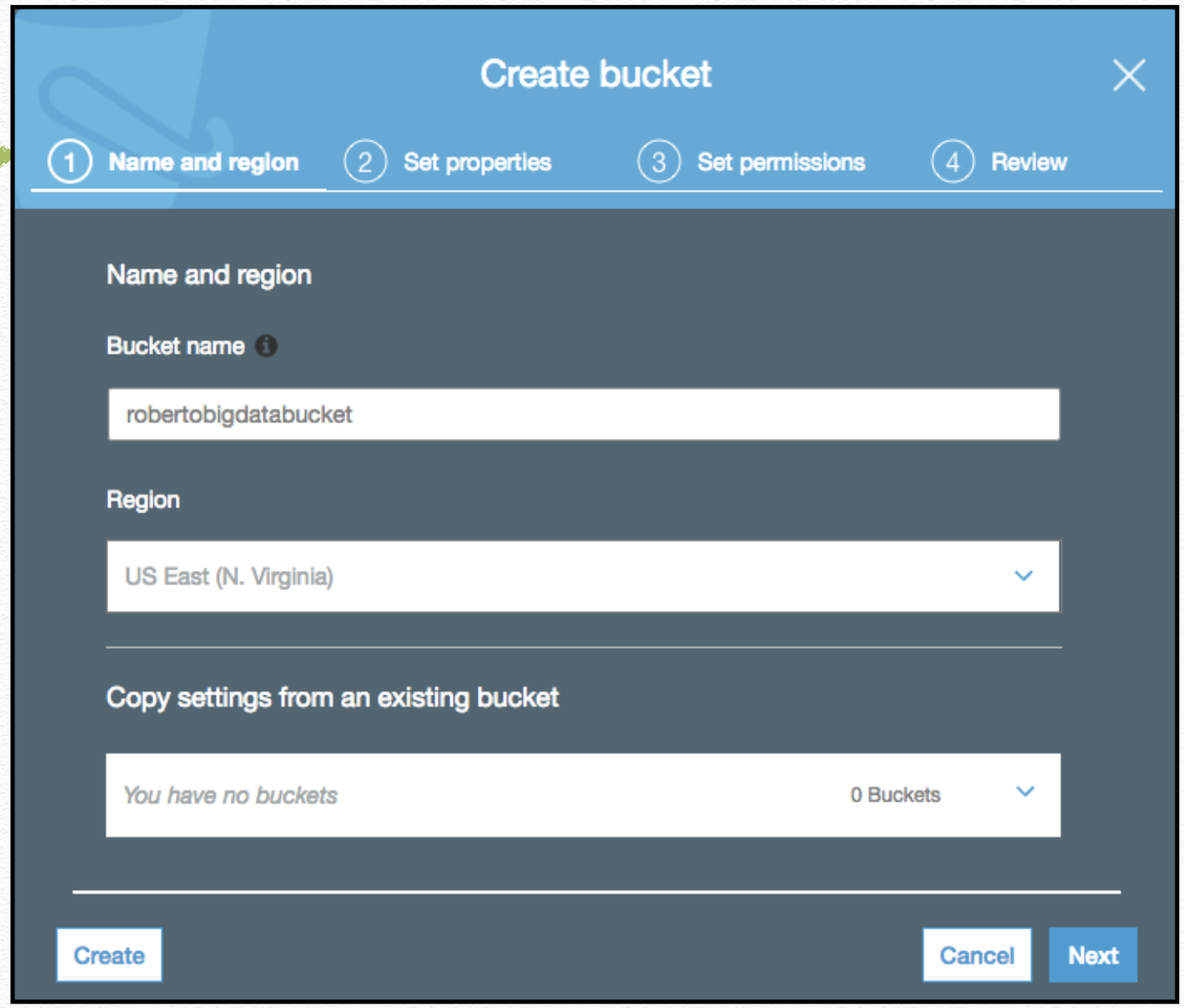
S3 and buckets



Hadoop 3 on AMAZON

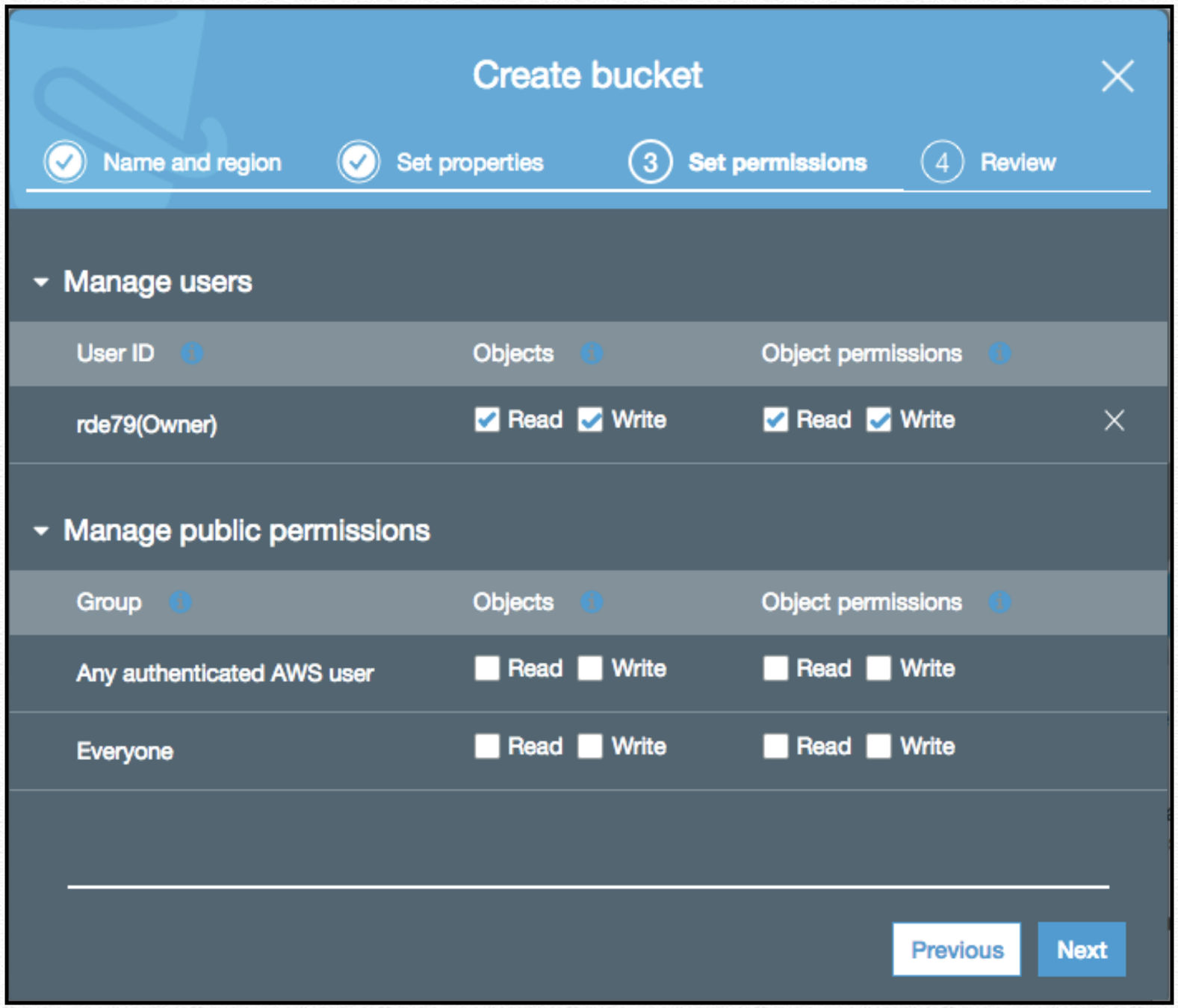
S3 and buckets





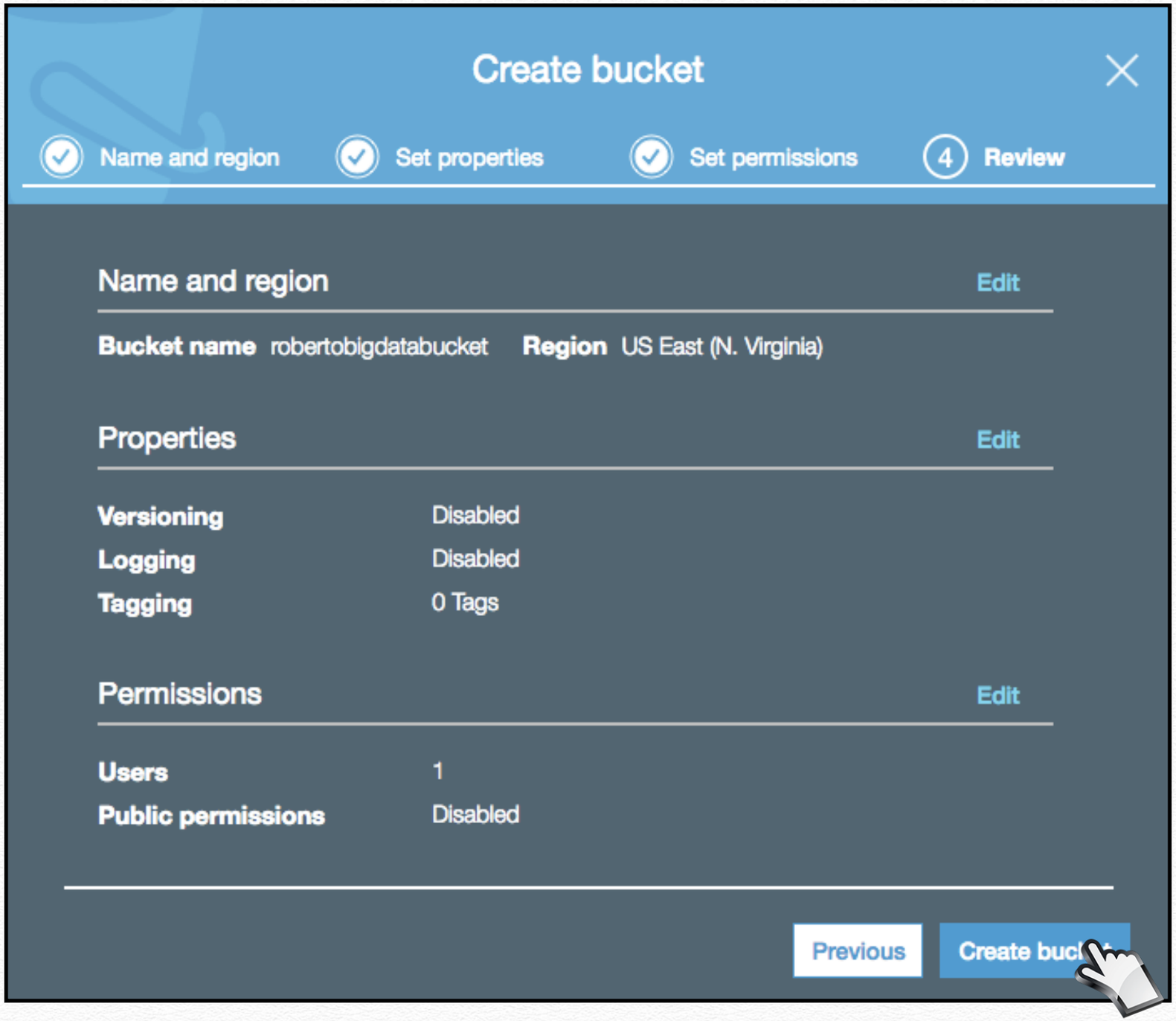
Hadoop 3 on AMAZON

S3 and buckets



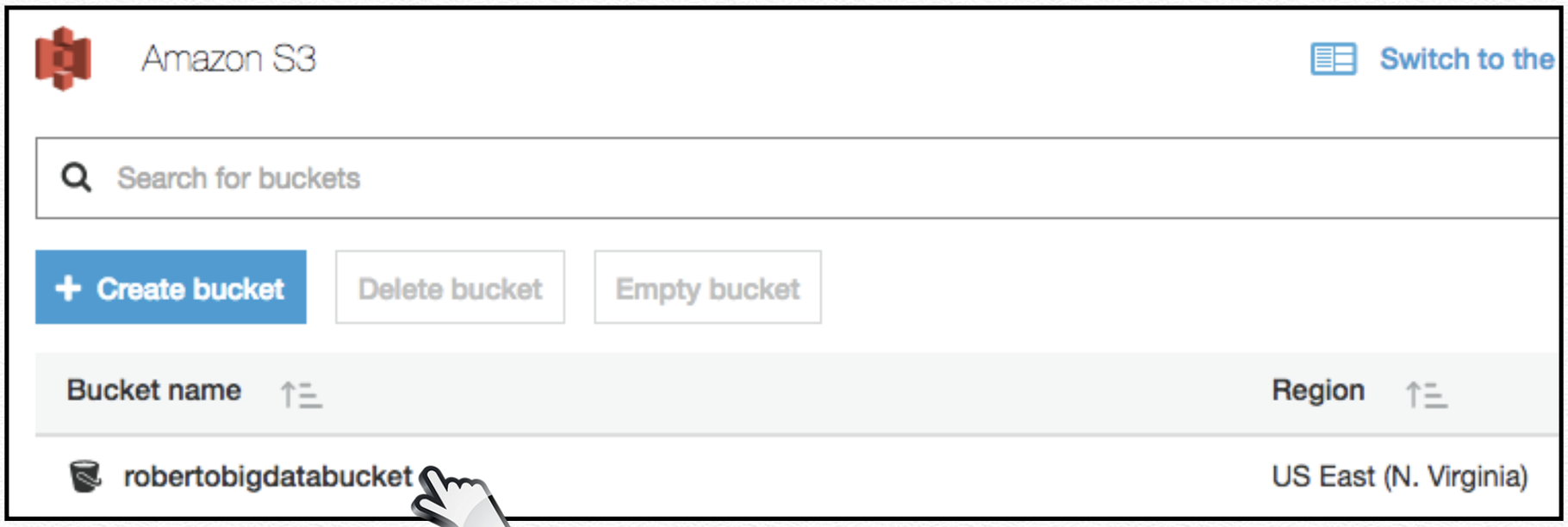
Hadoop 3 on AMAZON

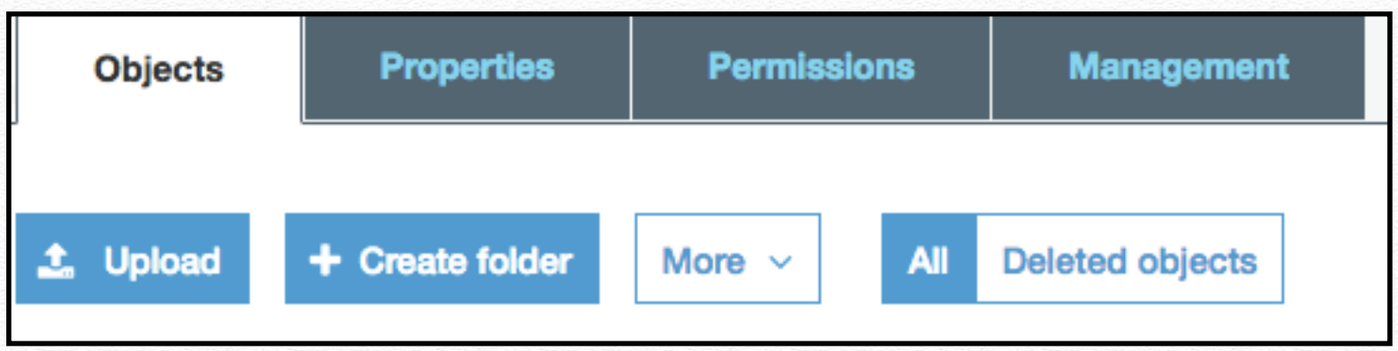
S3 and buckets



Hadoop 3 on AMAZON

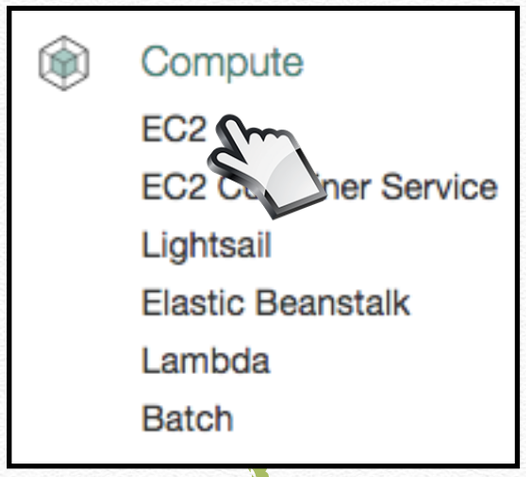
S3 and buckets

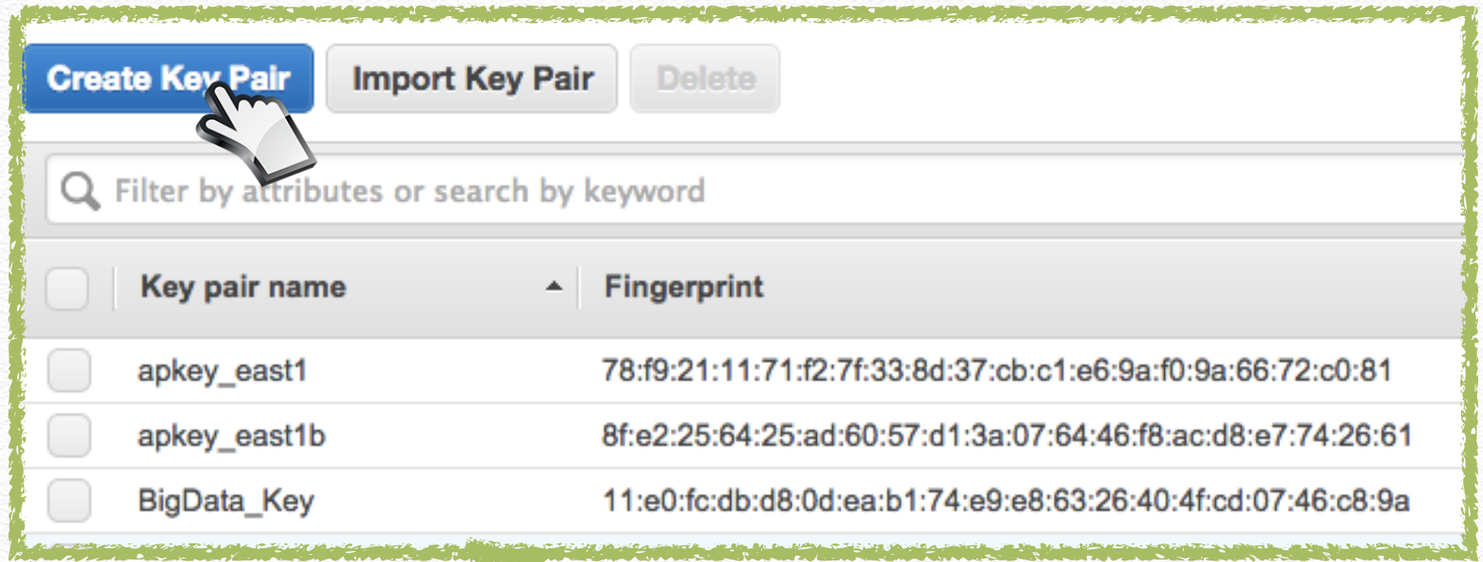


