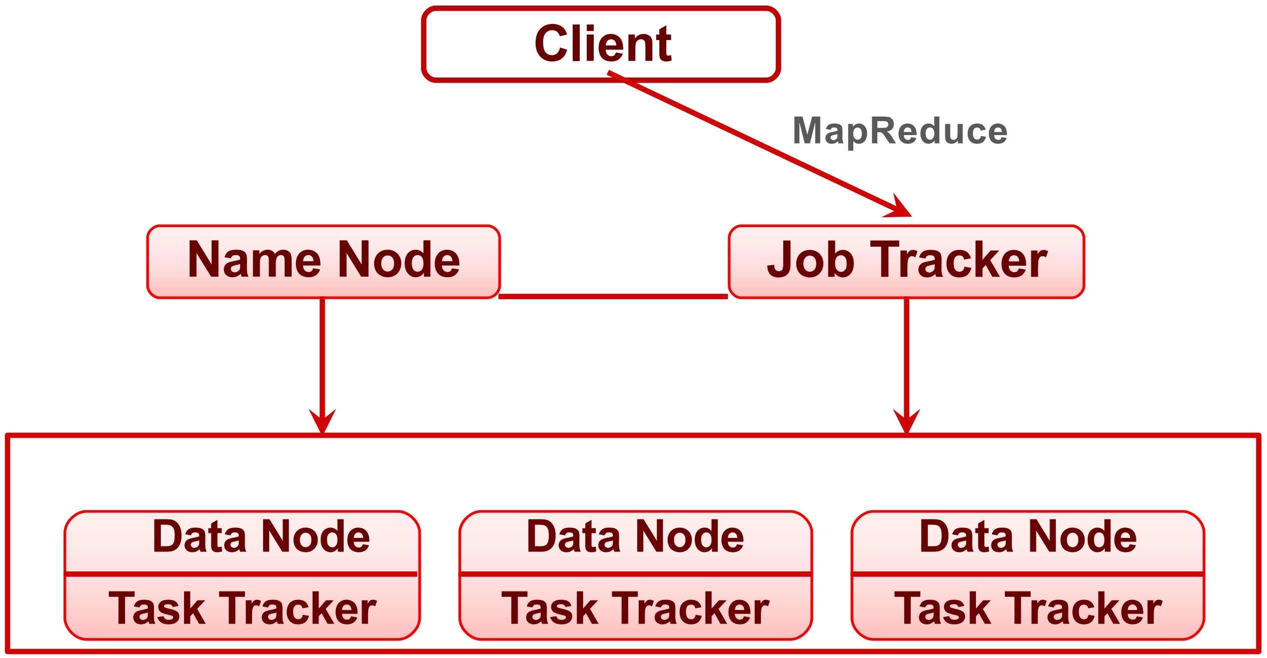
Module 4_2

Understanding MapReduce and Ozzie

Thanachart Numnonda, Executive Director, IMC Institute

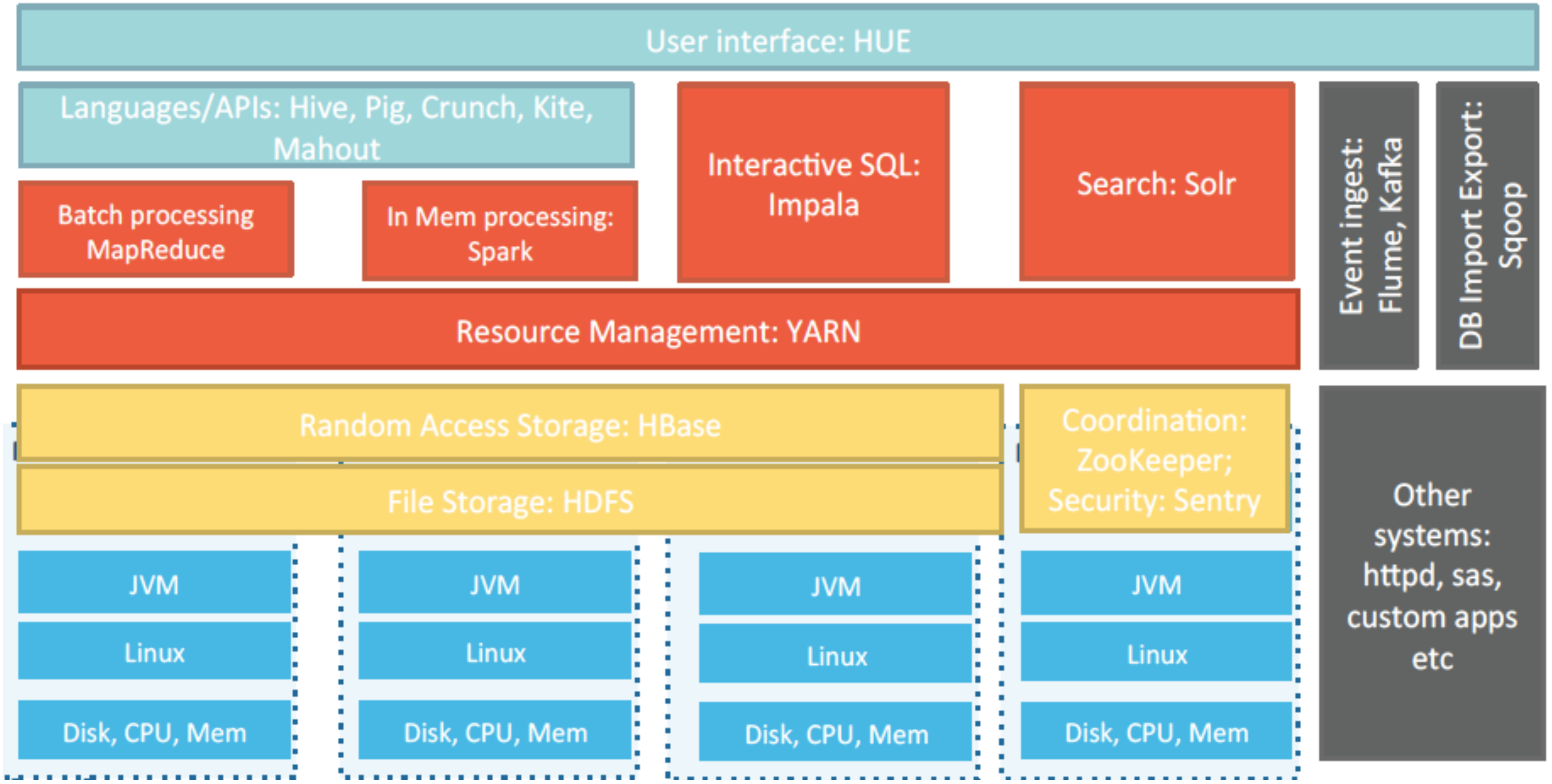
Thanisa Numnonda, Faculty of Information Technology,

