Module 4

HBase, MapReduce and Oozie

Thanachart Numnonda, Executive Director, IMC Institute

Mr. Aekanun Thongtae, Big Data Consultant, IMC Institute

Thanisa Numnonda, Faculty of Information Technology,

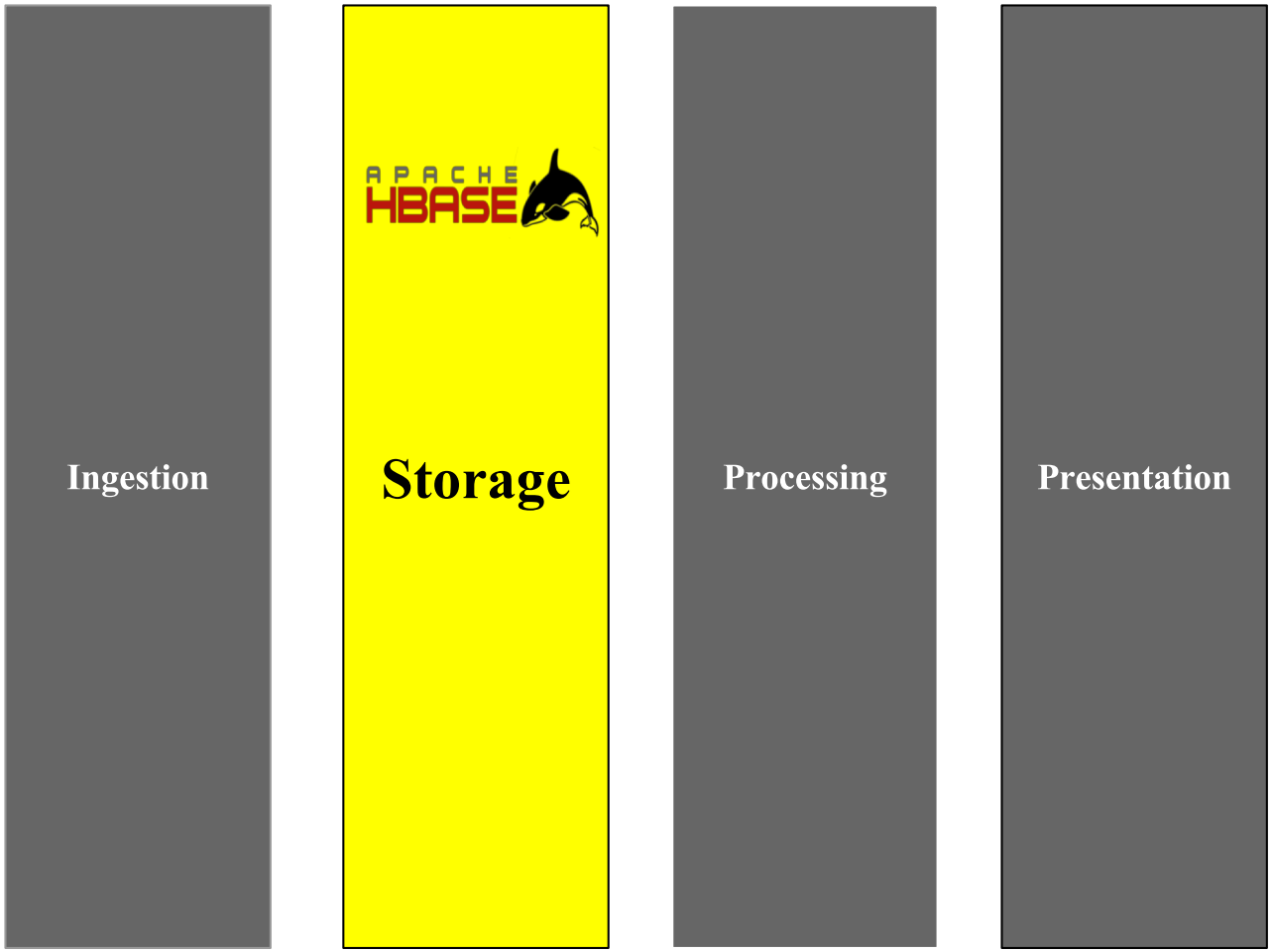
King Mongkut's Institute of Technology Ladkrabang

Introduction

An open source, non-relational, distributed database

- **Apache HBase™ is the Hadoop database, a distributed, scalable, big data store.**
- **Use Apache HBase™ when you need random, realtime read/write access to your Big Data.**
- **Goal is the hosting of very large tables -- billions of rows X millions of columns**

Key Infrastructure Elements



HBase Features

- Hadoop database modelled after Google's Bigtable
- Column oriented data store, known as Hadoop Database
- Support random realtime CRUD operations (unlike HDFS)
- NoSQL Database
- Open source, written in Java
- Run on a cluster of commodity hardware

Data Model in HBase

- Column-oriented database: the tables are sorted by row
- Tables
  - Table is a collection of rows.
  - Row is a collection of column families.
  - Column family is a collection of columns.
  - Column is a collection of key value pairs.

Rowid	Column Family			Column Family			Column Family			Column Family		
	col1	col2	col3	col1	col2	col3	col1	col2	col3	col1	col2	col3
1												
2												
3												

- Given this RDBMS:

ID (Primary key)	Last name	First name	Password	Timestamp
1234	Smith	John	Hello, world!	20130710
5678	Cooper	Joyce	wysiwyg	20120825
5678	Cooper	Joyce	wisiwig	20130916

- Logical view in HBase:

Row-Key	Value (CF, Qualifier, Version)
1234	info {'lastName': 'Smith', 'firstName': 'John'} pwd {'password': 'Hello, world!'}
5678	info {'lastName': 'Cooper', 'firstName': 'Joyce'} pwd {'password': 'wysiwyg'@ts 20130916, 'password': 'wisiwig'@ts 20120825}

HBase vs. RDBMS

	HBase	RDBMS
Hardware architecture	Similar to Hadoop. Clustered commodity hardware. Very affordable.	Typically large scalable multiprocessor systems. Very expensive.
Fault Tolerance	Built into the architecture. Lots of nodes means each is relatively insignificant. No need to worry about individual node downtime.	Requires configuration of the HW and the RDBMS with the appropriate high availability options.
Typical Database Size	Terabytes to Petabytes - hundred of millions to billions of rows.	Gigabytes to Terabytes – hundred of thousands to millions of rows.
Data Layout	A sparse, distributed, persistent, multidimensional sorted map.	Rows or column oriented.
Data Types	Bytes only.	Rich data type support.
Transactions	ACID support on a single row only	Full ACID compliance across rows and tables
Query Language	API primitive commands only, unless combined with Hive or other technology	SQL
Indexes	Row-Key only unless combined with other technologies such as Hive or IBM's BigSQL.	Yes
Throughput	Millions of queries per second	Thousands of queries per second

HBase Shell Commands

- List all tables:

```
hbase(main):001:0> list
```

- Create tables:

```
hbase(main):002:0> create 'testTable', 'cf'
```

- Insert data into a table:

```
hbase(main):003:0> put 'testTable','rowA','cf:columnName','val1'
```

HBase Shell Commands (Cont.)

- Retrieve data from a table:

```
hbase(main):009:0> get 'testTable','rowA'
```

- Iterate through a table:

```
hbase(main):011:0> scan 'testTable'
```

- Delete a table:

```
hbase(main):012:0> disable 'testTable'
hbase(main):013:0> drop 'testTable'
```

Which one to use?

- **HDFS**
  - Only append dataset (no random write)
  - Read the whole dataset (no random read)
- **HBase**
  - Need random write and/or read
  - Has thousands of operation per second on TB+ of data
- **RDBMS**
  - Data fits on one big node
  - Need full transaction support
  - Need real-time query capabilities

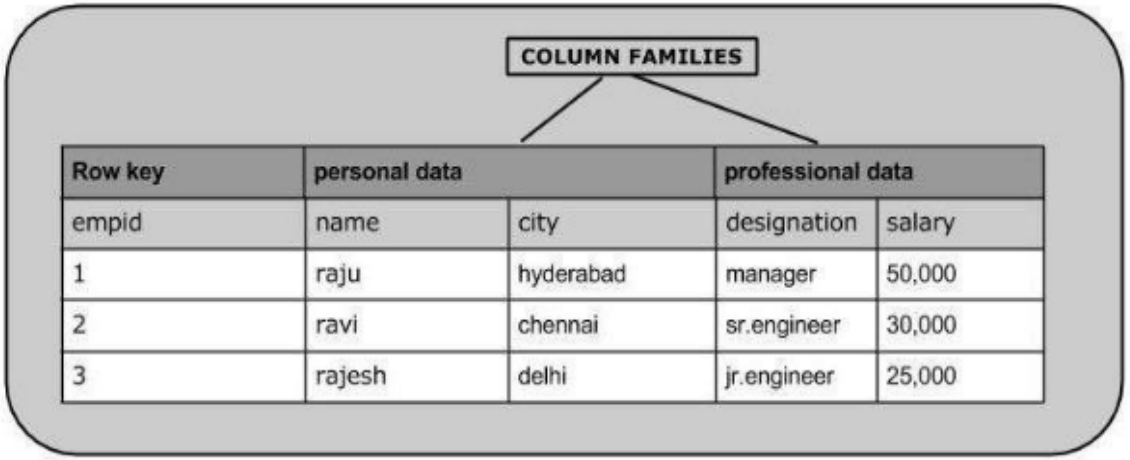
Hands-On: Running HBase

Hbase shell

Row key	personal data	professional data

```
[root@quickstart /]# hbase shell

hbase(main):001:0> create 'employee', 'personal
data','professional data'

hbase(main):002:0> list


 TABLE
 employee
 1 row(s) in 0.0310 seconds
```