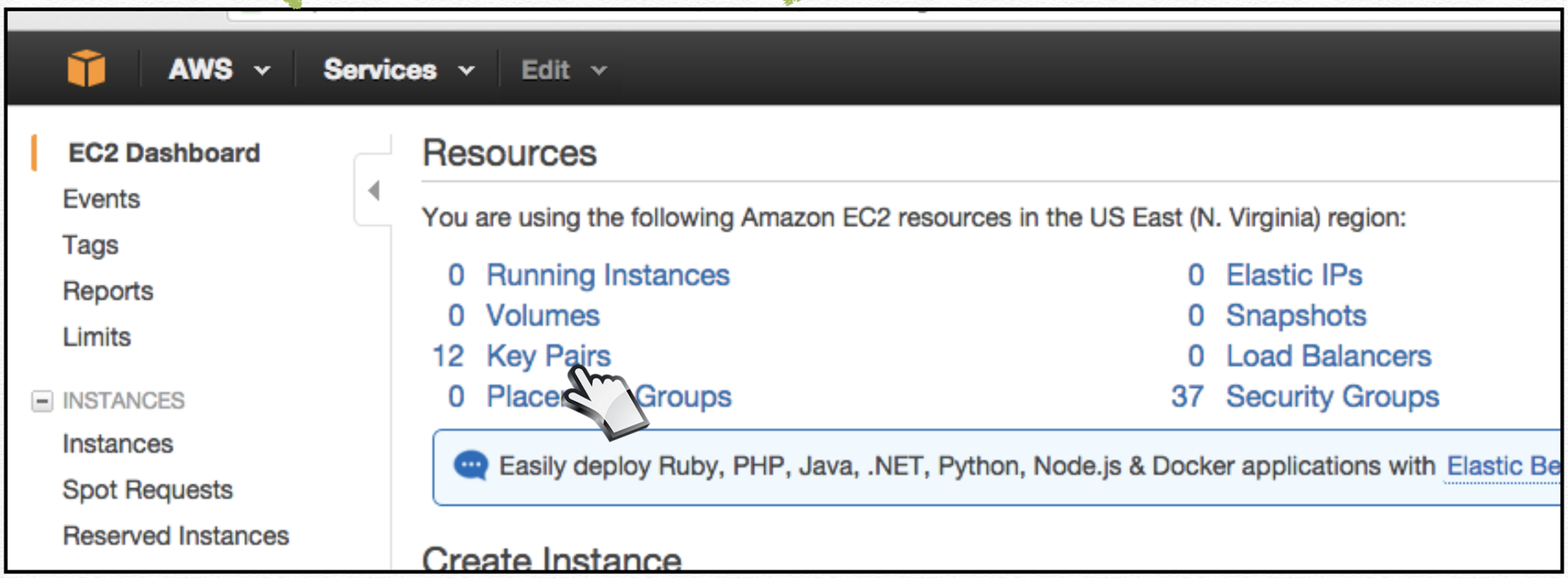


Hadoop 3 on AMAZON

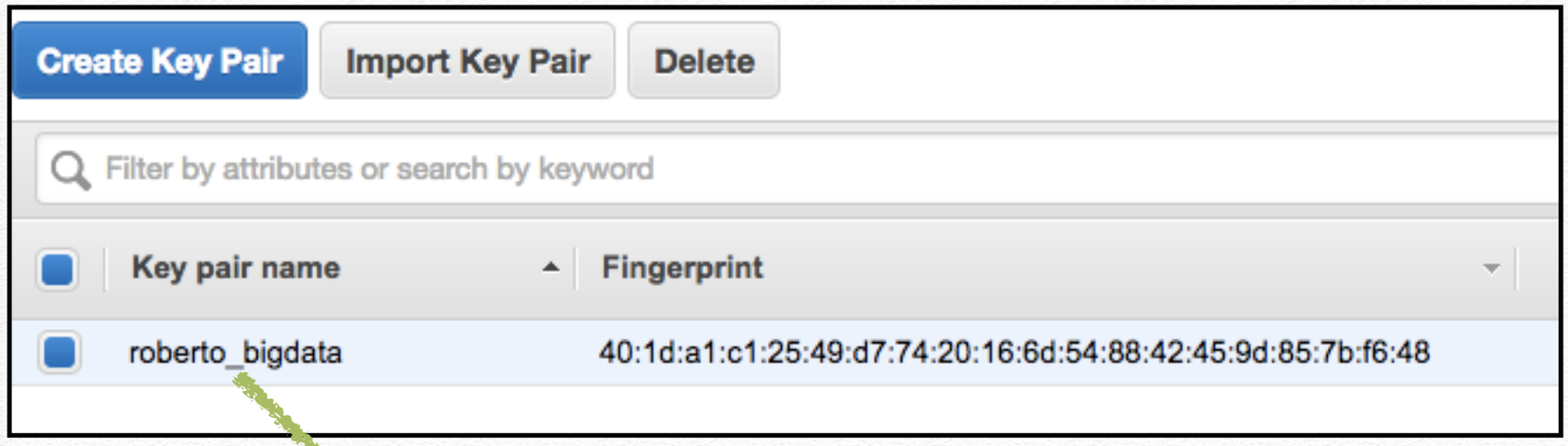
EC2 virtual servers







Hadoop 3 on AMAZON

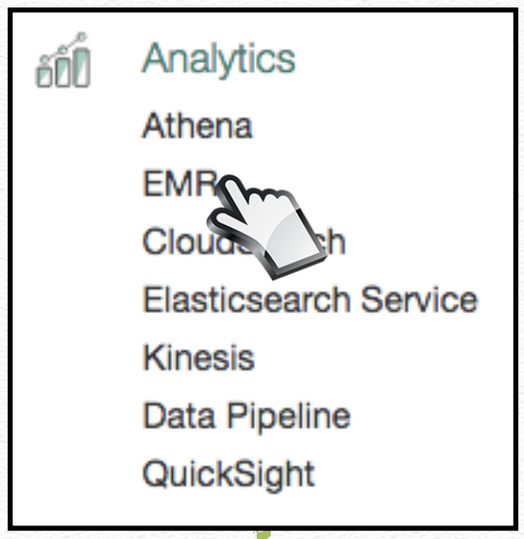


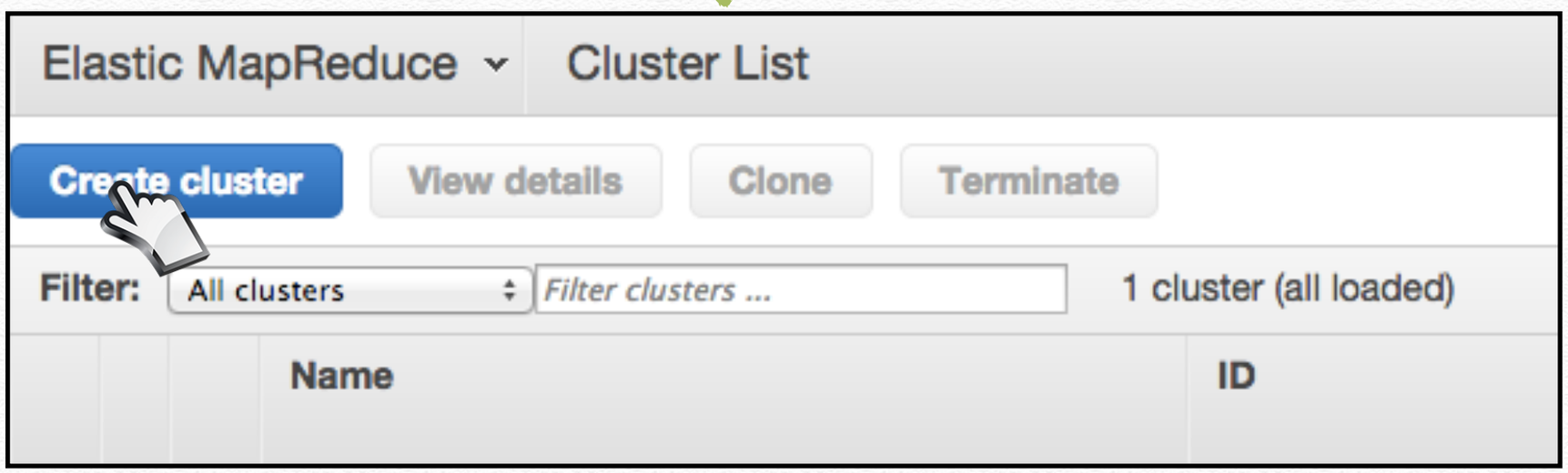
file .pem


```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA1y1mGpoREC945lmOEqM01VxOX3lCShOOrZTzf1tC9Oj4ReqChRuuF3X7ndvo
Y8LLTWKc4p/qFHjya9jjvlkOTcu4RuB8h8cP/HZhOZu1tZy1/XobA+elOpKBuhQACATXbs0ERIDD
fC6sLvwurMJHU0OznElbEJsMN/nEK8lYv4uu2gqfzbgoxHibHd6JiHFEk17ymZlB8D0dQhFGwBll