King Mongkut’s Institute of Technology Ladkrabang



Lecture: Understanding MapReduce Processing

Hadoop Ecosystem



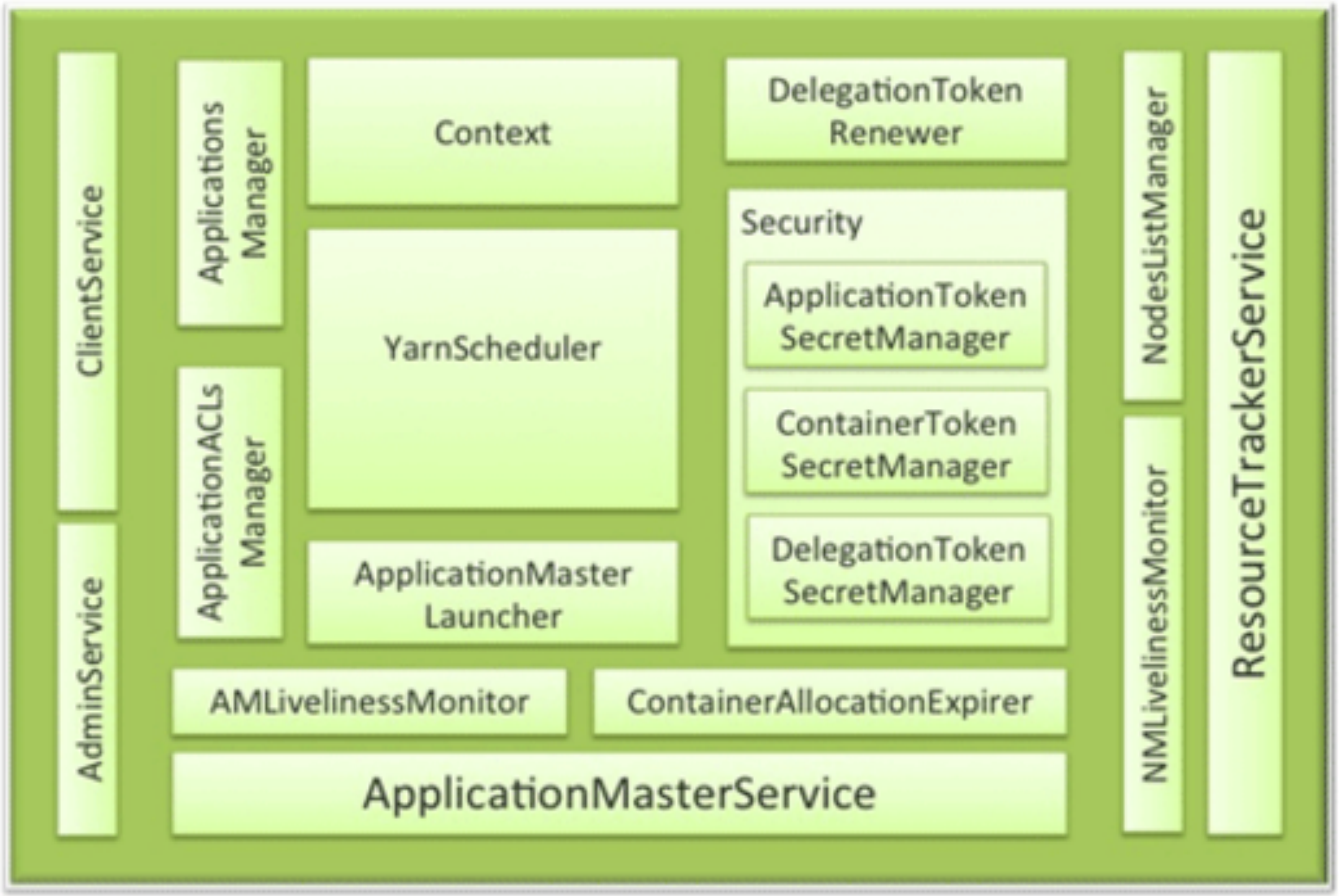
User interface: HUE
Languages/APIs: Hive, Pig, Crunch, Kite, Mahout
Batch processing MapReduce
In Mem processing: Spark
Interactive SQL: Impala
Search: Solr
Event ingest: Flume, Kafka
DB Import Export: Sqoop
Resource Management: YARN
Random Access Storage: HBase
File Storage: HDFS
Coordination: ZooKeeper; Security: Sentry
Other systems: httpd, sas, custom apps etc
JVM
Linux
Disk, CPU, Mem
JVM
Linux
Disk, CPU, Mem
JVM
Linux
Disk, CPU, Mem
JVM
Linux
Disk, CPU, Mem