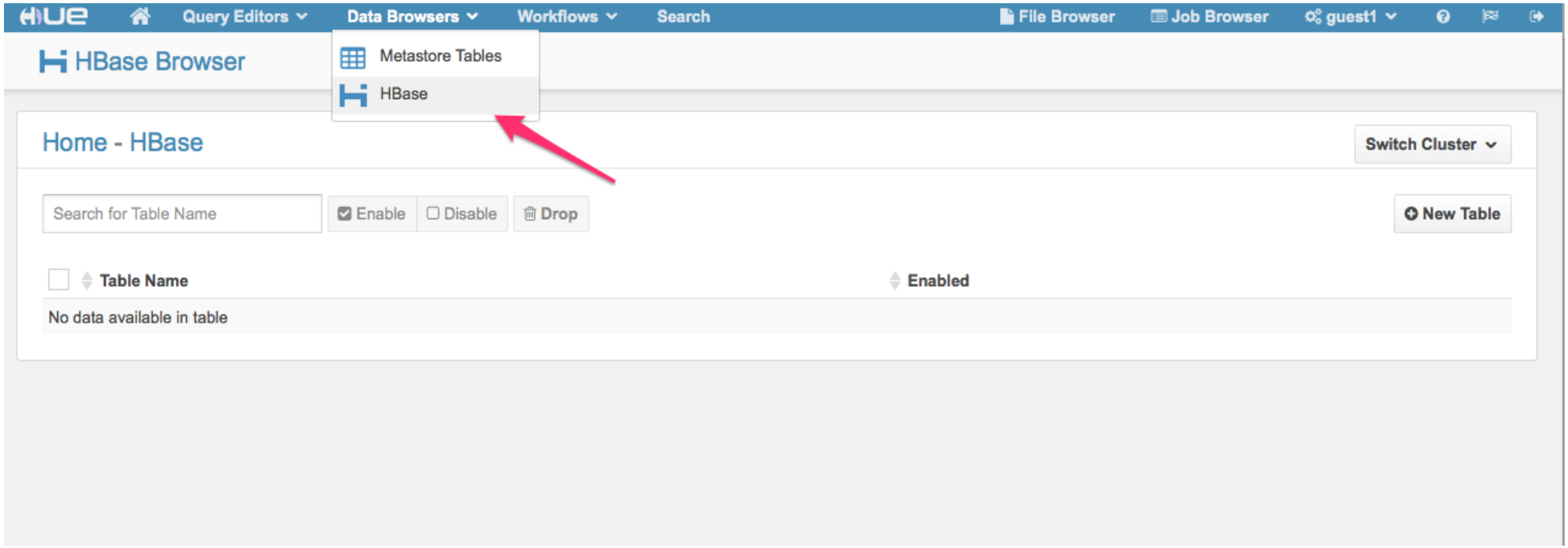
Create Data



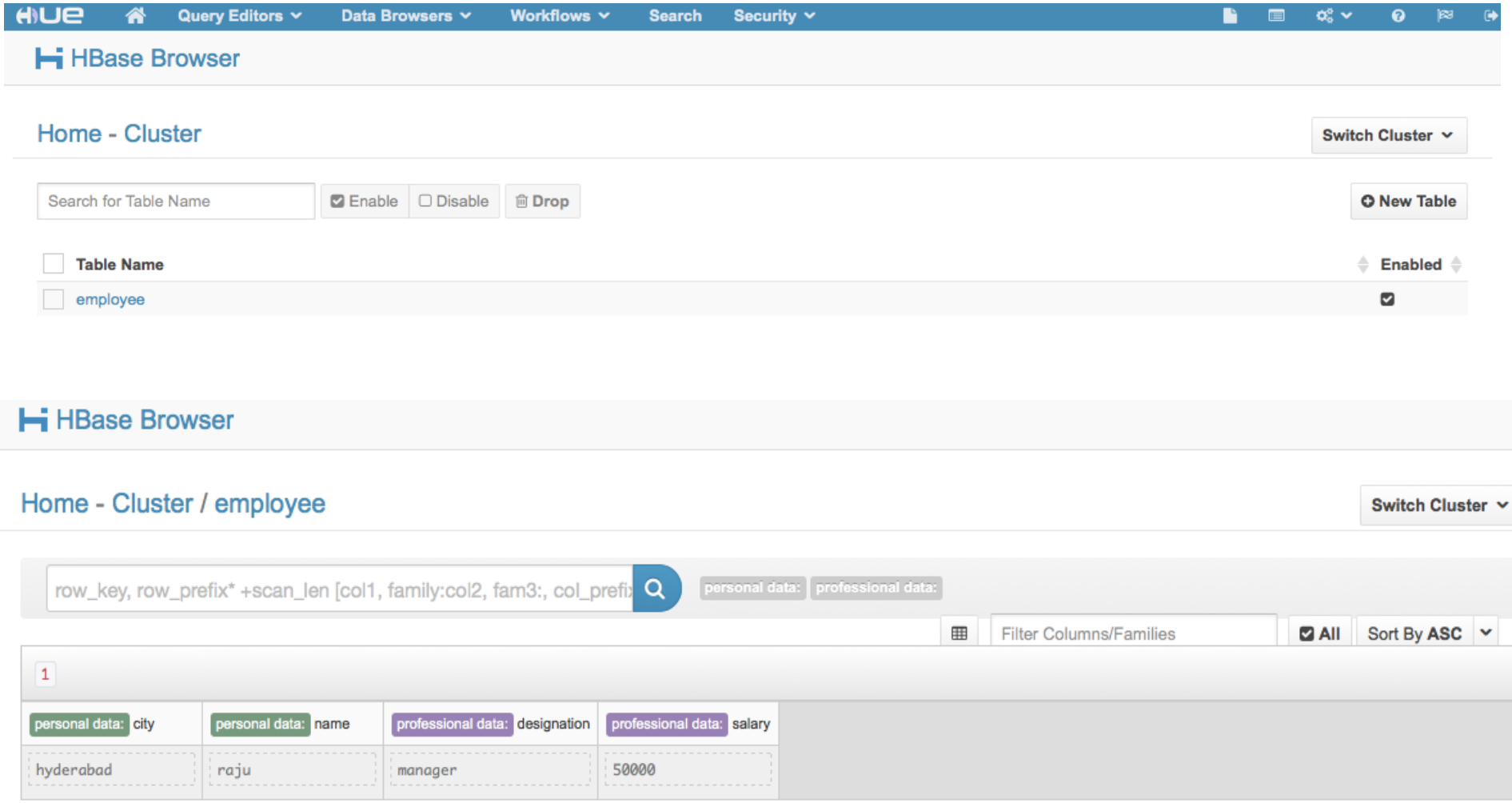
Row key	personal data		professional data	
empid	name	city	designation	salary
1	raju	hyderabad	manager	50,000
2	ravi	chennai	sr.engineer	30,000
3	rajesh	delhi	jr.engineer	25,000

```
hbase(main):023:0> put 'employee','1','personal data:name’,
'raju'

hbase(main):024:0> put 'employee','1','personal data:city’,
'hyderabad'

hbase(main):025:0> put 'employee','1','professional
data:designation’,'manager'

hbase(main):026:0> put 'employee','1','professional
data:salary','50000'
```