B29grlmyUCC1miA1pp8L3eHBqHXs0Vlw94tLM+ktLgeA4mdP4jvADRLt0quozmp89dj2ZHO8npGq
DMkDUJsXJlBcfHT7dmGBlxATuLdAjwHt56nwchA9dGXCVI7YFlJmtwlDAQABAoIBAHkVrAJGNlnk
TOVa+c7VFnMF+XhSVRI4RS66xfch1ODahHNbjsz2kZXUJ55iVDhnMl4+osgIcwHIOaqkpyq9+VWf
0PfZdVo2k3Fe8EEptSwYnnSgFLmyzdDTBrs+a/lXP1+zcLZXuymTXgMml+FRhi99xoGo6dzDUzHg
lBqHo8OM/wuFk3zStDlRJta1q6T12LyCyxQ5lDEWo5MJlBfiKUtQOEs73UlLHp2tylEeYcQKHpBO
BW9jkafue/QramwsyZmyBtRMcZm4wDmRqm0YjfGRP+gNCYD/bXhofqzHlsjFsuxVIMhLlQrYZm9A
iFZdZBSavWw0rxlc2zUdgJ3FAskCgYEA/aExJIQwvSdvHb0AedqpTvfbmj9iAKJ1xYwbz04grnph
Nw1eWt40vjVRZaVCaNk33QJjlHZJBr5w7BFqcjdLuoes2kGMnOFANk7kQ0HAnPDvQttt3gqlvcn6
Eq23C7Dhj+3qKSI2g4b7qSW43eys/FiYMvvWXNfo1L0PDmtGJ+MCgYEA2TA1uMX4Vu3SPDDLDXlV
cExRw5RY60dUwnfbqealTtf+VNui+9XZuXSBzQj0782xmeJwzbFHSQFzeZzMABWYL8tUufVutoGD
1SkC0uxcYq/5URD3zw5dgybwlJZMzcTpN3Ug7u6oeNPRYQ9FDcfMMMpEyBjFM2JzGn2KHmcSNh0C
gYADfaJ1hDQ/hjlrSVymULIf2h11pAWvSI0iaOlBEywz+o83+MEhswk6zPUP1xRw6Pxx2ESQ/8/F
ff2D4kF6zPQH3zIr1tcemaMOnEtb+Z2zC0Xfv7FlavQZBpyggS+Rw593laklY8koSkVQcKp4s3c2
CeoeWYT7xiT4OJnqcuOGTQKBgEhXQ32RCz/BJSaBRhDt2iR2d0GCtv9fGf9QNu1naP2B+ie/pJDn
UQbxpI5rrlH1nHNaNiJ9KvZJMnJOZLCegLtiqYzrD/5gM04bw+IHclo4rP1wfmoMKd+WZ6MjDN/4
94lOTTLocVsVioces+x8ISobJT/U6FJON3KaYBfyUuGNAoGBAKmZhyvN4M7Dkcj3nCRFahyrMnaz
okfHacmb3vQ3Q0wSP51k8YMM4qXIqGhCWm3wY8jsufUt+Lk5lcfNjTLyAMP98WY5LppZv/80lAZ5
Wltx8fAuMRtoR5ojqp3wo1ZUNzp+0saPPVQ7n6CBWjj3TbzU5w6bR4Ul2kiPomt40rEx
-----END RSA PRIVATE KEY-----
```