Source: Apache Hadoop Operations for Production Systems, Cloudera

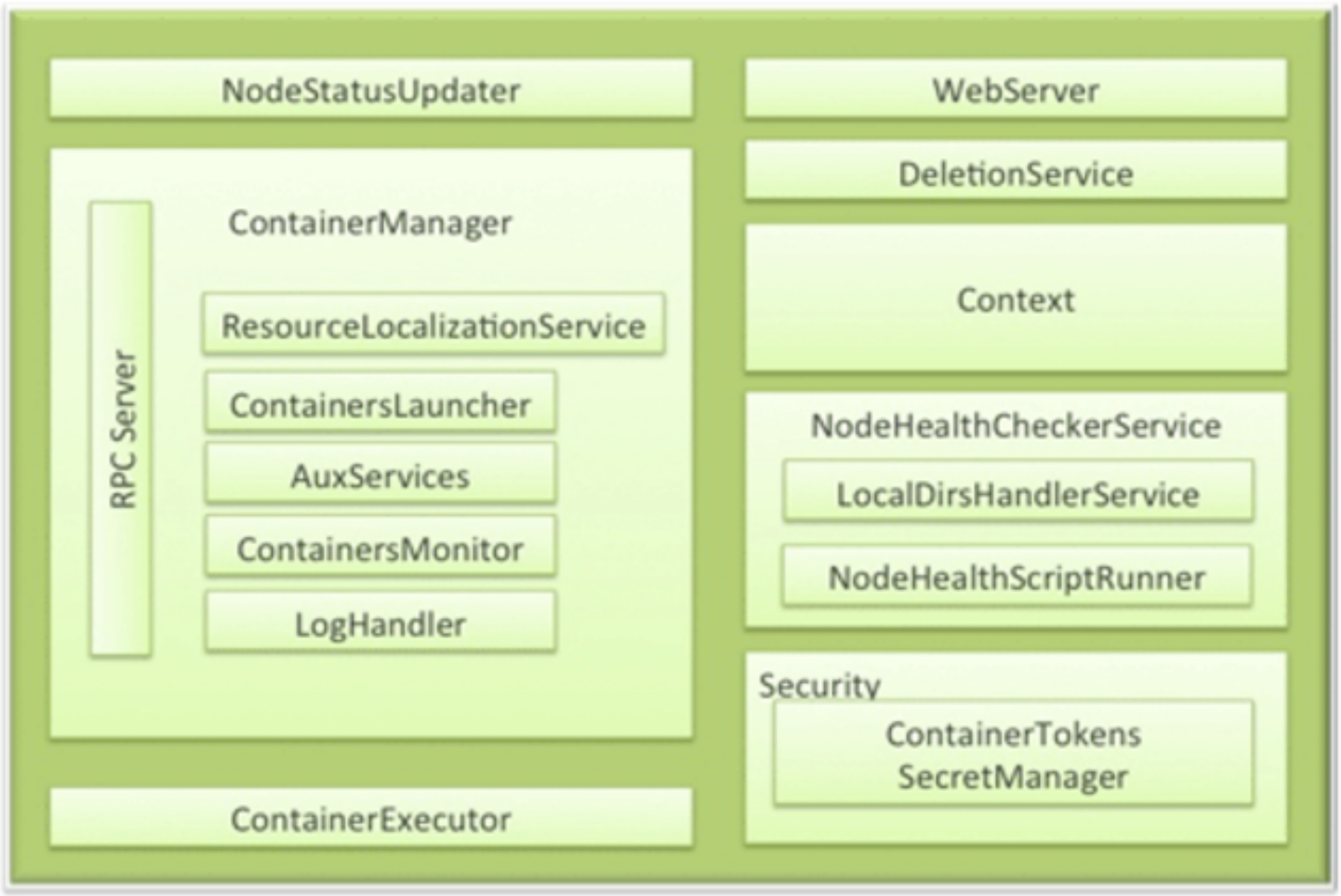
Yarn Components

- Resource Manager
  - a pluggable scheduler > primarily limited to scheduling
  - ApplicationManager >> manages user jobs on the cluster
- Node Manager
  - Containers
  - manages users' jobs and workflow on a given node
- Application Master
  - a user job life-cycle manager