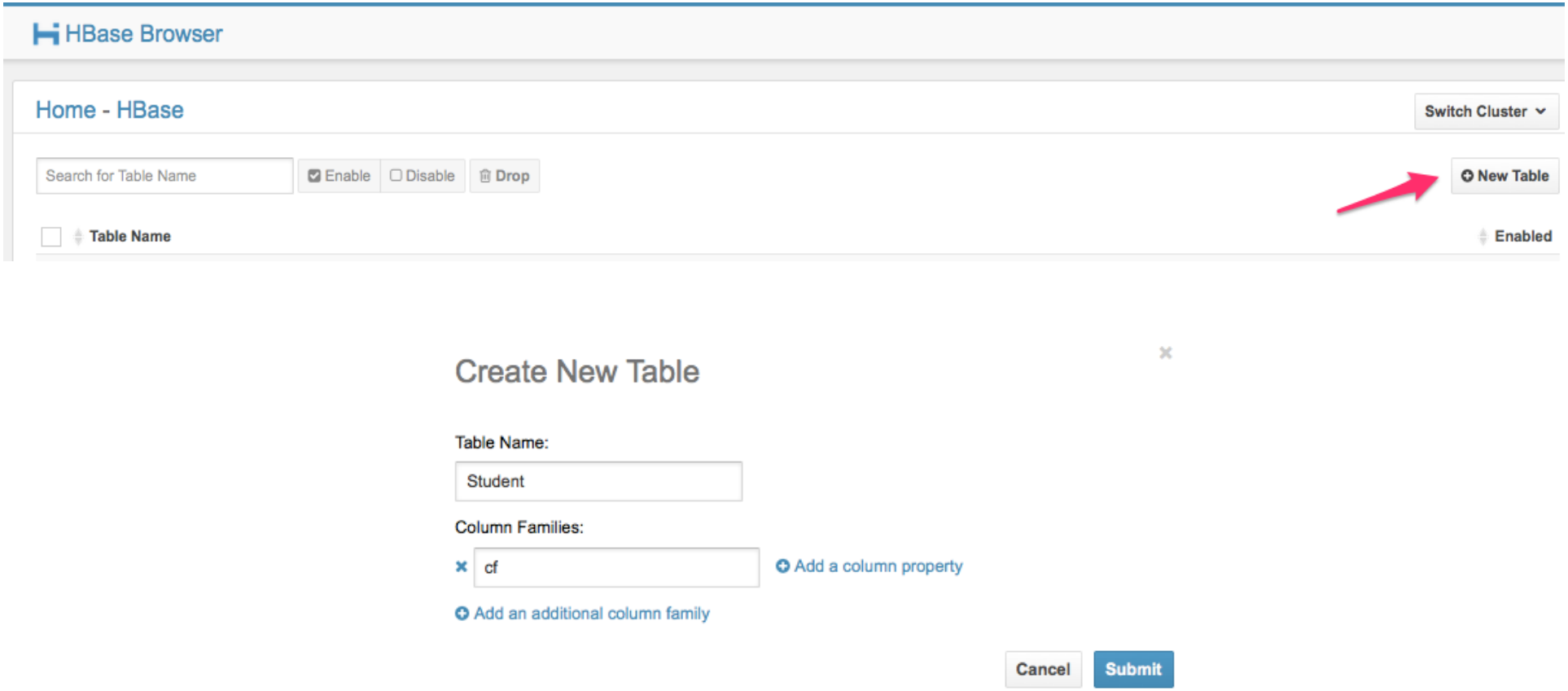
Running HBase Browser



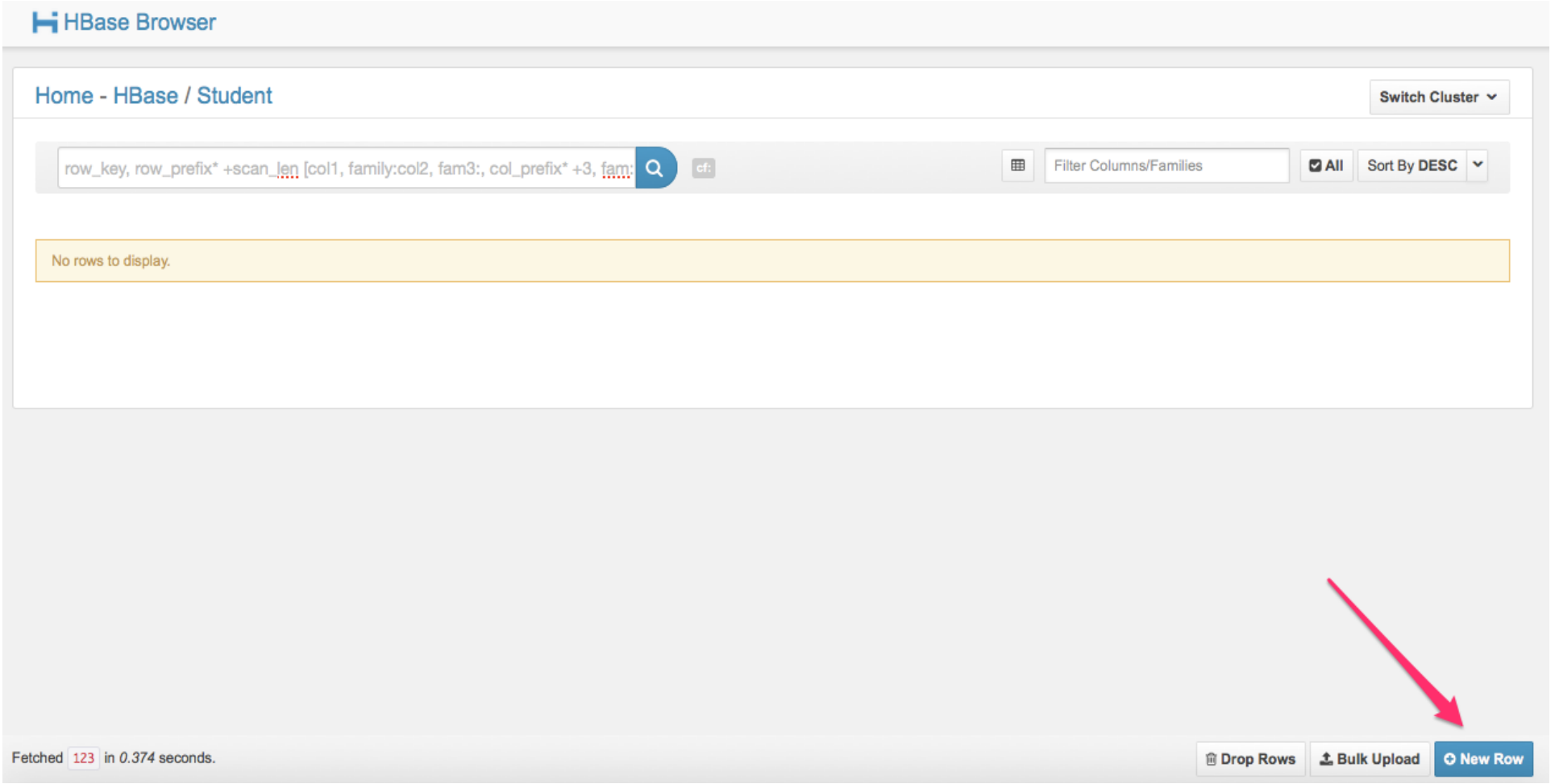
Viewing Employee Table



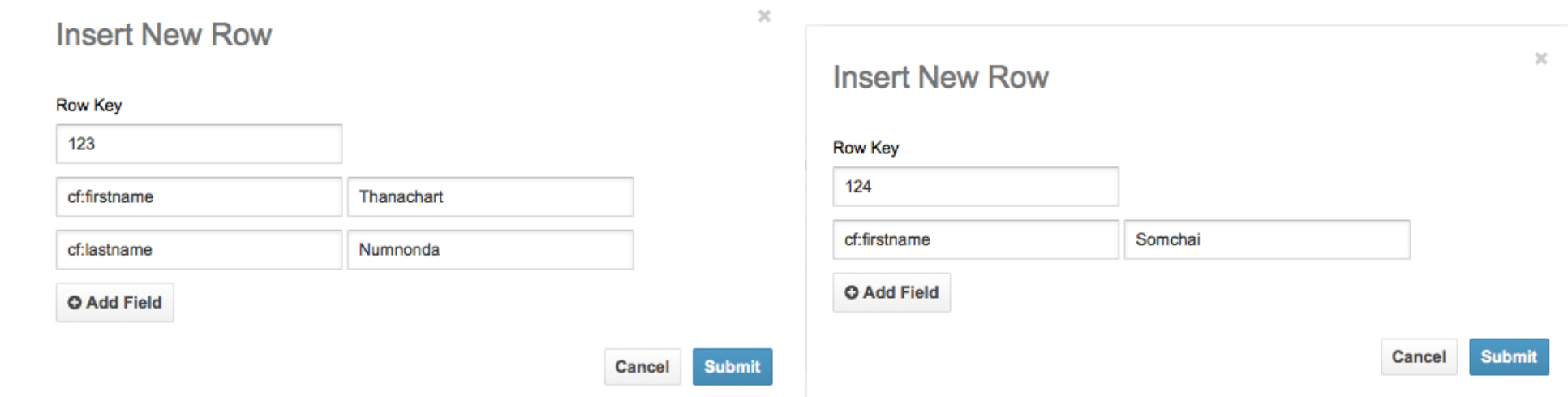
Create a table in HBase

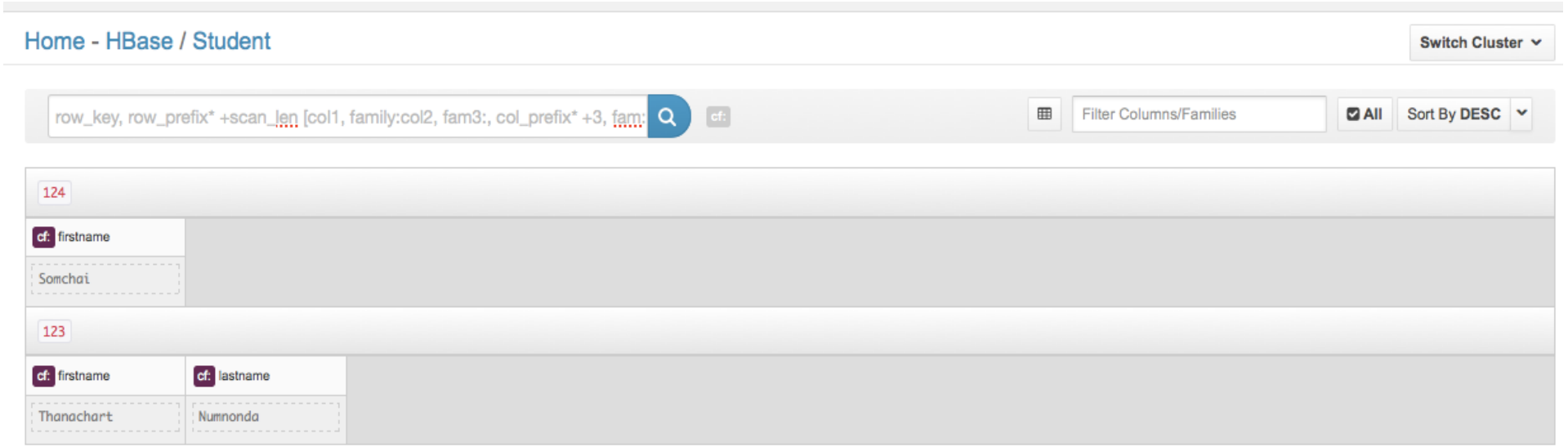


Insert a new row in a table



Add field into a new row

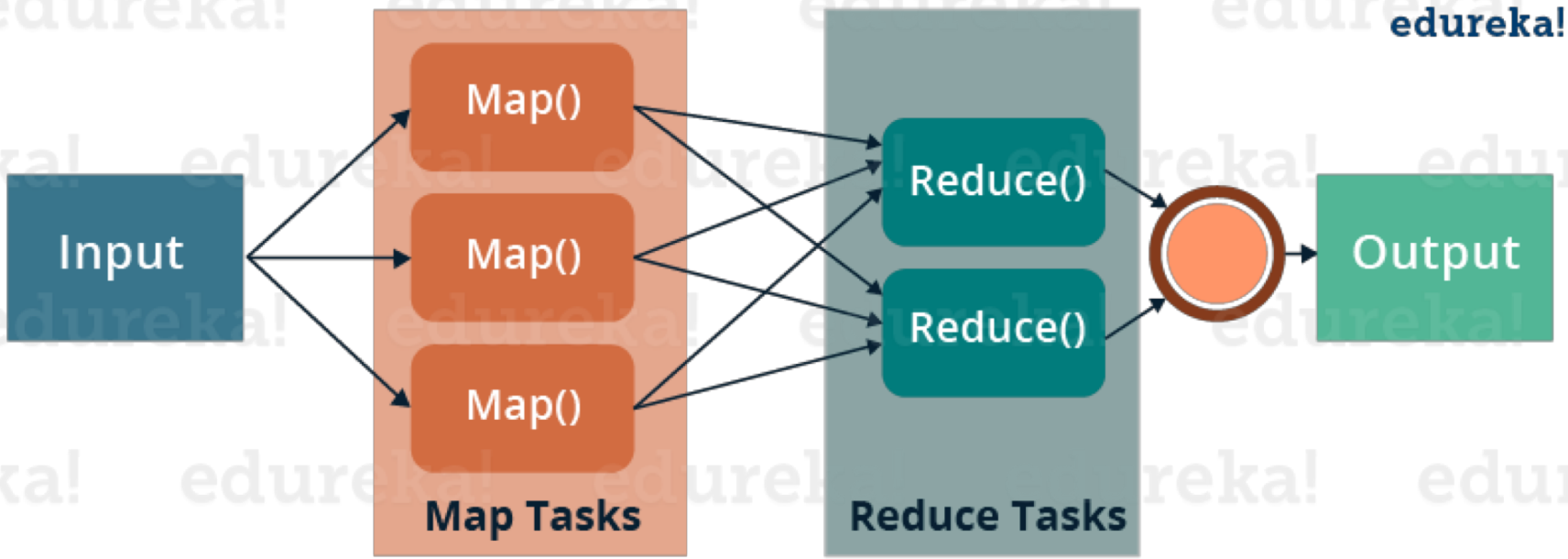




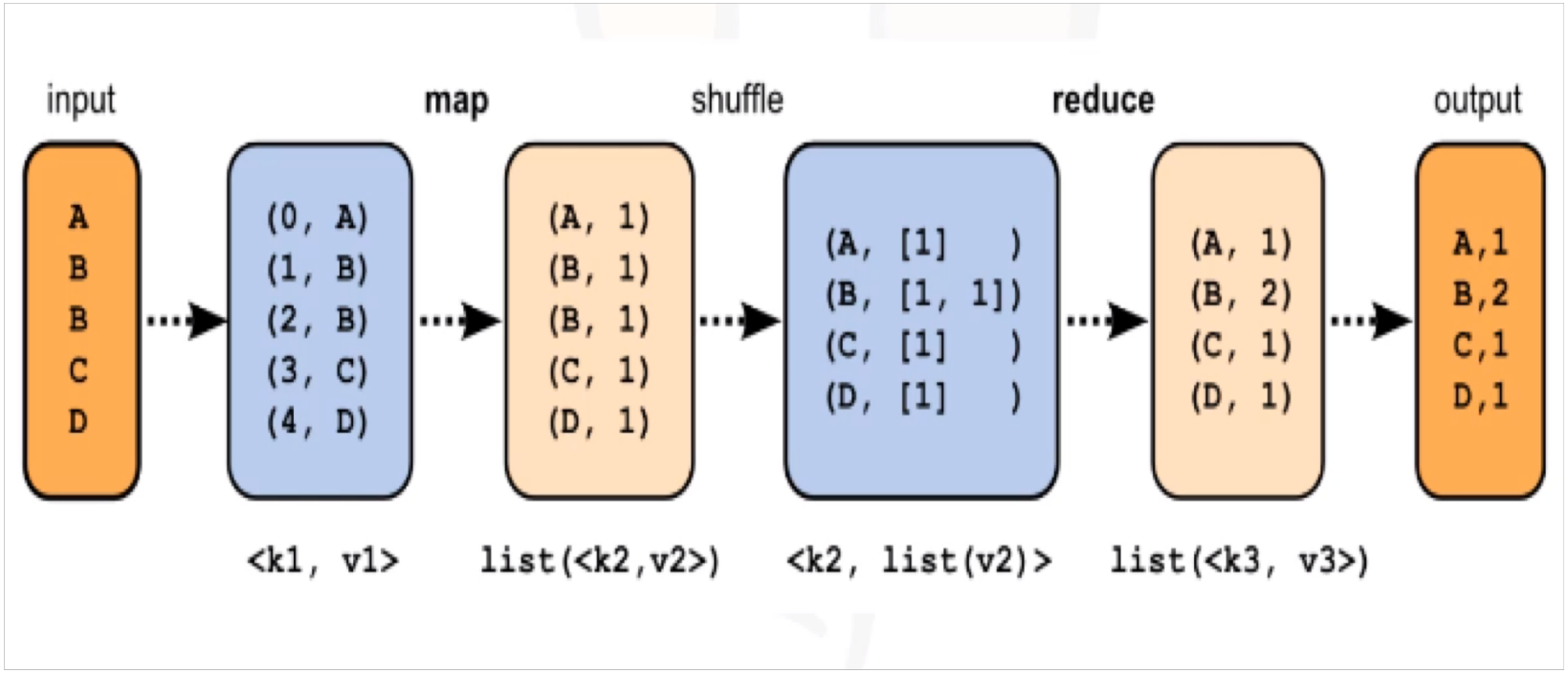

hadoop
Map Reduce

MapReduce Framework

- MapReduce is a programming framework that allows us to perform distributed and parallel processing on large data sets in a distributed environment.