Hadoop 3 on AMAZON

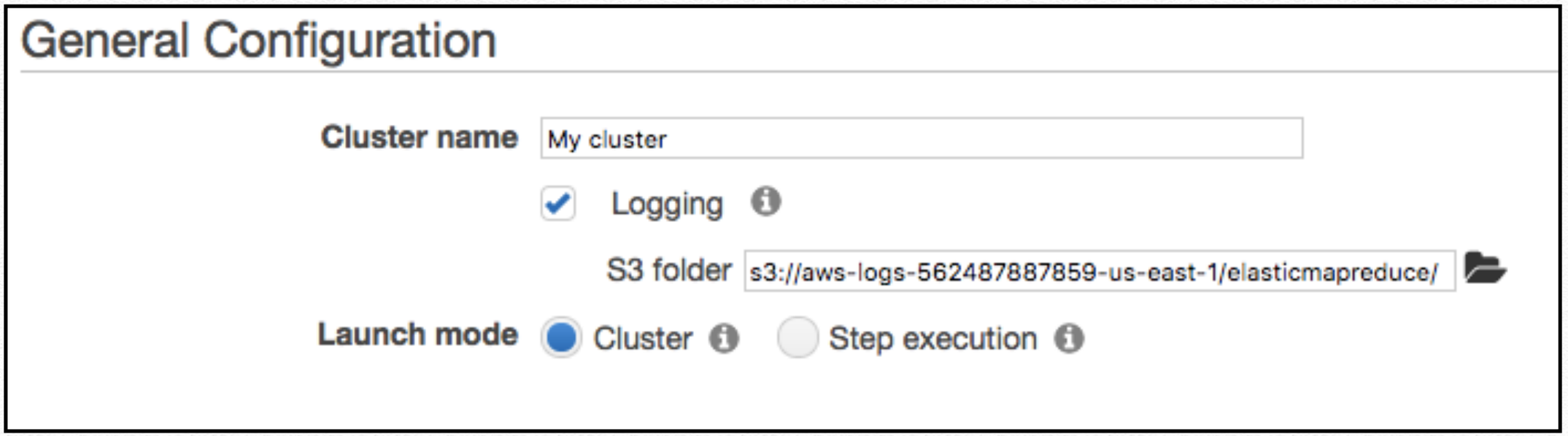
Elastic Map Reduce





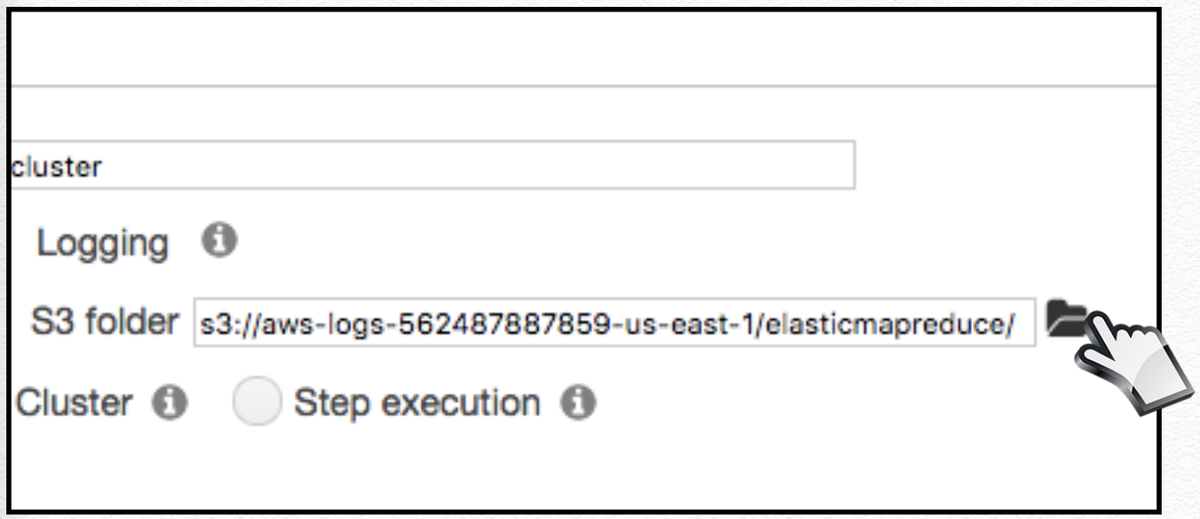
Hadoop 3 on AMAZON

EMR cluster



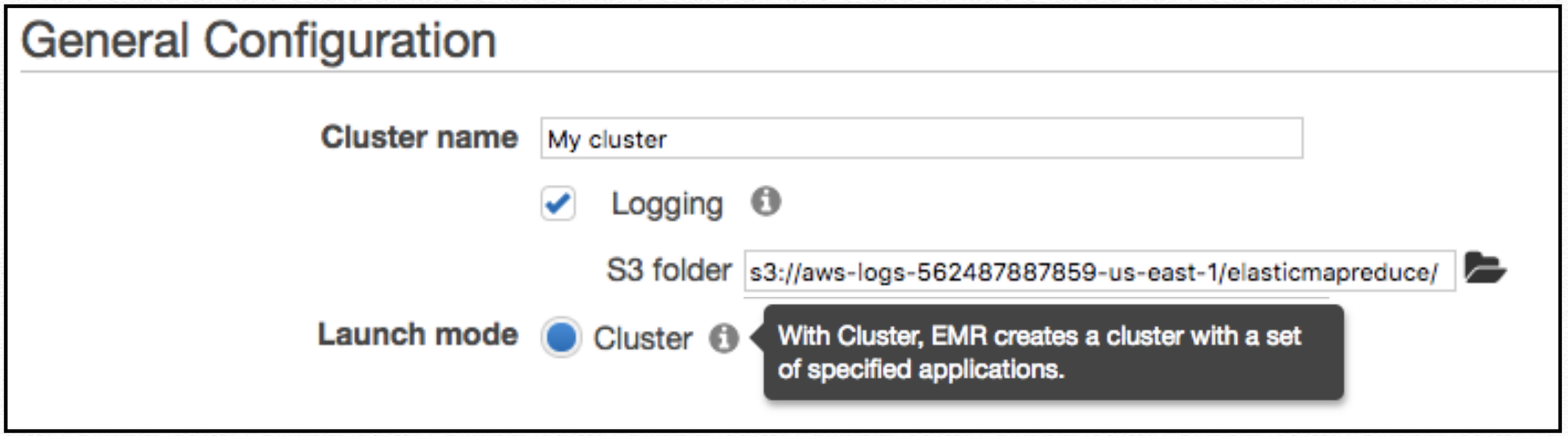
Hadoop 3 on AMAZON

EMR cluster



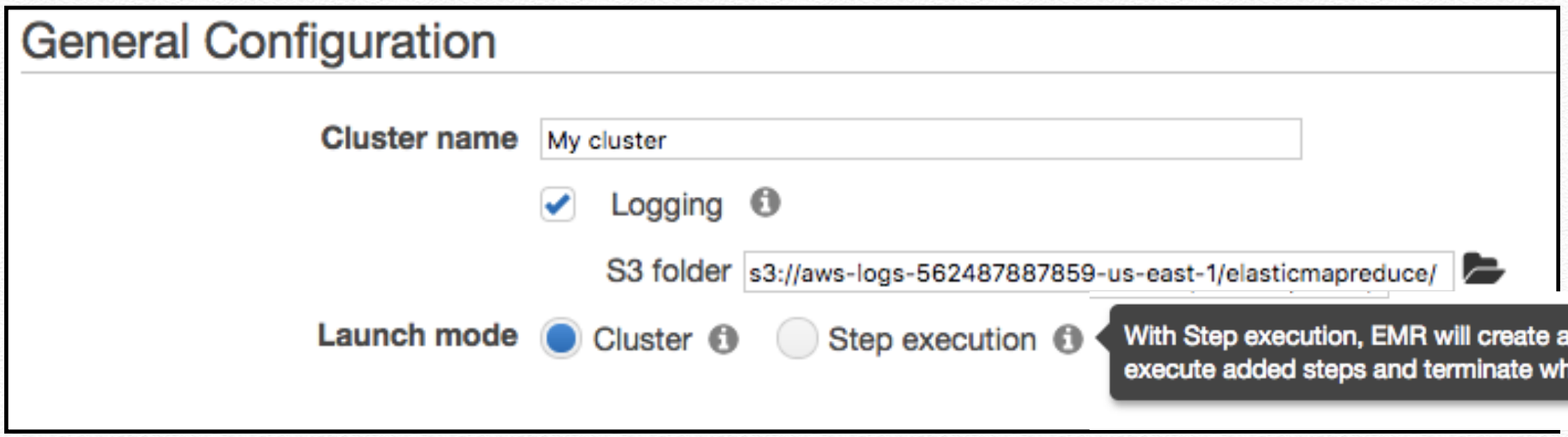
Hadoop 3 on AMAZON

EMR cluster



Hadoop 3 on AMAZON

EMR cluster



Hadoop 3 on AMAZON

EMR cluster

Software configuration

Vendor (●) Amazon ( ) MapR

Release emr-5.4.0 ⓘ

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 MapR - M3 - 4.0.2

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

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

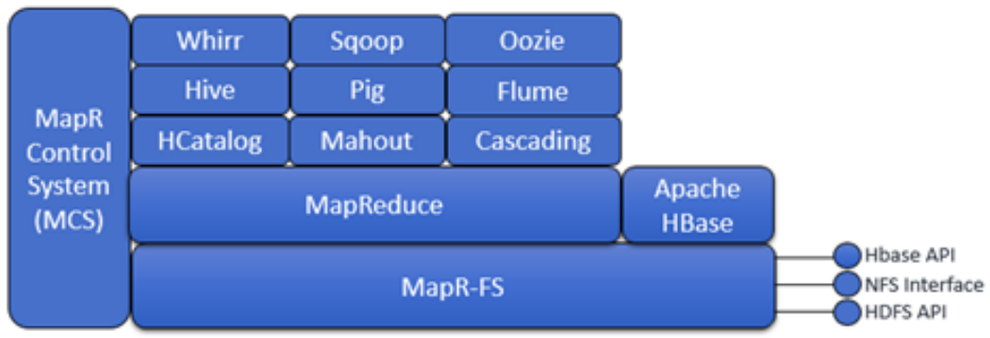
MapR 5.0 Documentation / Home

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.