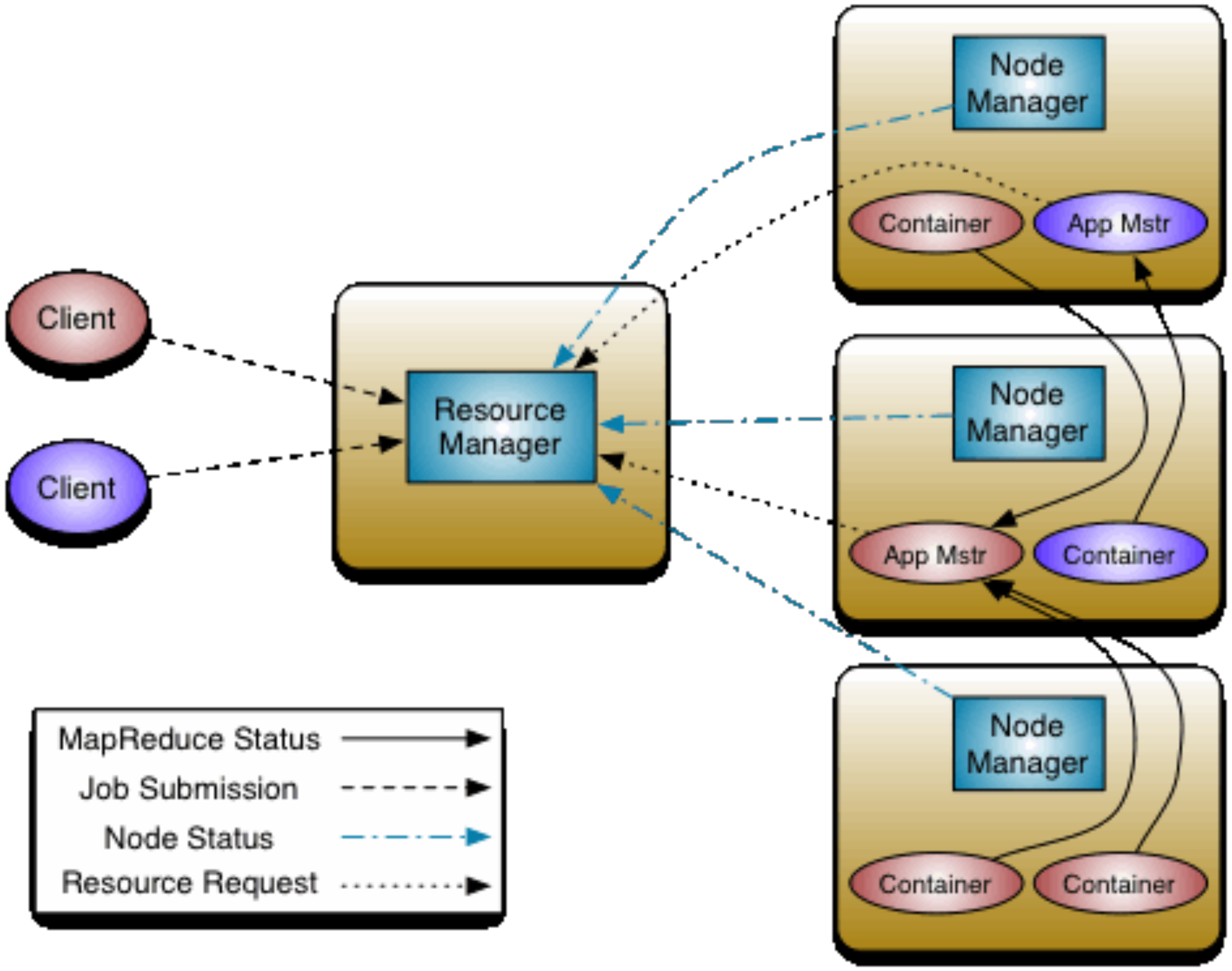
Resource Manager



Node Manager

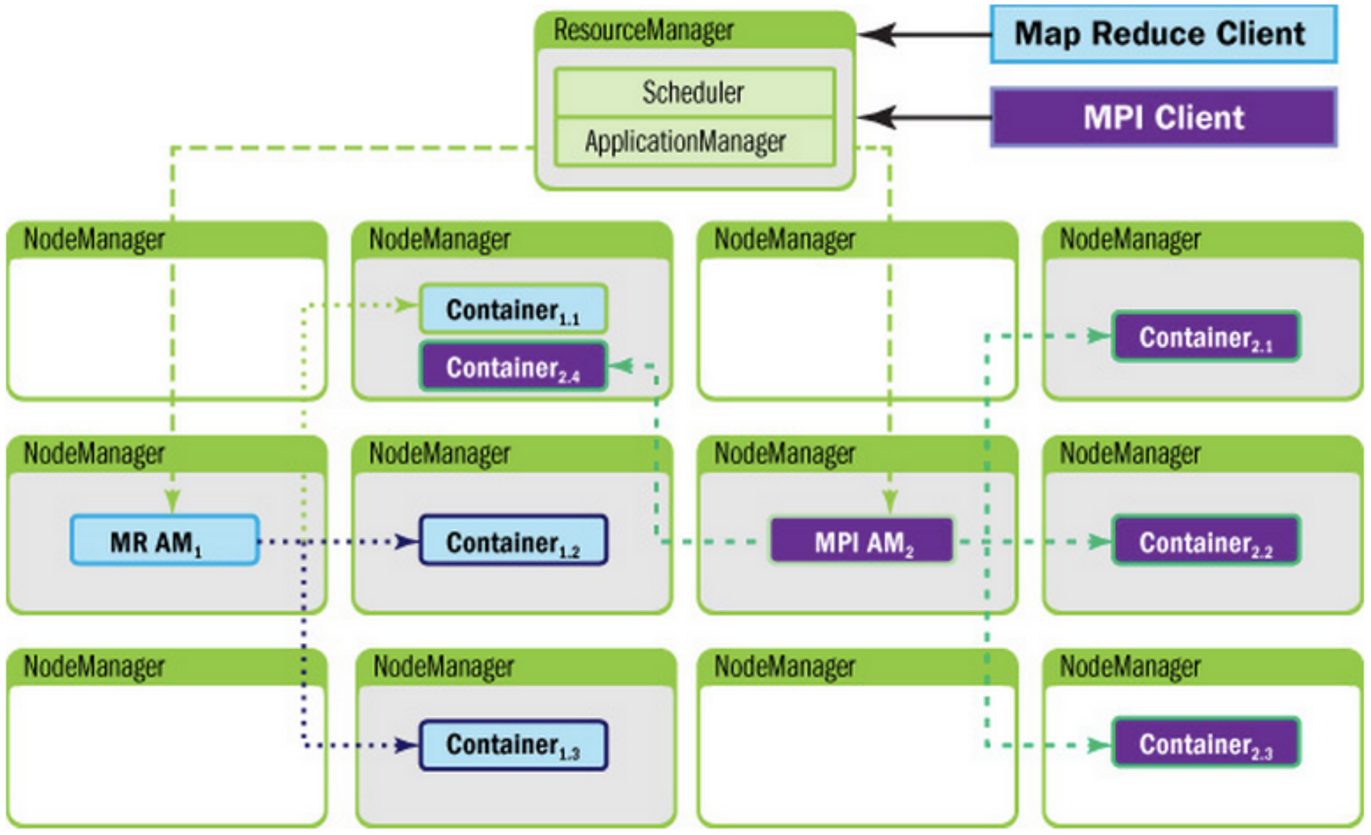


YARN: – CONCEPTS APPLICATIONS



Source: Apache Hadoop™ YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop™ 2

YARN: – CONCEPTS APPLICATIONS



Source: Apache Hadoop™ YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop™ 2

Yarn Workloads

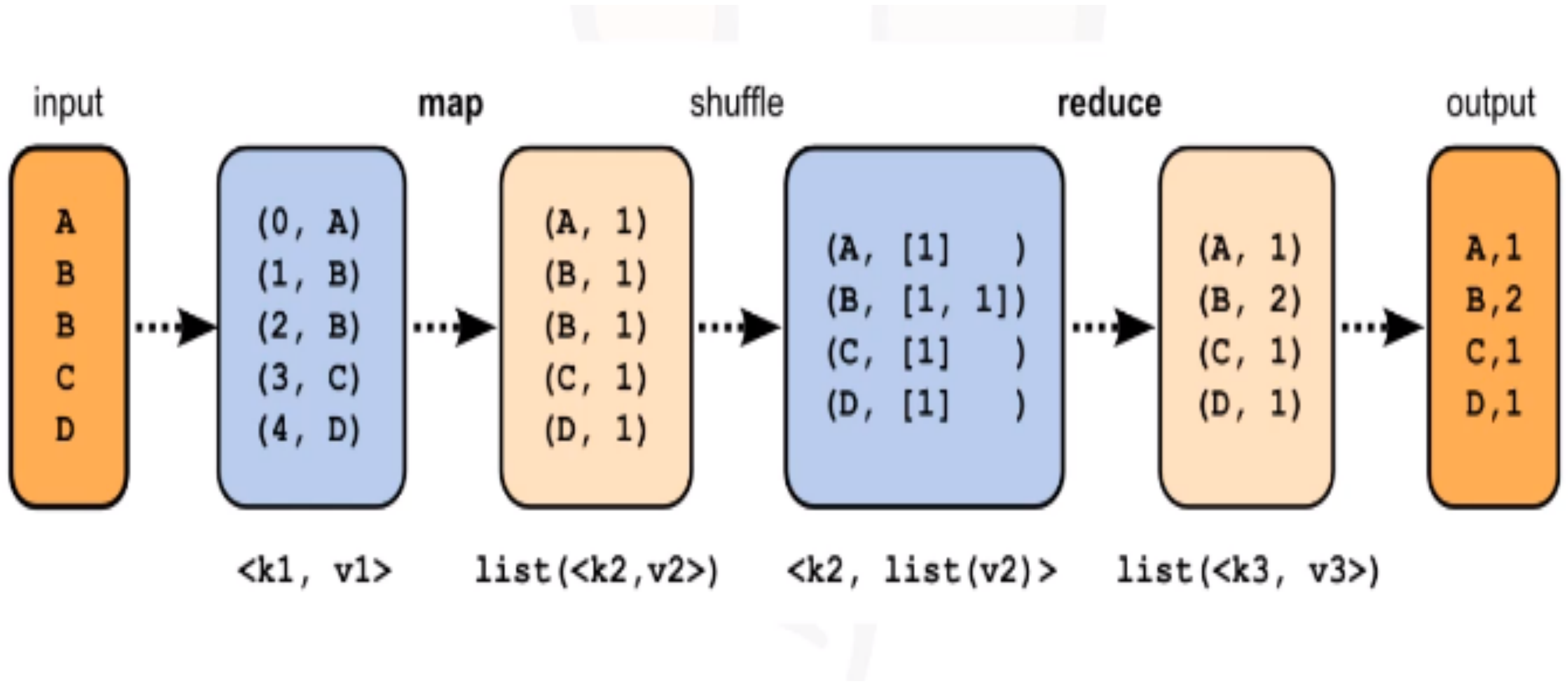
- **Batch:** MapReduce that is the compatible with Hadoop 1.x
- **Script:** Pig
- **Interactive SQL**: Hive or Tez
- **NoSQL:** HBase and Accumulo
- **Streaming:** Storm
- **In-memory:** Spark
- **Search:** SOLR