MapReduce Framework



Source: www.bigdatauniversity.com

MapReduce-Architecture View

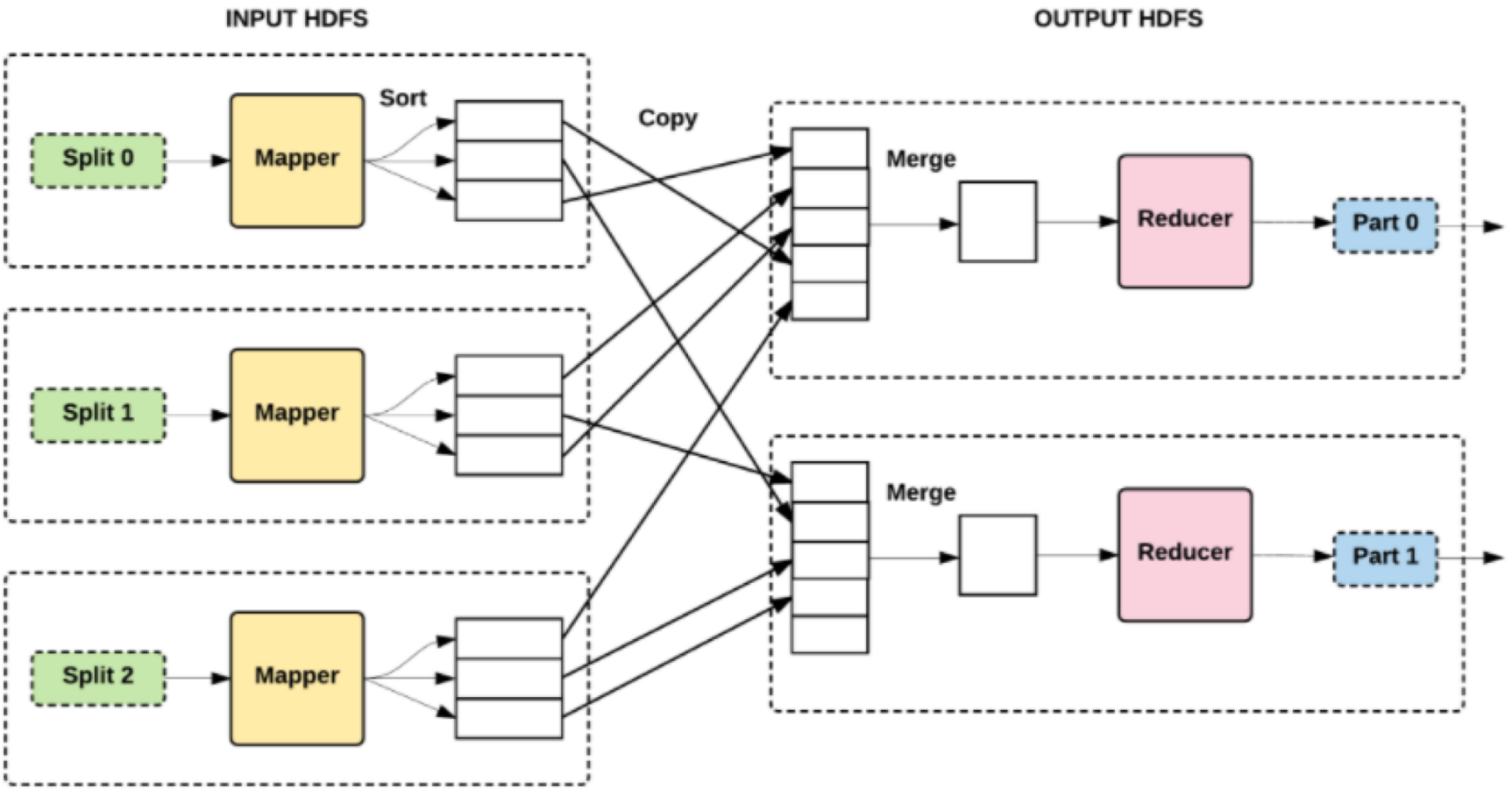


Figure 06: MapReduce in action

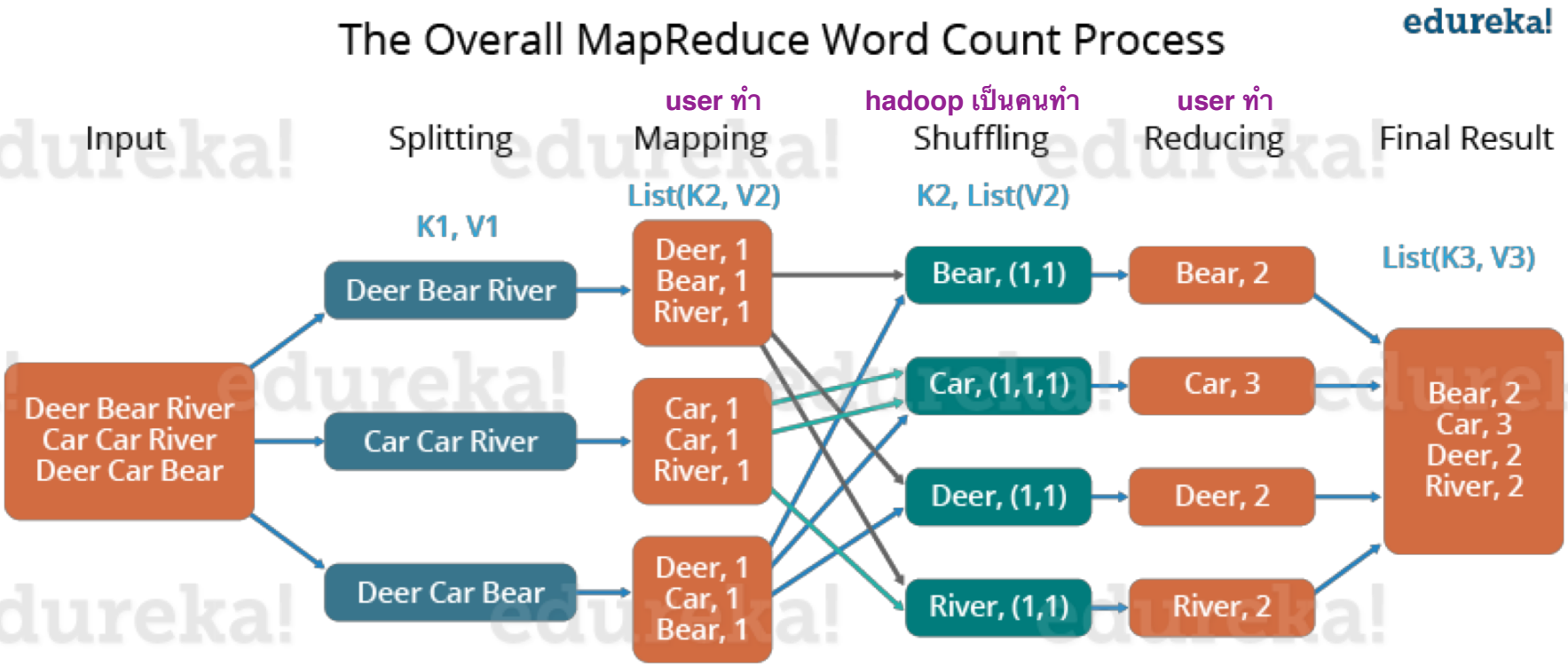
Source: Tomcy John and Pankaj Misra, "Data Lake for Enterprises," Packt Publishing, 2017

MapReduce Tutorial: A Word Count Example of MapReduce

- Text file: example.txt

Dear, Bear, River, Car, Car, River, Deer, Car and Bear

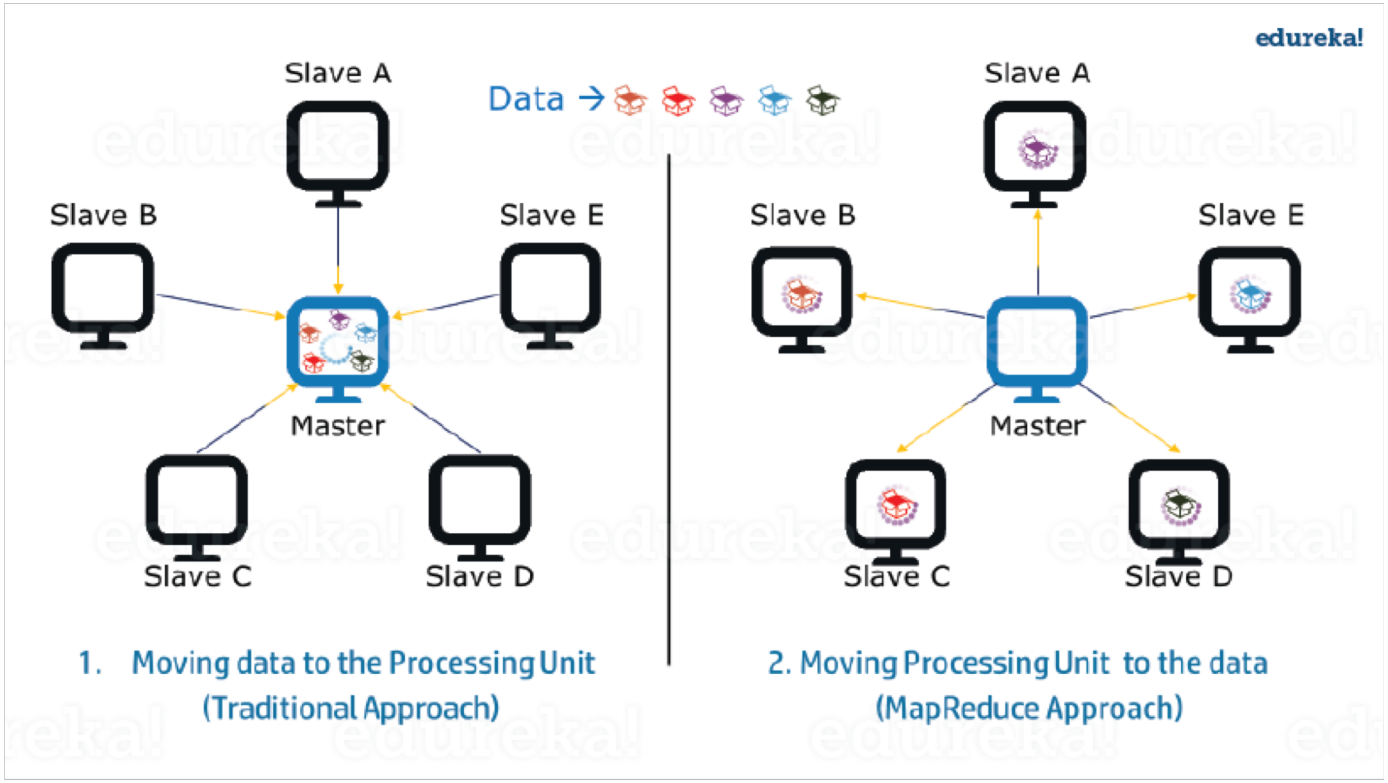
- Finding: number of occurrences of those unique words