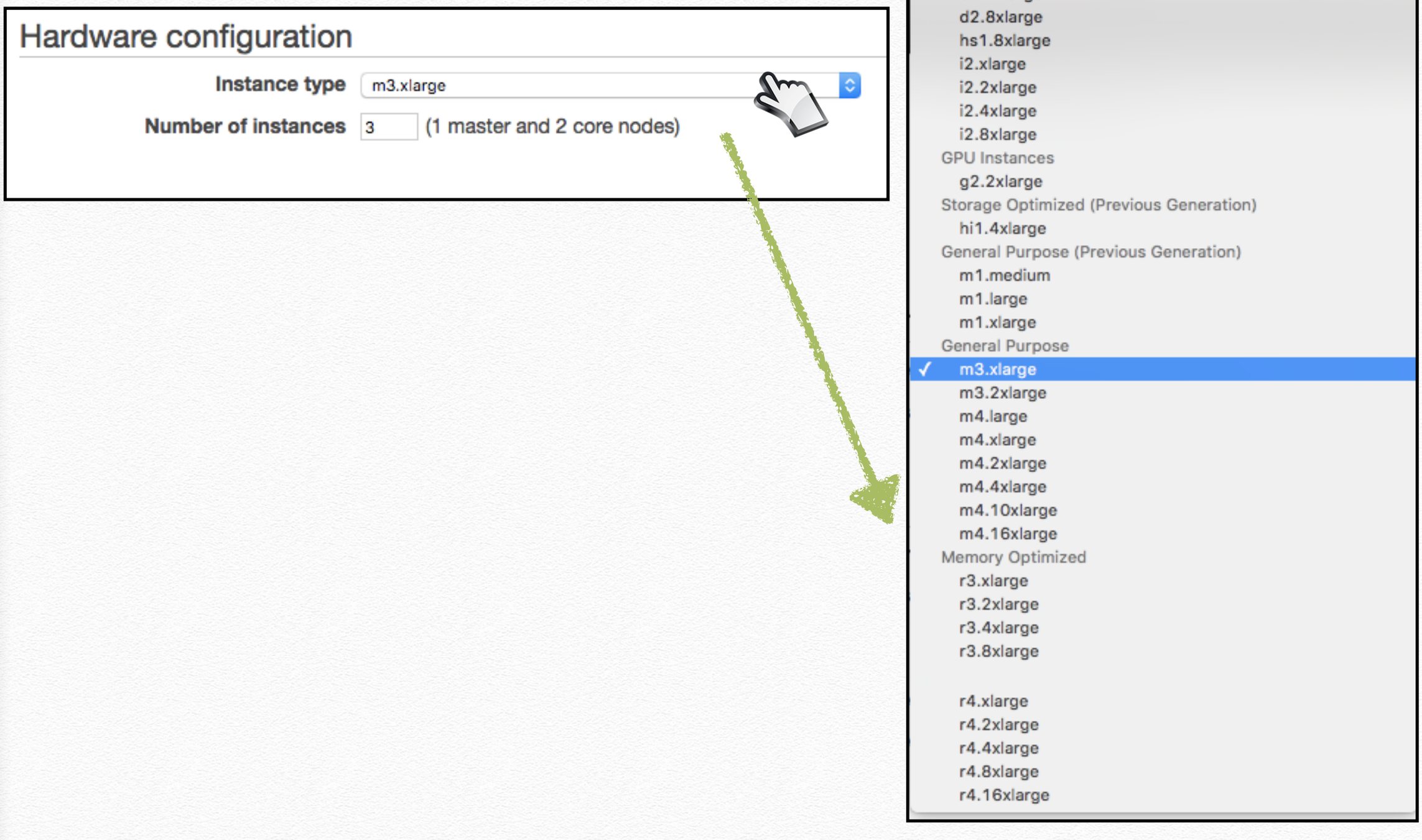
The following image displays a high-level view of the MapR Distribution for Apache Hadoop:



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

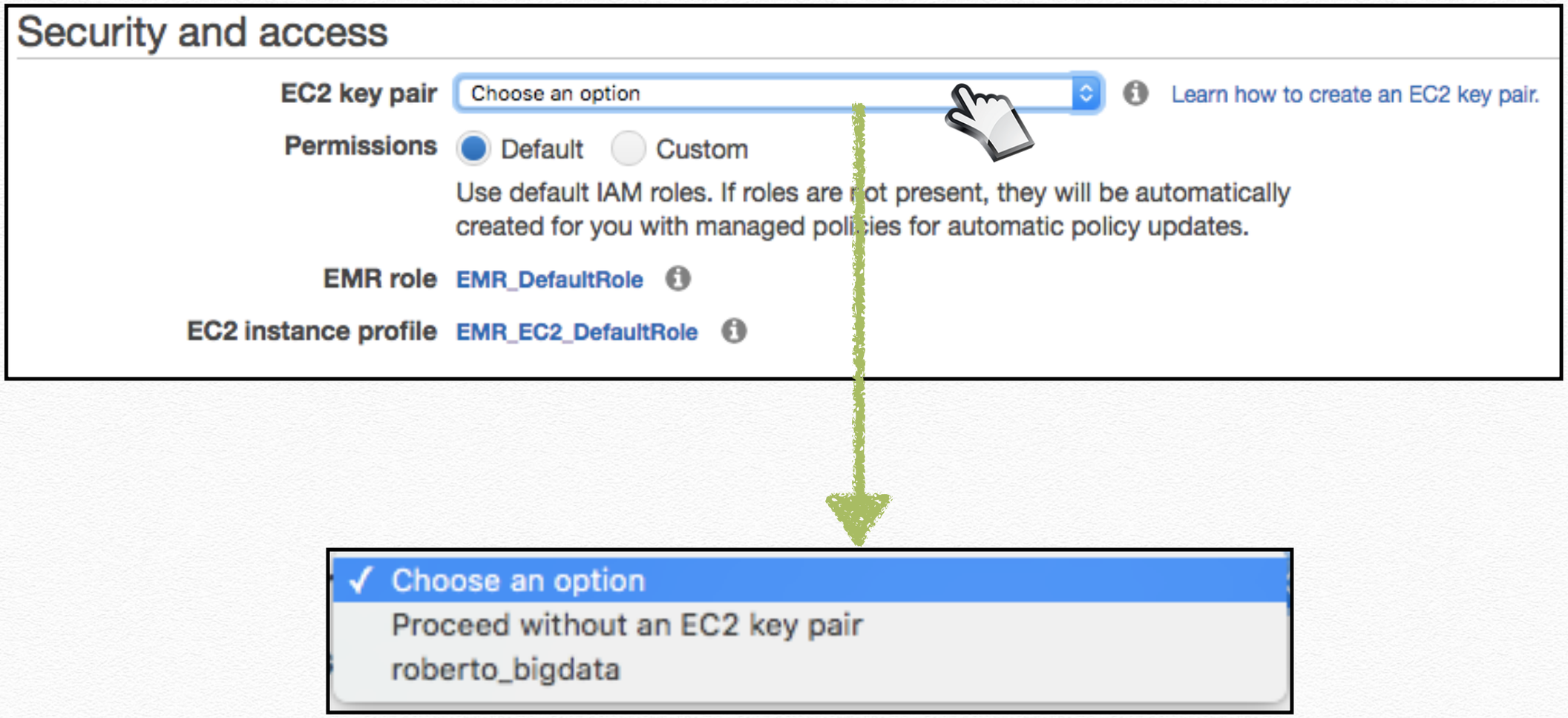
Hadoop 3 on AMAZON

EMR cluster



Hadoop 3 on AMAZON

EMR cluster



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

ID: j-47LWBSL8AKPS
Creation date: 2017-04-04 11:55 (UTC+2)
Elapsed time: 2 minutes
Auto-terminate: No
Termination protection: Off Change

Configuration Details

Release label: emr-5.4.0
Hadoop distribution: Amazon 2.7.3
Applications: Ganglia 3.7.2, Hive 2.1.1, Hue 3.11.0, Mahout 0.12.2, Pig 0.16.0, Tez 0.8.4
Log URI: s3://robertobigdatabucket/
EMRFS consistent view: Disabled

Network and Hardware

Availability zone: us-east-1c
Subnet ID: --
Master: Bootstrapping 1 m3.xlarge
Core: Provisioning 2 m3.xlarge
Task: --

Security and Access

Key name: roberto_bigdata
EC2 instance profile: EMR_EC2_DefaultRole
EMR role: EMR_DefaultRole
Visible to all users: All Change
Security groups for Master: sg-98da73f3 (ElasticMapReduce-master)

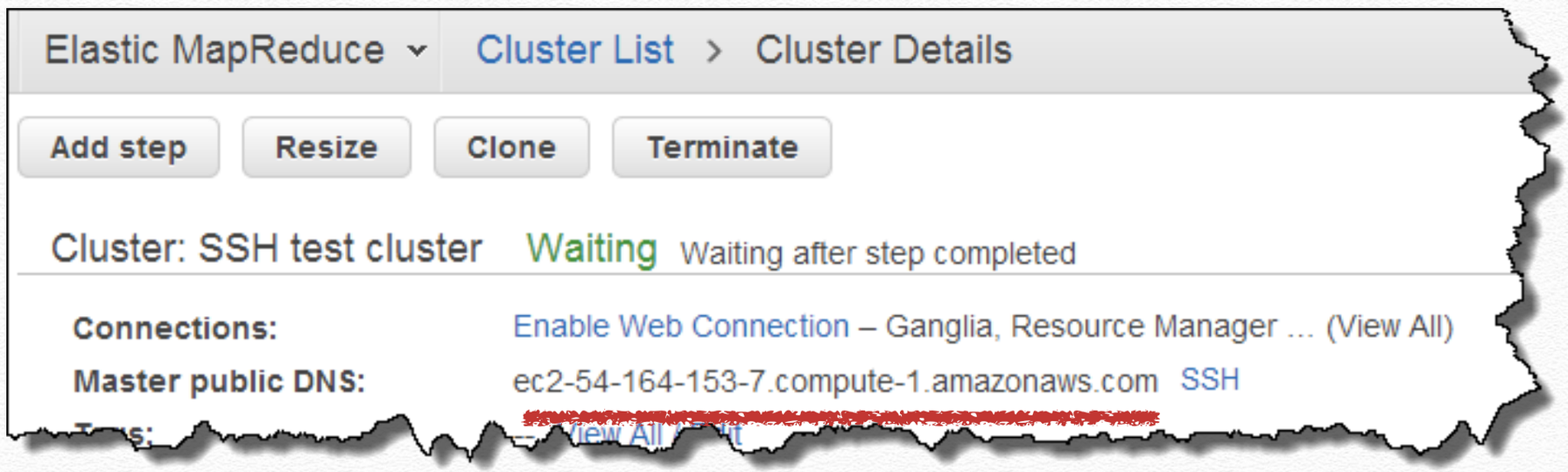
Hadoop 3 Running on AWS

- **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>:~
```




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

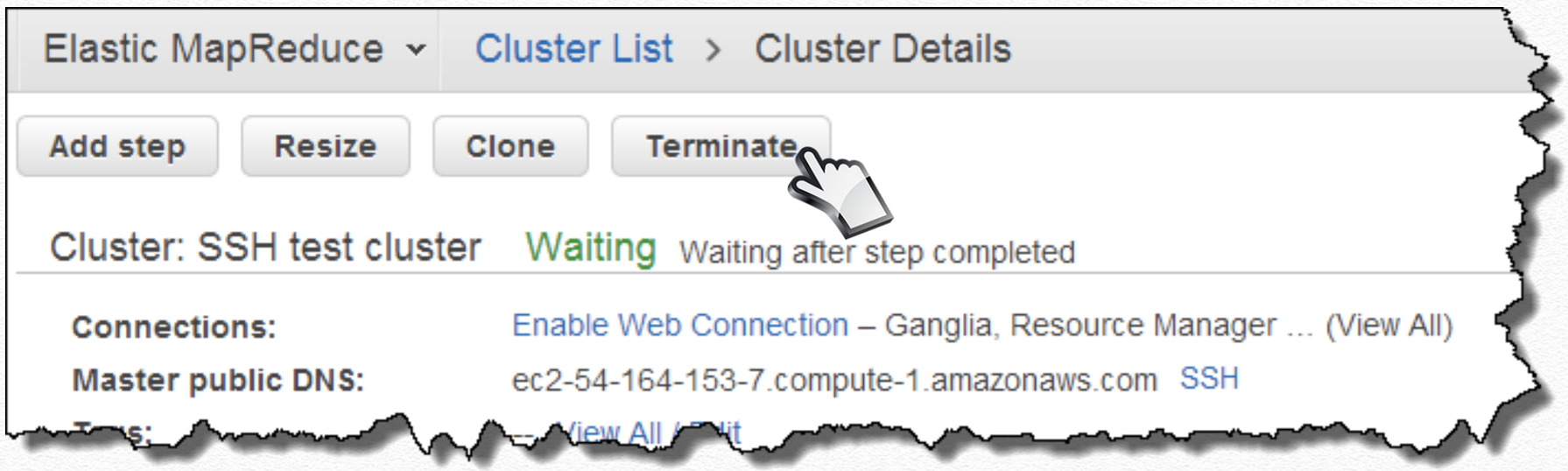
Hadoop 3 Running on AWS

- **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
- Finally **TERMINATE** your cluster



Hadoop 3 Configuration and First Examples

Big Data - 25/03/2020