Before MapReduce…

- Large scale data processing was difficult!
  - Managing hundreds or thousands of processors
  - Managing parallelization and distribution
  - I/O Scheduling
  - Status and monitoring
  - Fault/crash tolerance
- MapReduce provides all of these, easily!

Source: http://labs.google.com/papers/mapreduce-osdi04-slides/index-auto-0002.html

MapReduce Framework



Source: www.bigdatauniversity.com

How Map and Reduce Work Together

- Map returns information
- Reduces accepts information
- Reduce applies a user defined function to reduce the amount of data

Map Abstraction

- Inputs a key/value pair
  - Key is a reference to the input value
  - Value is the data set on which to operate
- Evaluation
  - Function defined by user
  - Applies to every value in value input
    - Might need to parse input
- Produces a new list of key/value pairs
  - Can be different type from input pair

Reduce Abstraction

- Starts with intermediate Key / Value pairs
- Ends with finalized Key / Value pairs

- Starting pairs are sorted by key
- Iterator supplies the values for a given key to the Reduce function.

Reduce Abstraction

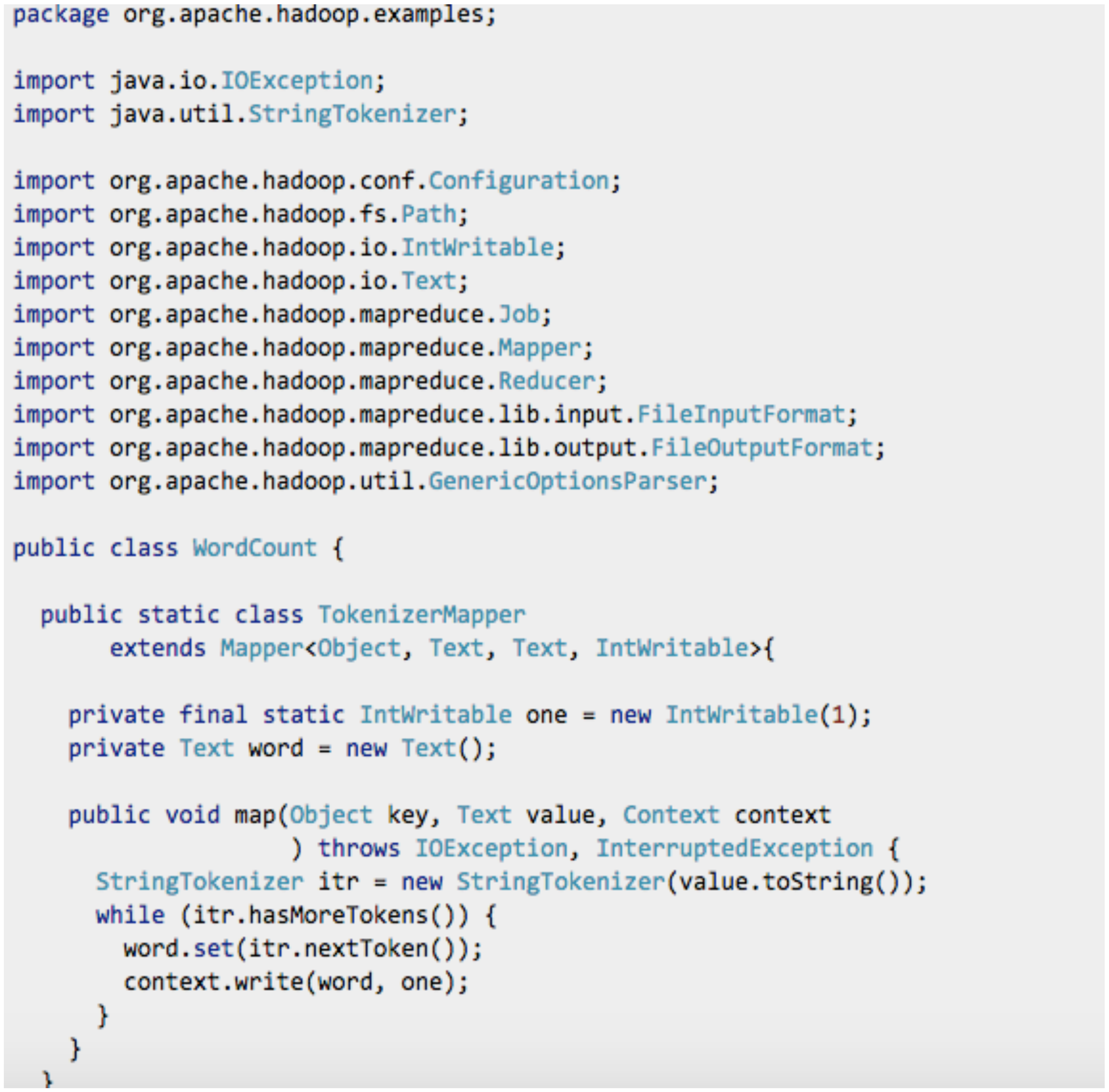
- Typically a function that:
  - Starts with a large number of key/value pairs
    - One key/value for each word in all files being greped (including multiple entries for the same word)
  - Ends with very few key/value pairs
    - One key/value for each unique word across all the files with the number of instances summed into this entry
- Broken up so a given worker works with input of the same key.

Why is this approach better?