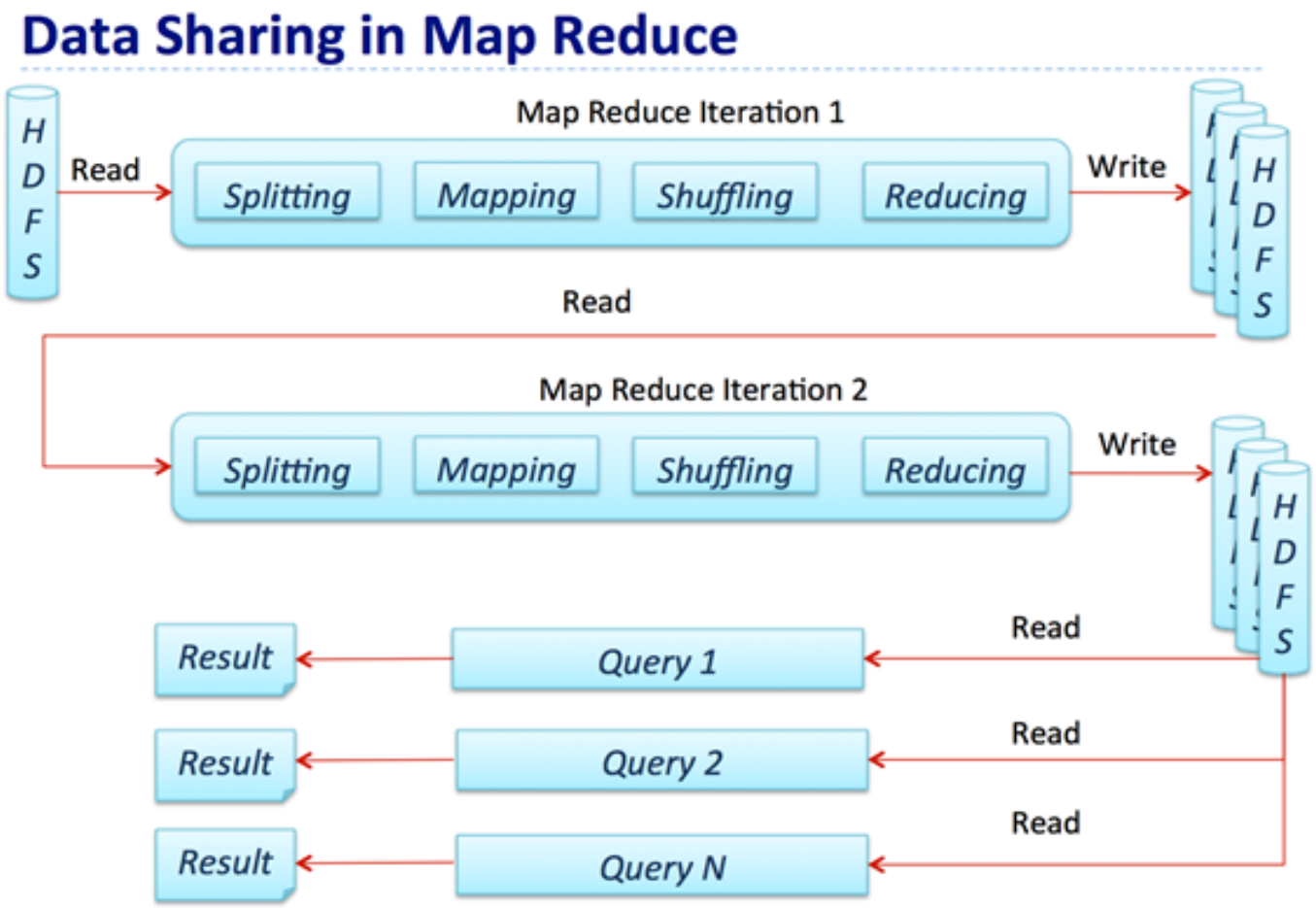
Source: edureka.co

MapReduce

- Parallel Processing
  - The time taken to process the data gets reduced by a tremendous amount
- Data Locality
  - Instead of moving data to the processing unit, we are moving processing unit to the data

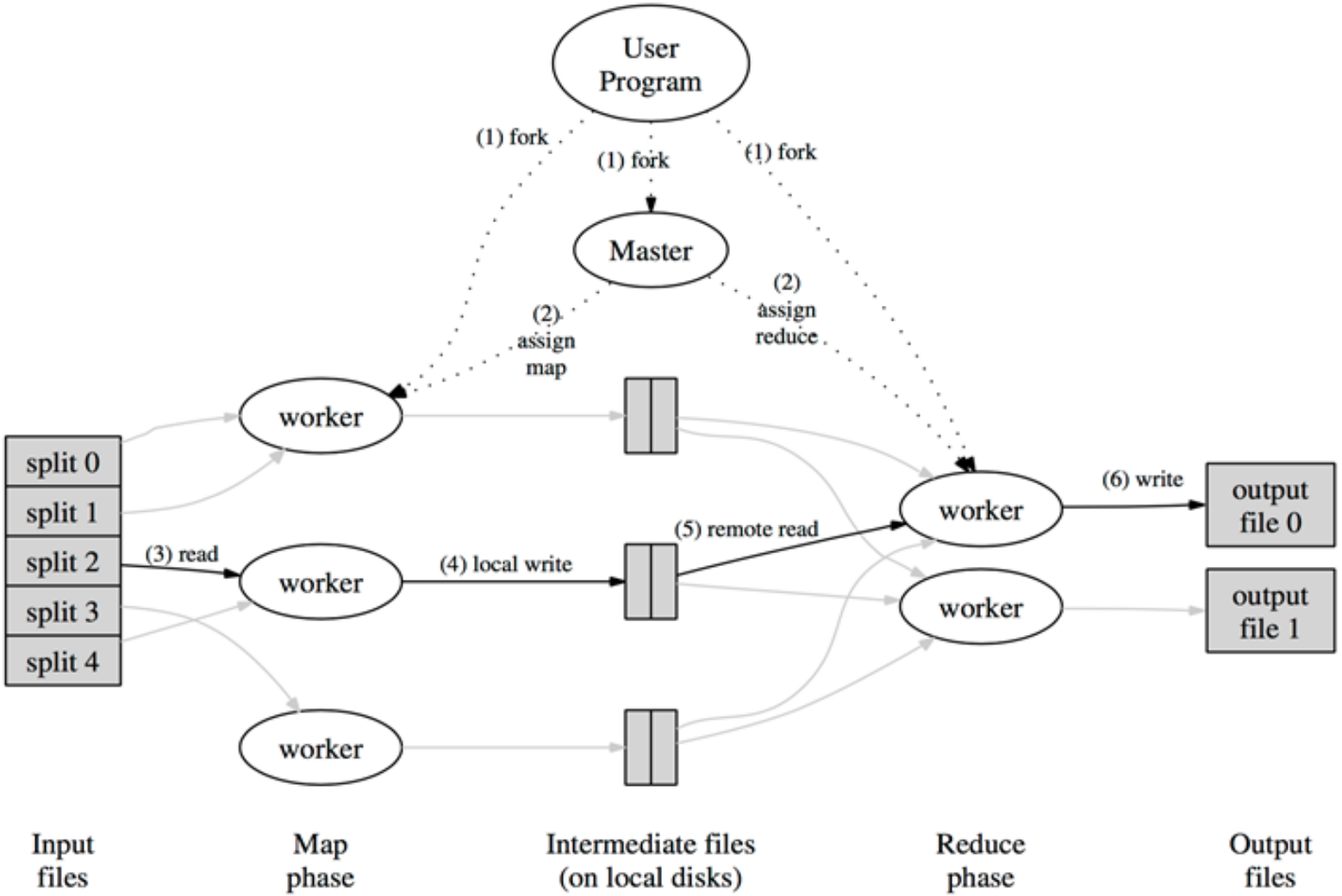


Typical Hadoop Job: Map-Reduce



Source: dzone.com

Typical Hadoop Job: Map-Reduce (Cont.)



Hands-On: Writing you MapReduce Program

Example MapReduce: WordCount

```
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

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

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

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

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

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

```
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
      System.err.println("Usage: wordcount <in> <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
    FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Running MapReduce Program

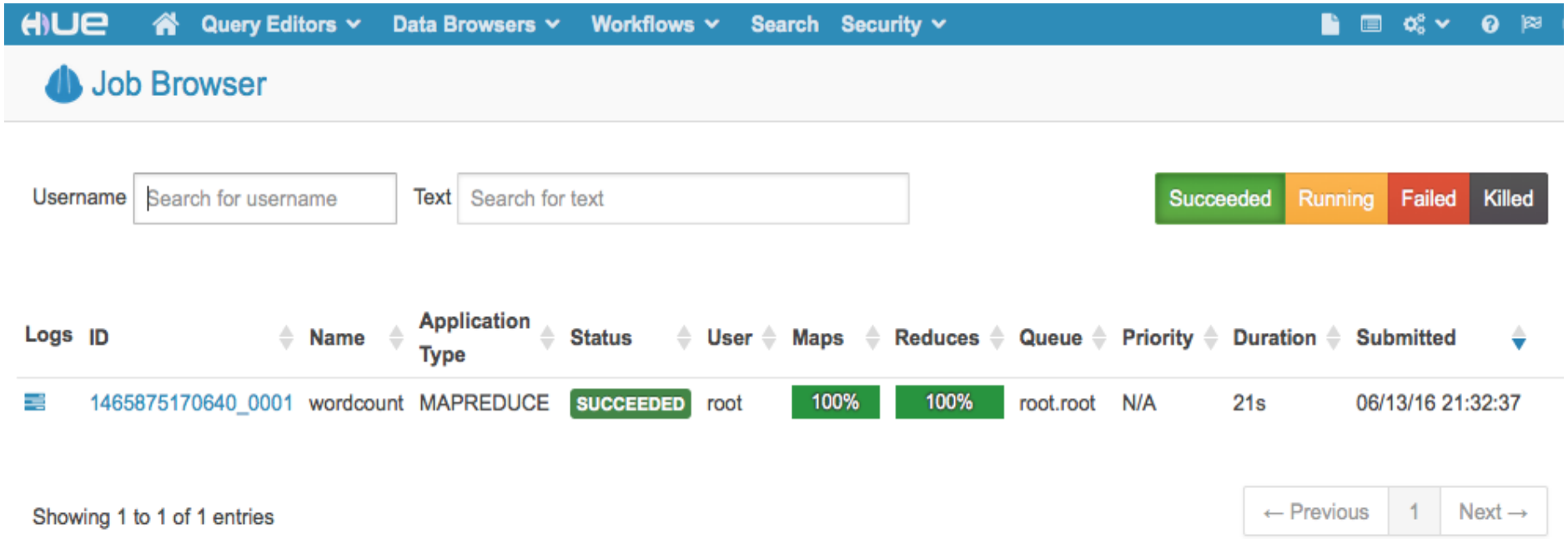
```
[cloudera@quickstart ~]$ cd workspace/
[cloudera@quickstart workspace]$ hadoop jar wordcount.jar org.myorg.WordCount input/*  output/wordcount_output
15/02/08 10:30:31 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/02/08 10:30:32 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/02/08 10:30:33 WARN mapreduce.JobSubmitter: Hadoop command-line option parsing not performed. Implement the Tool i
ation with ToolRunner to remedy this.
15/02/08 10:30:33 INFO mapred.FileInputFormat: Total input paths to process : 1
15/02/08 10:30:34 INFO mapreduce.JobSubmitter: number of splits:2
15/02/08 10:30:34 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1423408479621_0009
15/02/08 10:30:35 INFO impl.YarnClientImpl: Submitted application application_1423408479621_0009
15/02/08 10:30:35 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_142
15/02/08 10:30:35 INFO mapreduce.Job: Running job: job_1423408479621_0009
15/02/08 10:30:52 INFO mapreduce.Job: Job job_1423408479621_0009 running in uber mode : false
15/02/08 10:30:52 INFO mapreduce.Job:  map 0% reduce 0%
15/02/08 10:31:22 INFO mapreduce.Job:  map 58% reduce 0%
15/02/08 10:31:25 INFO mapreduce.Job:  map 100% reduce 0%
15/02/08 10:31:52 INFO mapreduce.Job:  map 100% reduce 100%
15/02/08 10:31:53 INFO mapreduce.Job: Job job_1423408479621_0009 completed successfully
15/02/08 10:31:53 INFO mapreduce.Job: Counters: 49
```