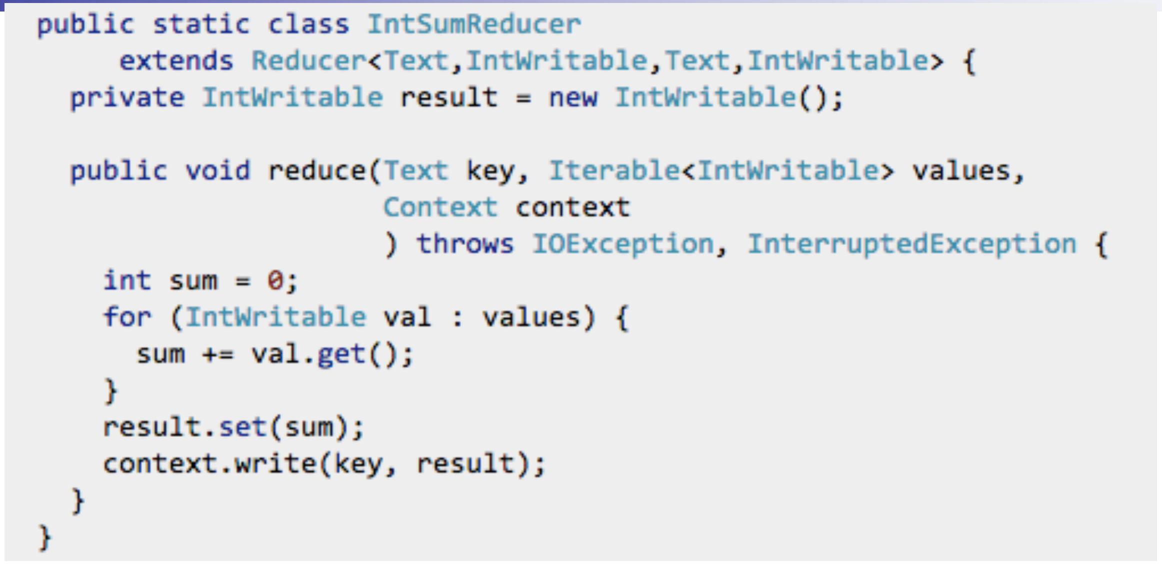
- Creates an abstraction for dealing with complex overhead
  - The computations are simple, the overhead is messy
- Removing the overhead makes programs much smaller and thus easier to use
  - Less testing is required as well. The MapReduce libraries can be assumed to work properly, so only user code needs to be tested
- Division of labor also handled by the MapReduce libraries, so programmers only need to focus on the actual computation

Hands-On: Writing you MapReduce Program

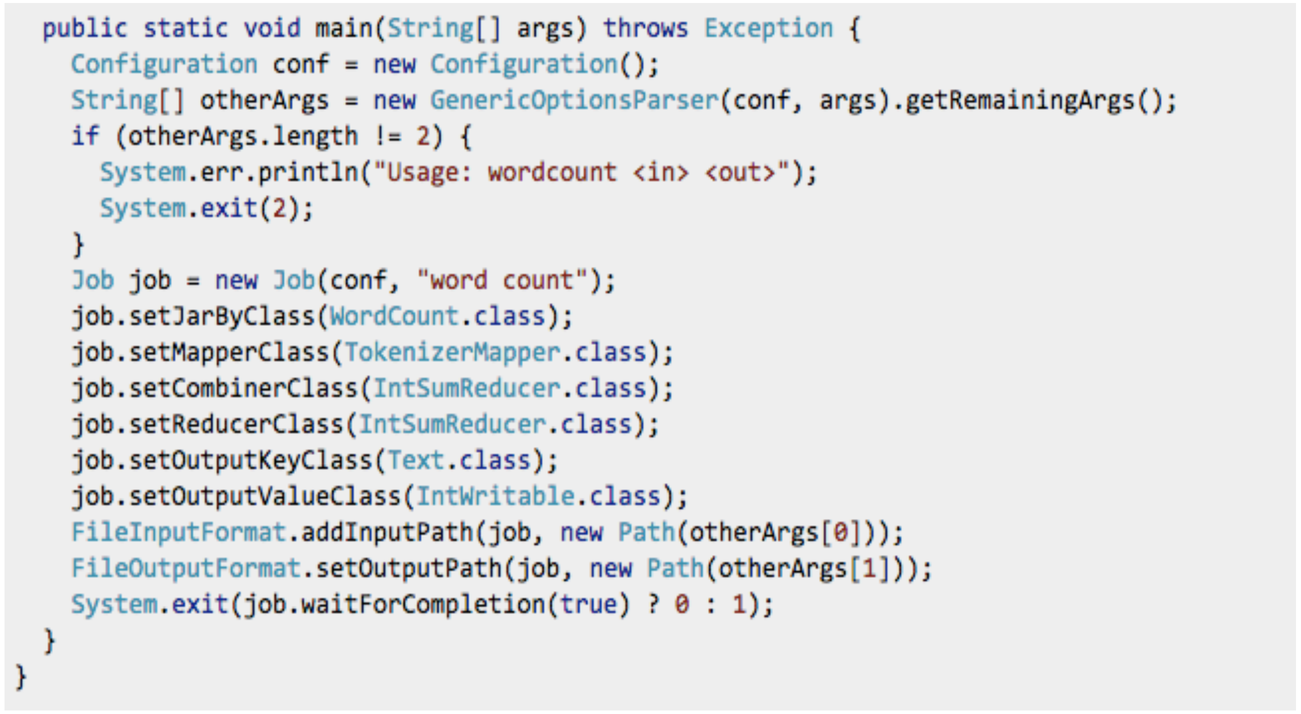
Example MapReduce: WordCount



```
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);
      }
    }
  }
```