$wget -O wordcount.jar 'https://drive.google.com/uc?export=download&id=1kn4GSjxe8dvaYOrKqUPqJwwY1HnoZ2BK'

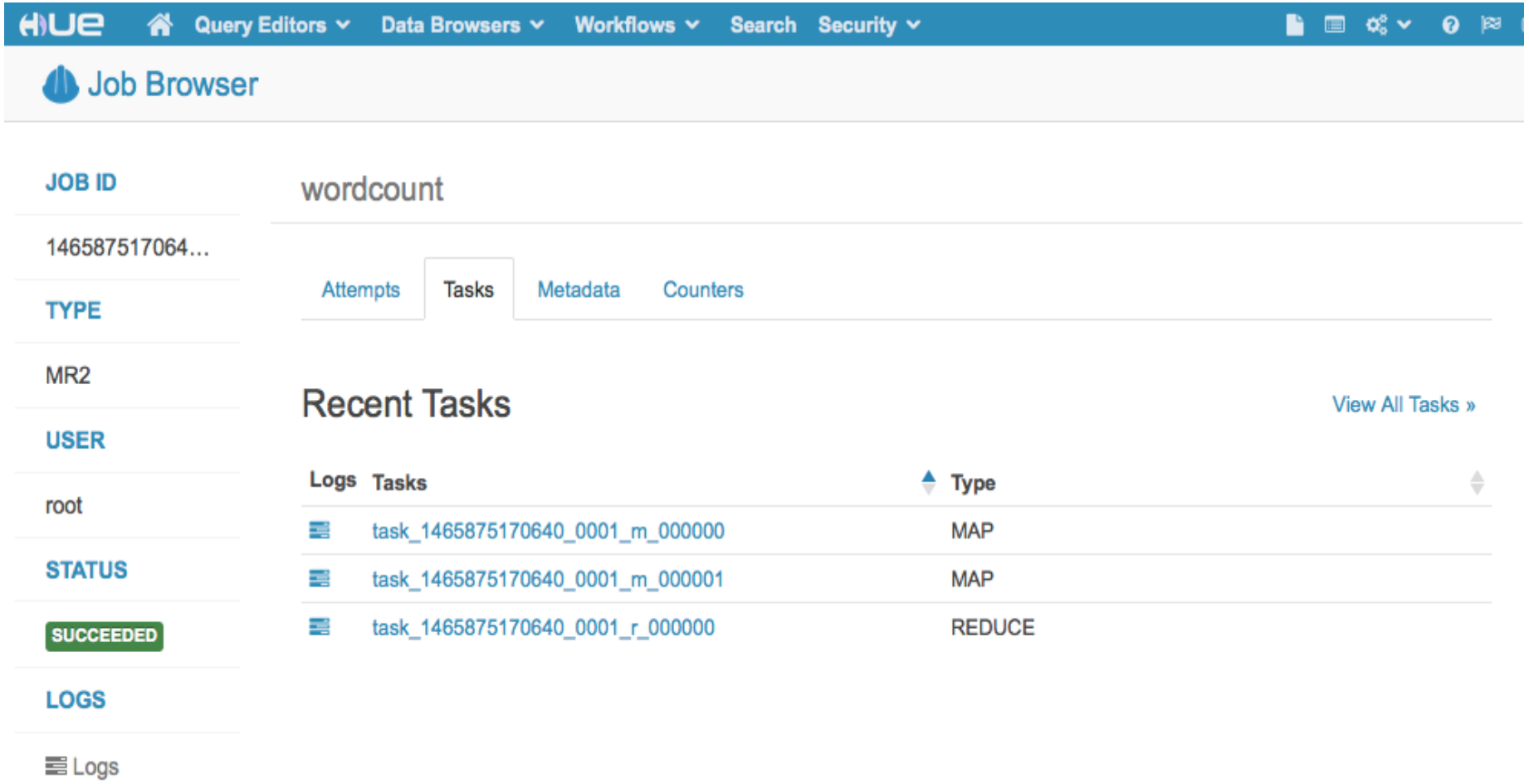
$ hdfs dfs -put wordcount.jar /user/cloudera/input/

$hadoop jar wordcount.jar org.myorg.WordCount /user/cloudera/input/* /user/cloudera/output/wordcount

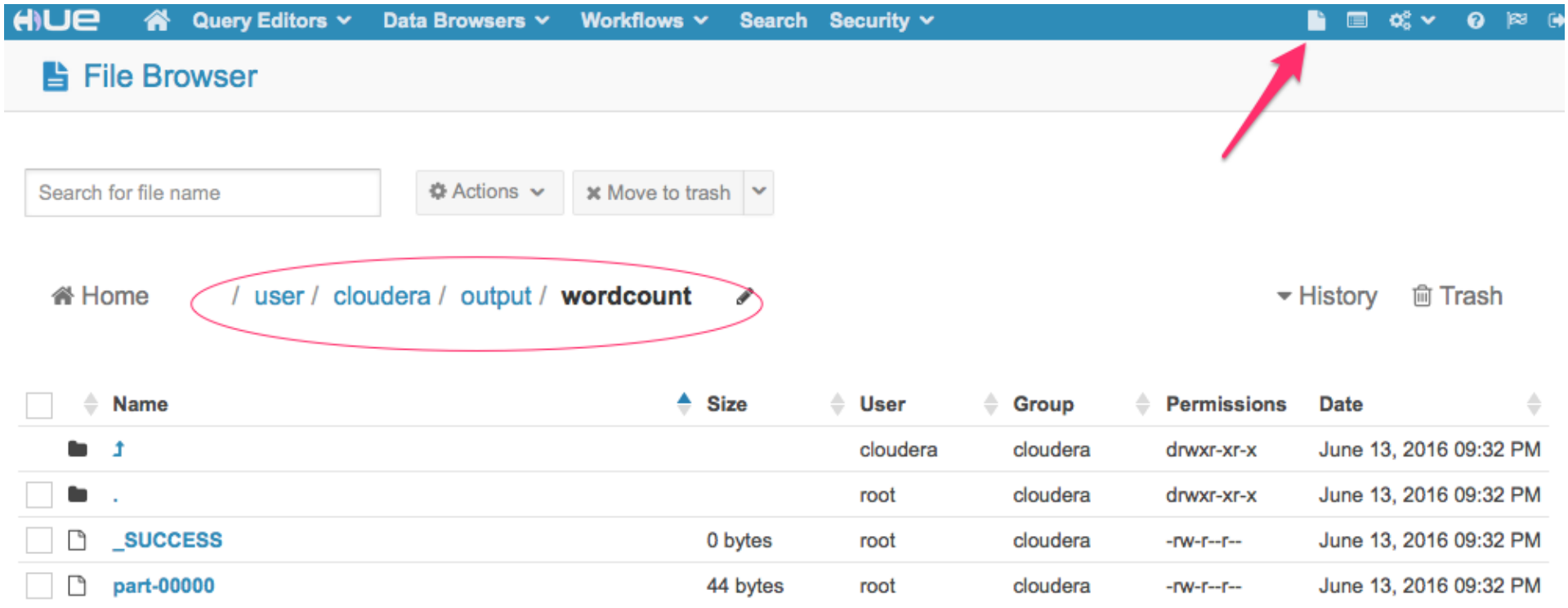
Reviewing MapReduce Job in Hue



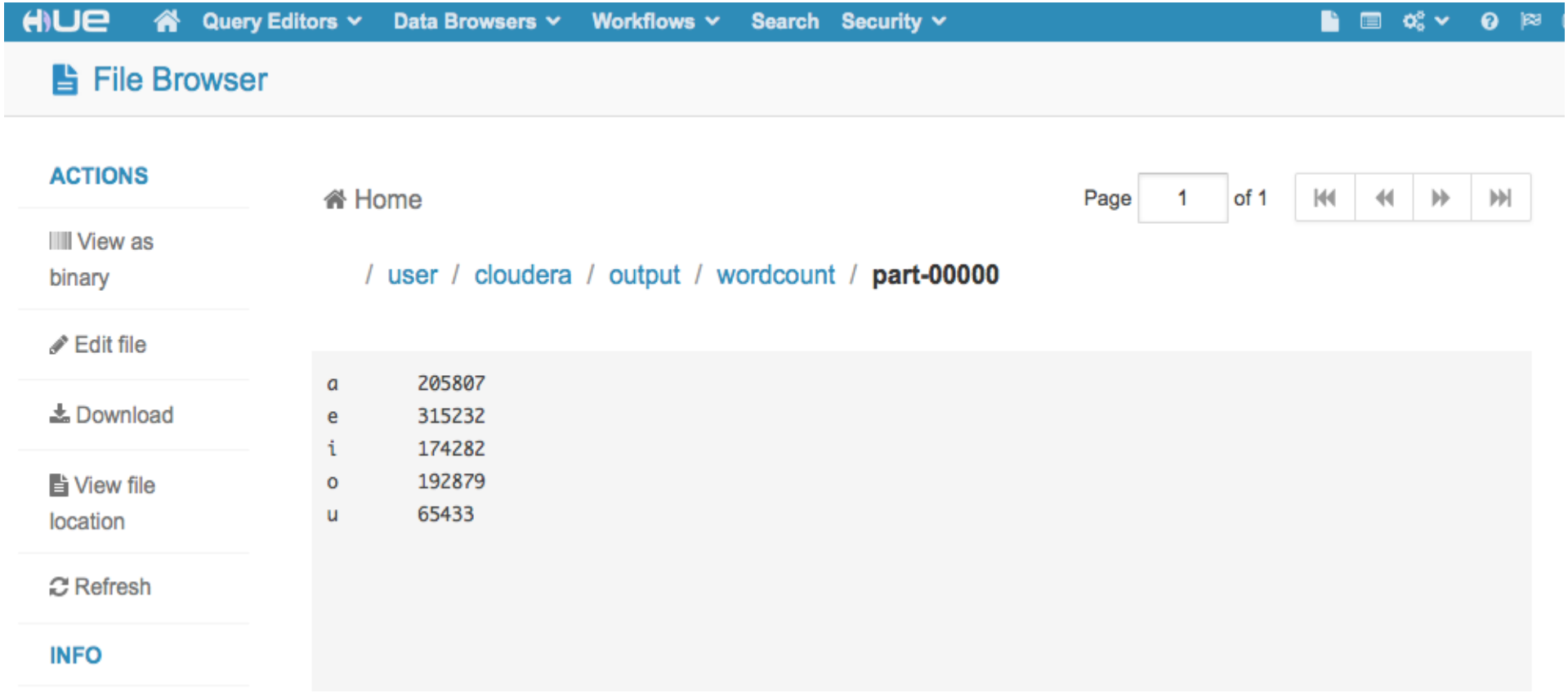
Reviewing MapReduce Job in Hue



Reviewing MapReduce Output Result



Reviewing MapReduce Output Result



Lecture
Understanding Oozie

Introduction

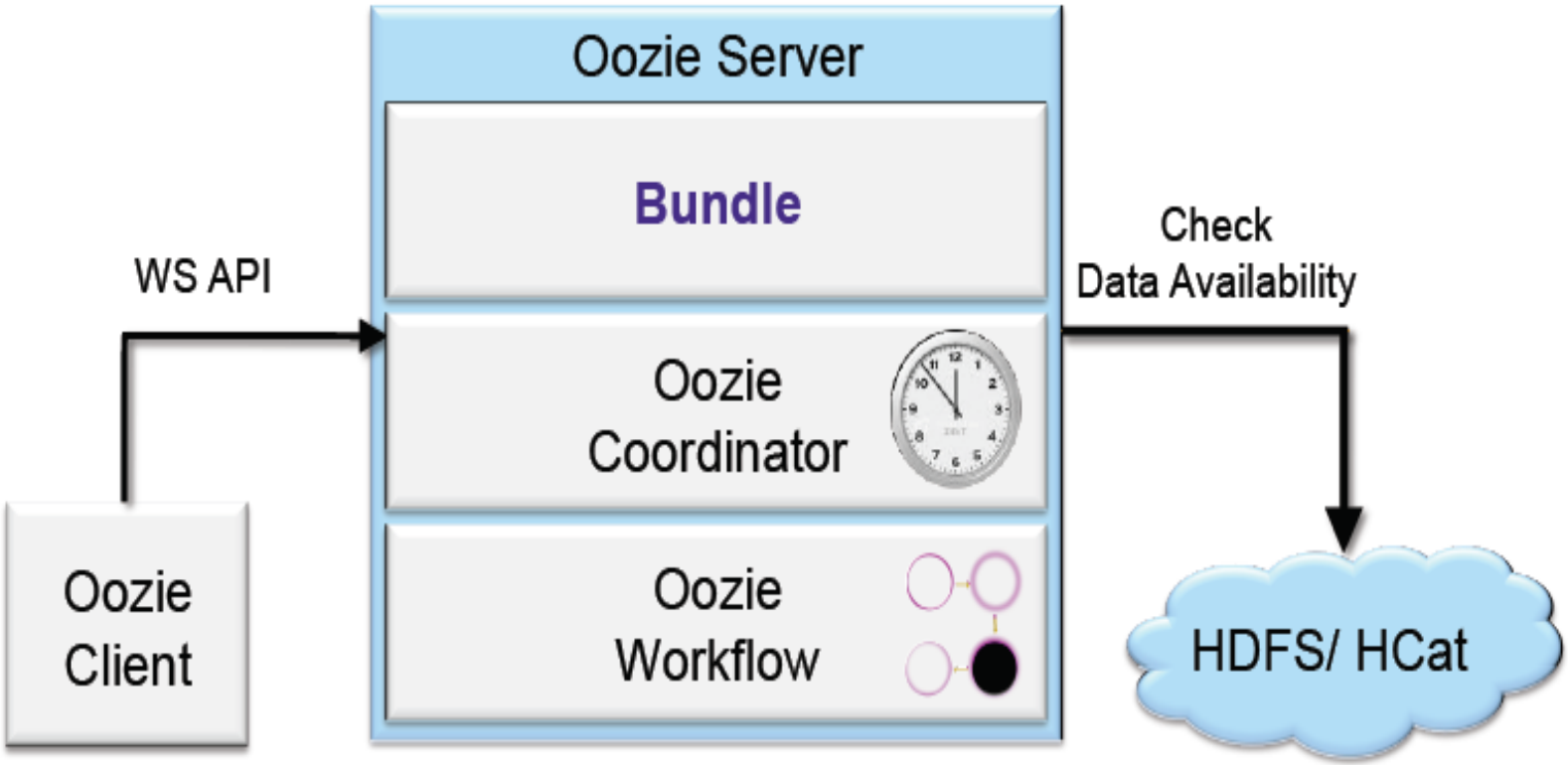
Workslow scheduler for Hadoop

Oozie is a workflow scheduler system to manage Apache Hadoop jobs. Oozie is integrated with the rest of the Hadoop stack supporting several types of Hadoop jobs out of the box (such as Java map-reduce, Streaming map-reduce, Pig, Hive, Sqoop and Distcp) as well as system specific jobs (such as Java programs and shell scripts).

What is Oozie?

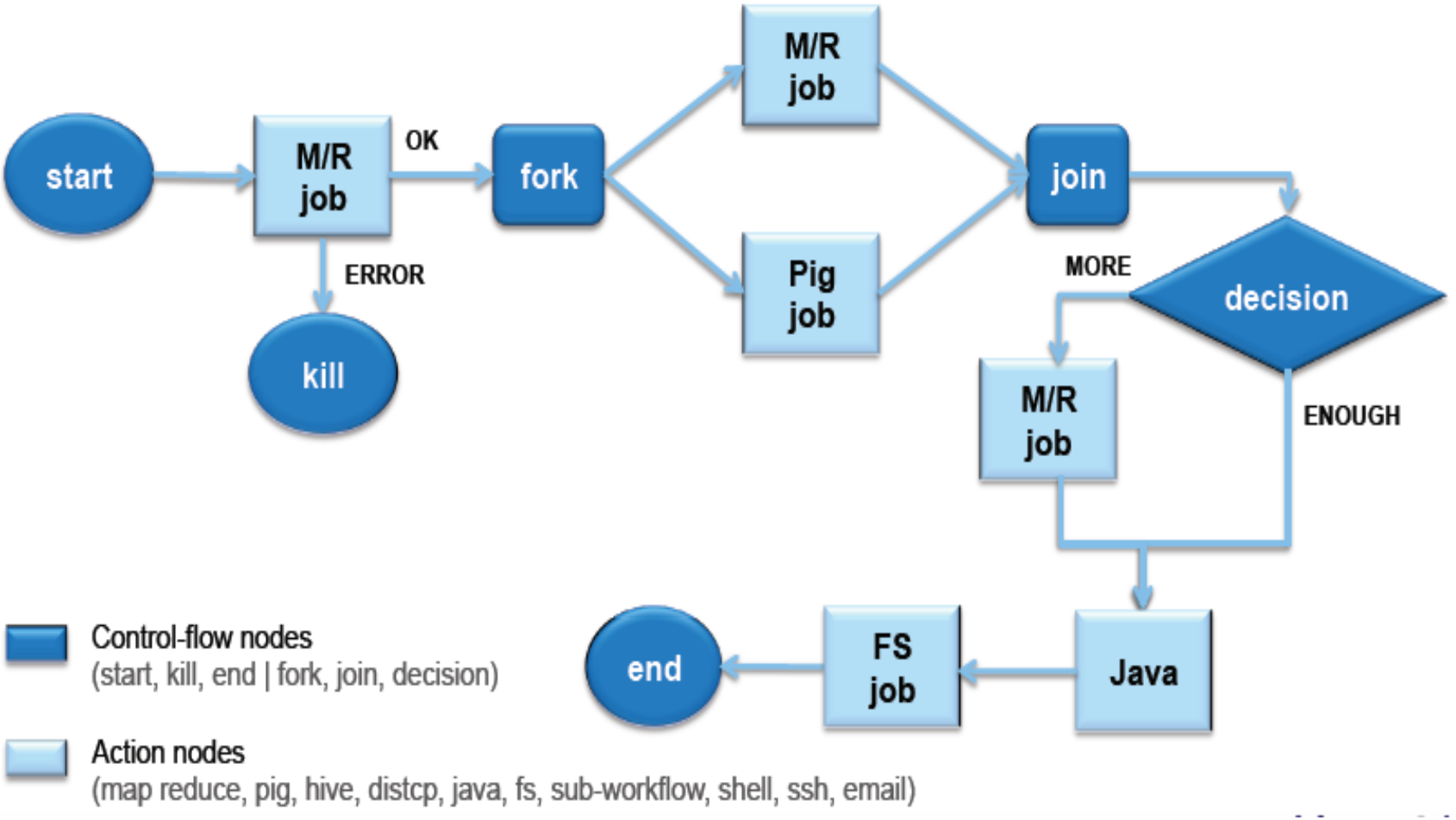
- Work flow scheduler for Hadoop
- Manages Hadoop Jobs
- Integrated with many Hadoop apps i.e. Pig, Hive
- Scaleable
- Schedule jobs
- A work flow is a collection of actions.
- A work flow is
  - Arranged as a DAG (direct acyclic graph)
  - Graph stored as hPDL (XML process definition)