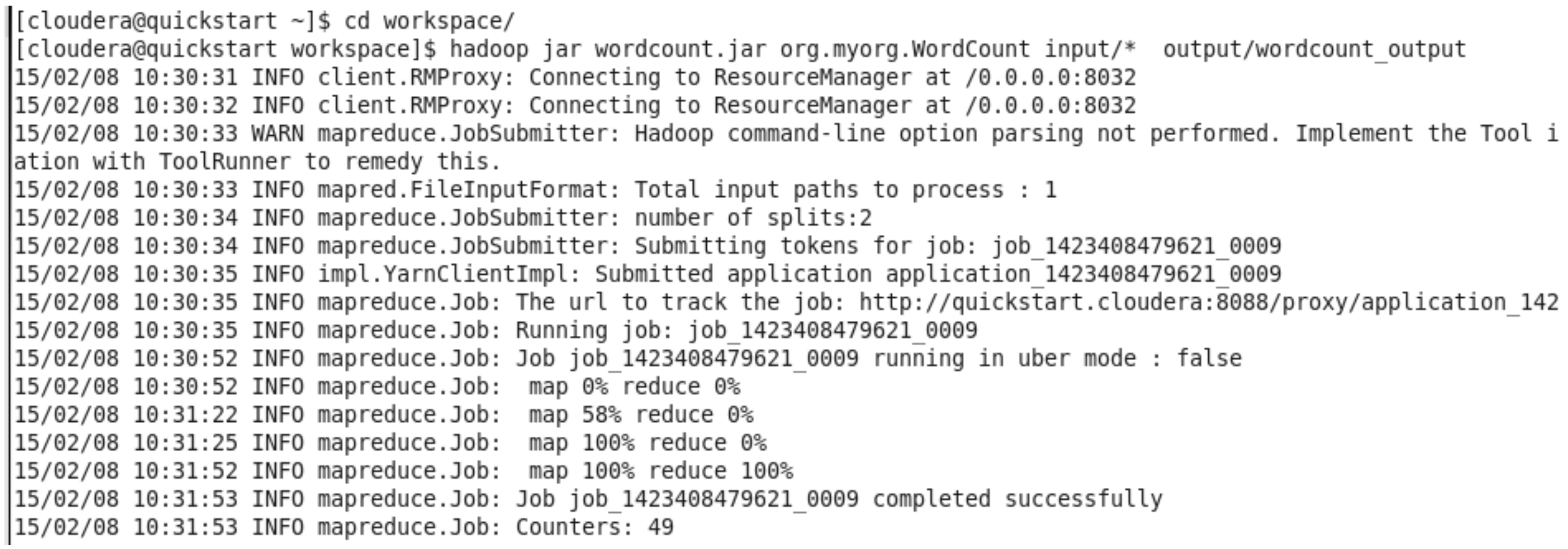


```
public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  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



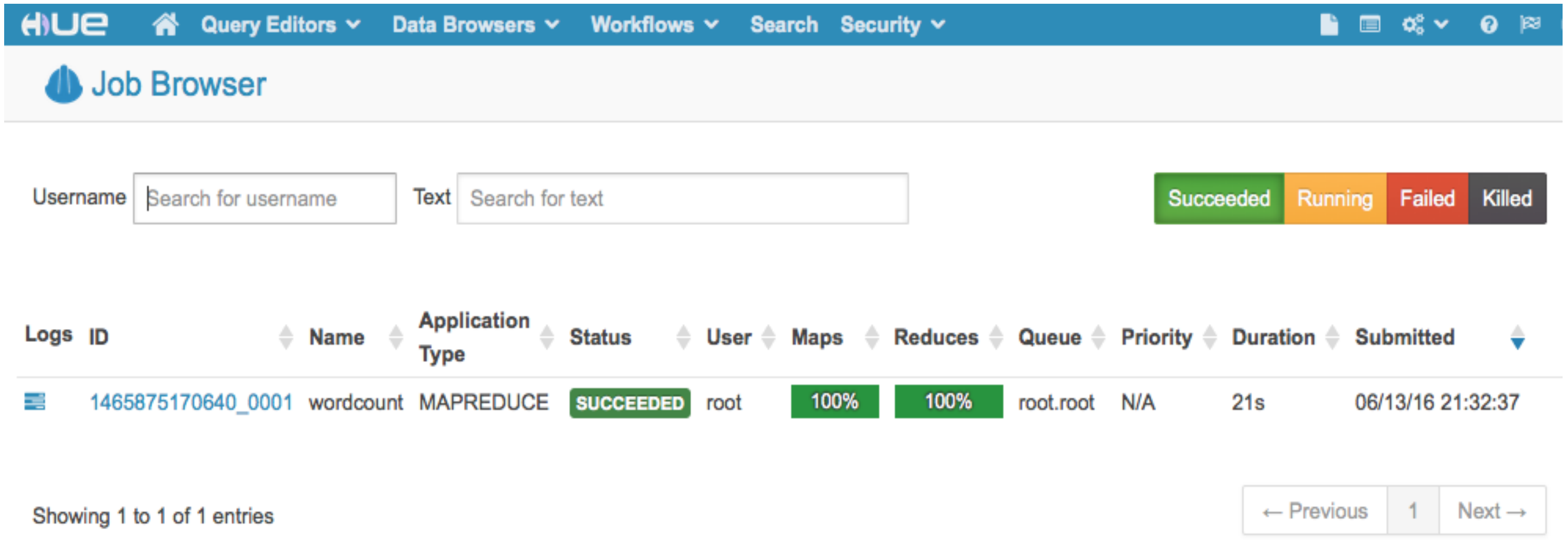
```
[cloudera@quickstart ~]$ cd workspace/
[cloudera@quickstart workspace]$ hadoop jar wordcount.jar org.myorg.WordCount input/*  output/wordcount_output
15/02/08 10:30:31 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/02/08 10:30:32 INFO client.RMProxy: Connecting to ResourceManager 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
```

https://www.dropbox.com/s/nhzlcvcfteyoqlu/wordcount.jar

:

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

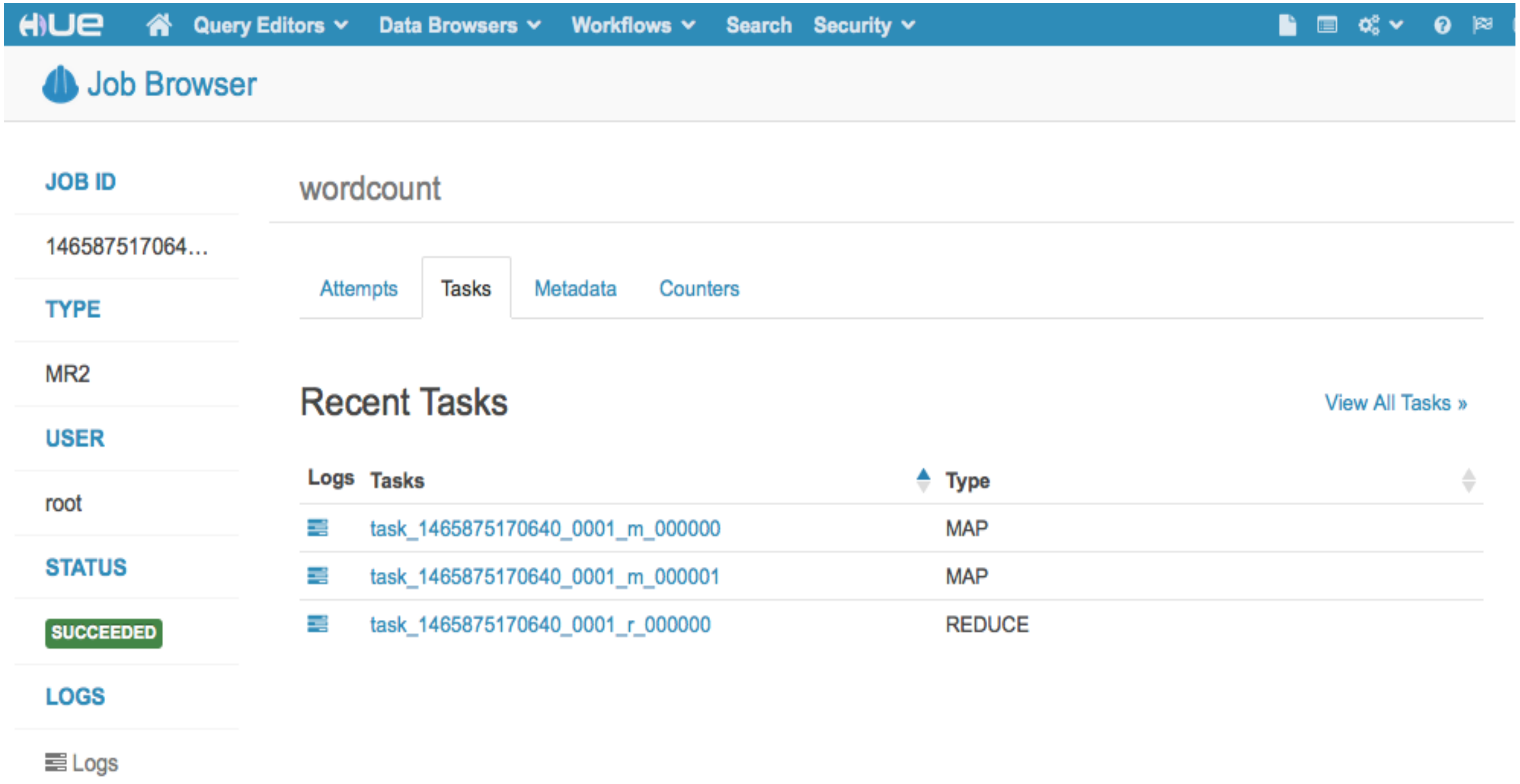