Oozie Server



Source: Oozie – Now and Beyond, Yahoo, 2013

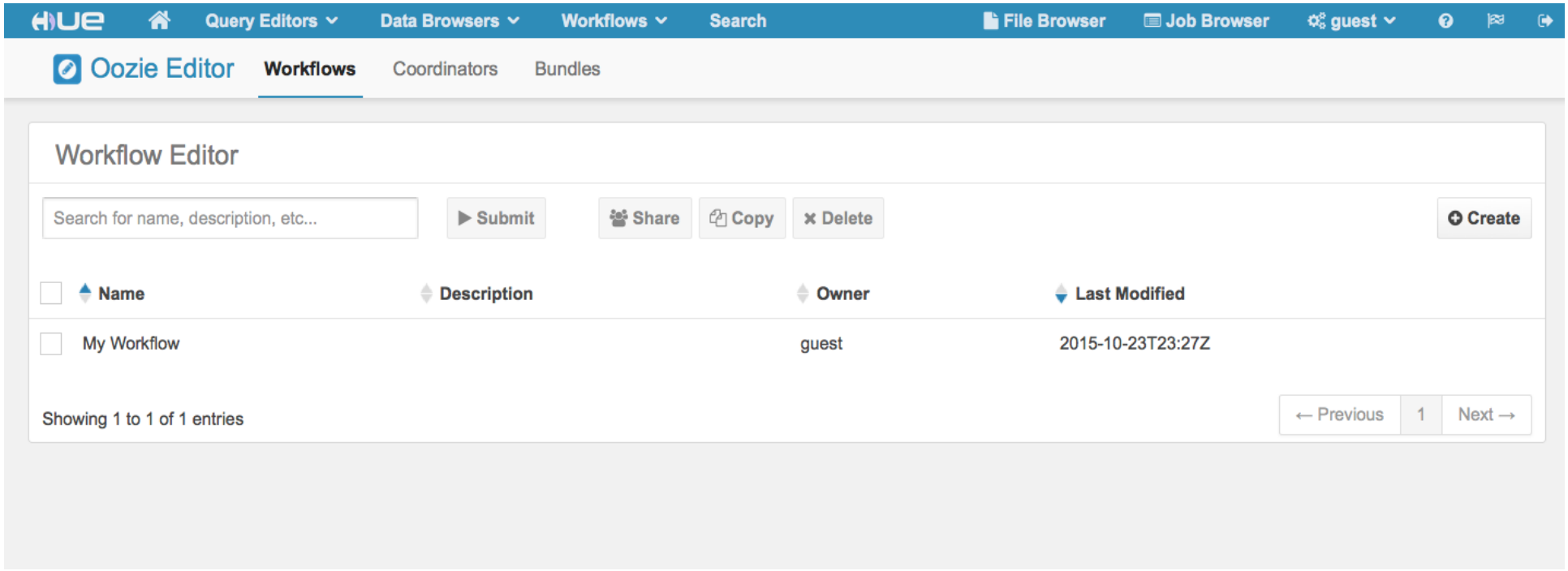
Oozie Workflow



Source: Oozie – Now and Beyond, Yahoo, 2013

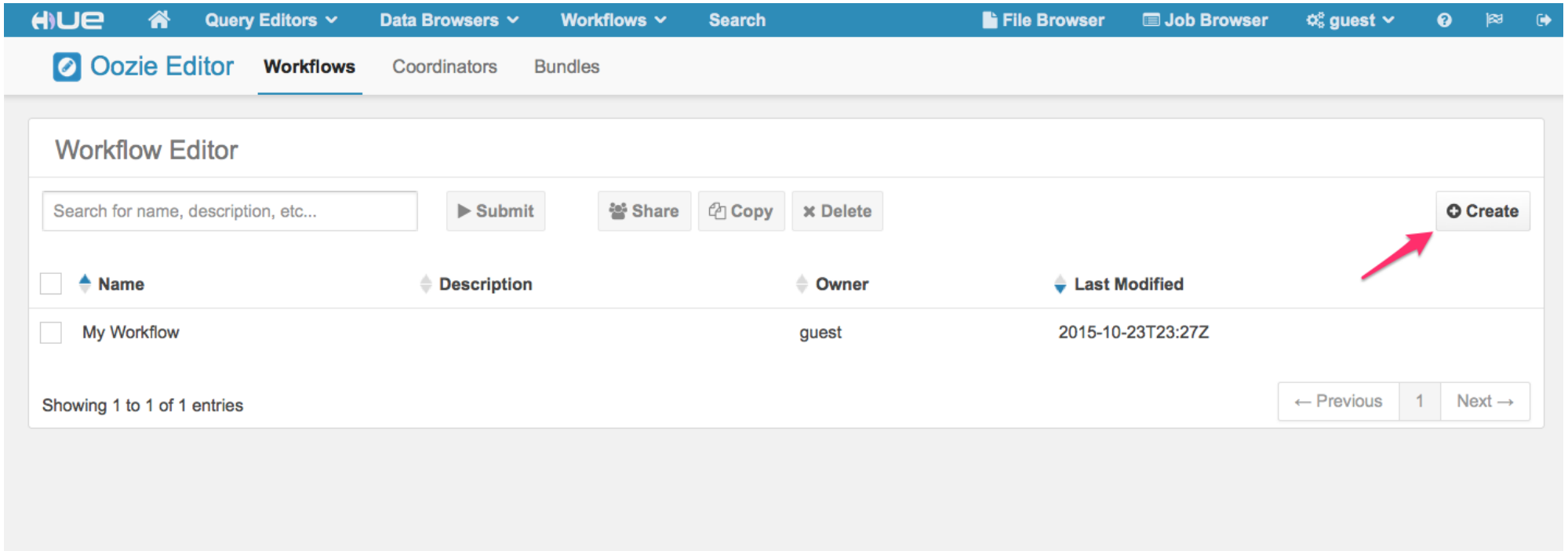
Hands-On: Running Map Reduce using Oozie workflow

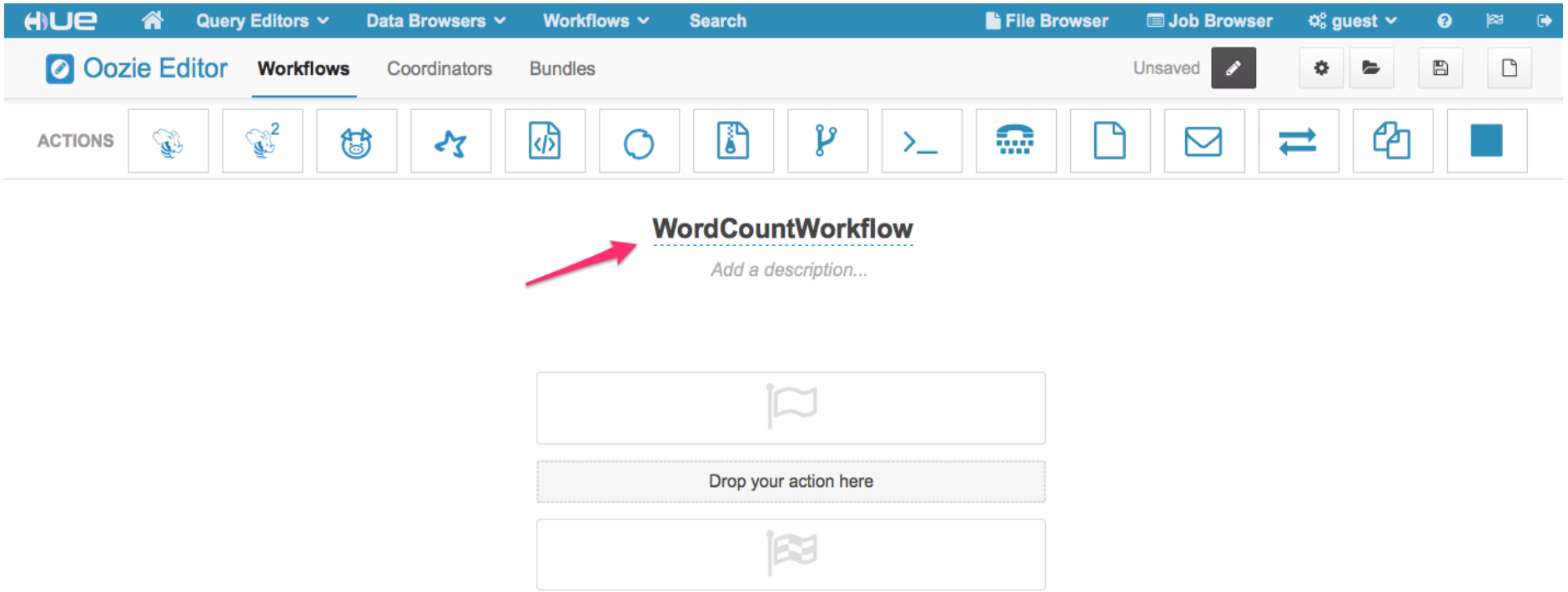
Using Hue: select WorkFlows >> Editors >> Workflows



Create a new workflow

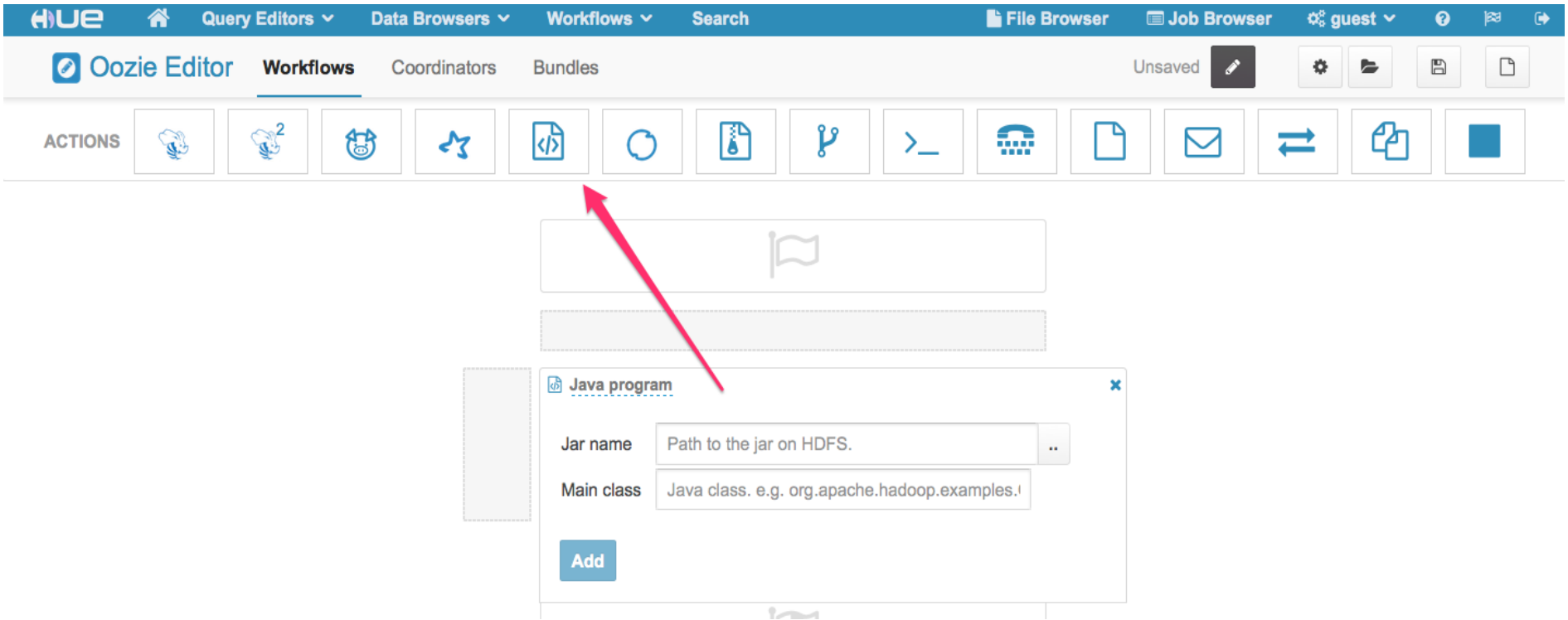
- Click Create button; the following screen will be displayed
- Name the workflow as WordCountWorkflow




HUE
Query Editors
Data Browsers
Workflows
Search
File Browser
Job Browser
guest
Oozie Editor
Workflows
Coordinators
Bundles
Unsaved
ACTIONS
WordCountWorkflow
Add a description...
Drop your action here

Select a Java job for the workflow

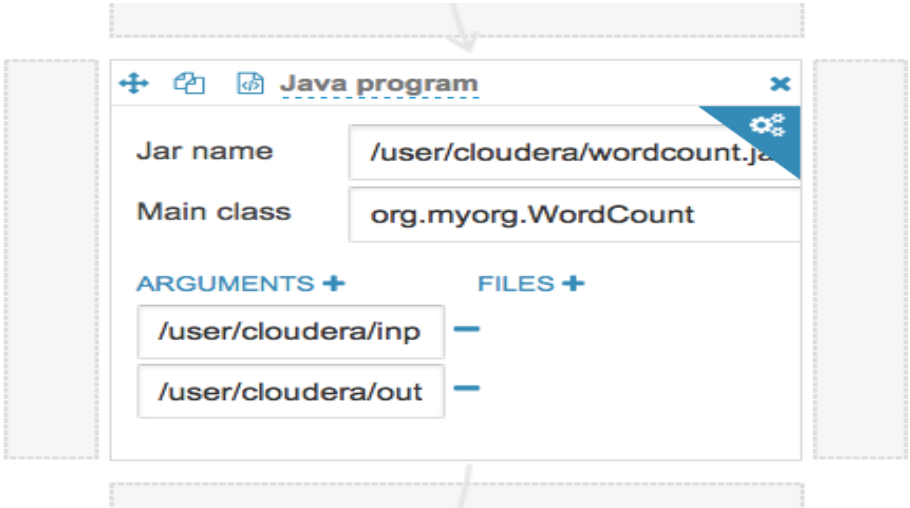
From the Oozie editor, drag **Java Program** and drop between start and end



Edit the Java Job

Assign the following value

- Jar name: wordcount.jar (select … choose upload from local machine)
- Main Class: org.myorg.WordCount
- Arguments: /user/cloudera/input/*
- /user/cloudera/output1/wordcount



Submit the workflow

- Click Save
- Then click submit