Reviewing MapReduce Job in Hue



Logs	ID	Name	Application Type	Status	User	Maps	Reduces	Queue	Priority	Duration	Submitted
	1465875170640_0001	wordcount	MAPREDUCE	SUCCEEDED	root	100%	100%	root.root	N/A	21s	06/13/16 21:32:37

Showing 1 to 1 of 1 entries

Reviewing MapReduce Job in Hue



HUE | Query Editors | Data Browsers | Workflows | Search | Security

Job Browser

JOB ID	146587517064...
TYPE	MR2
USER	root
STATUS	SUCCEEDED
LOGS	Logs

wordcount

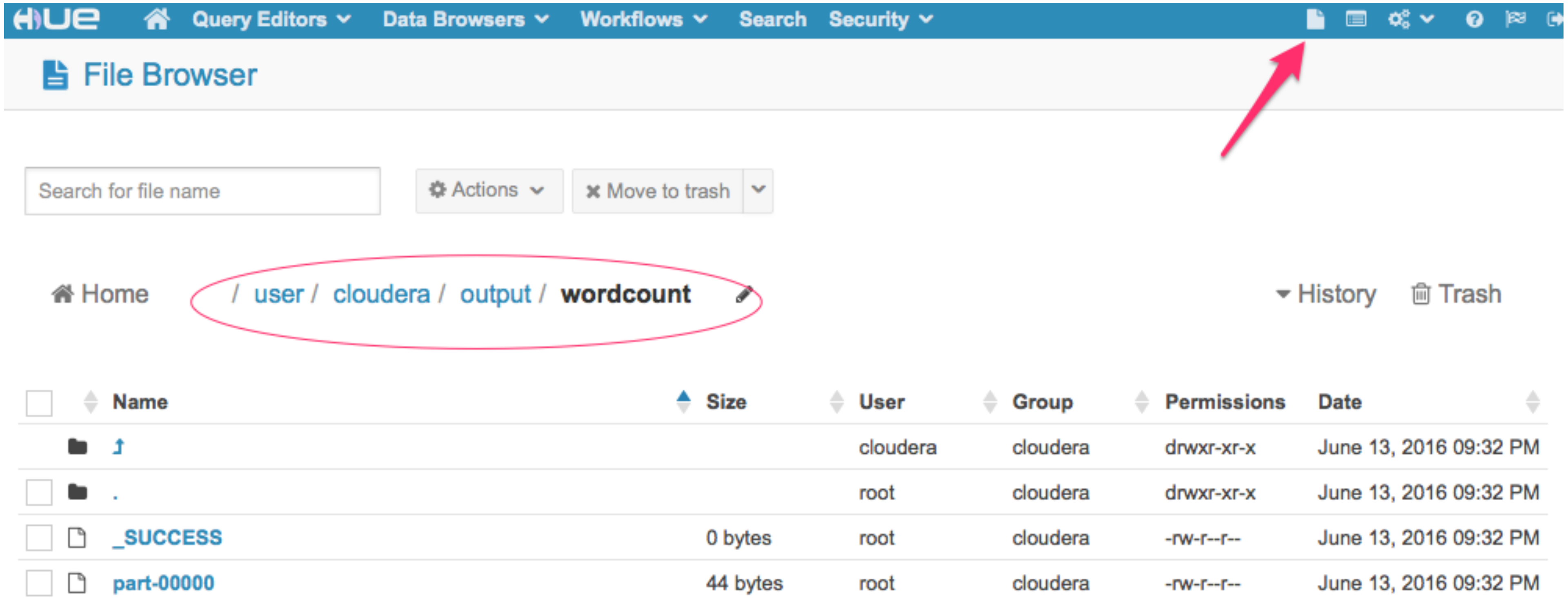
Attempts | Tasks | Metadata | Counters

Recent Tasks

View All Tasks »

Logs	Tasks	Type
☰	task_1465875170640_0001_m_000000	MAP
☰	task_1465875170640_0001_m_000001	MAP
☰	task_1465875170640_0001_r_000000	REDUCE

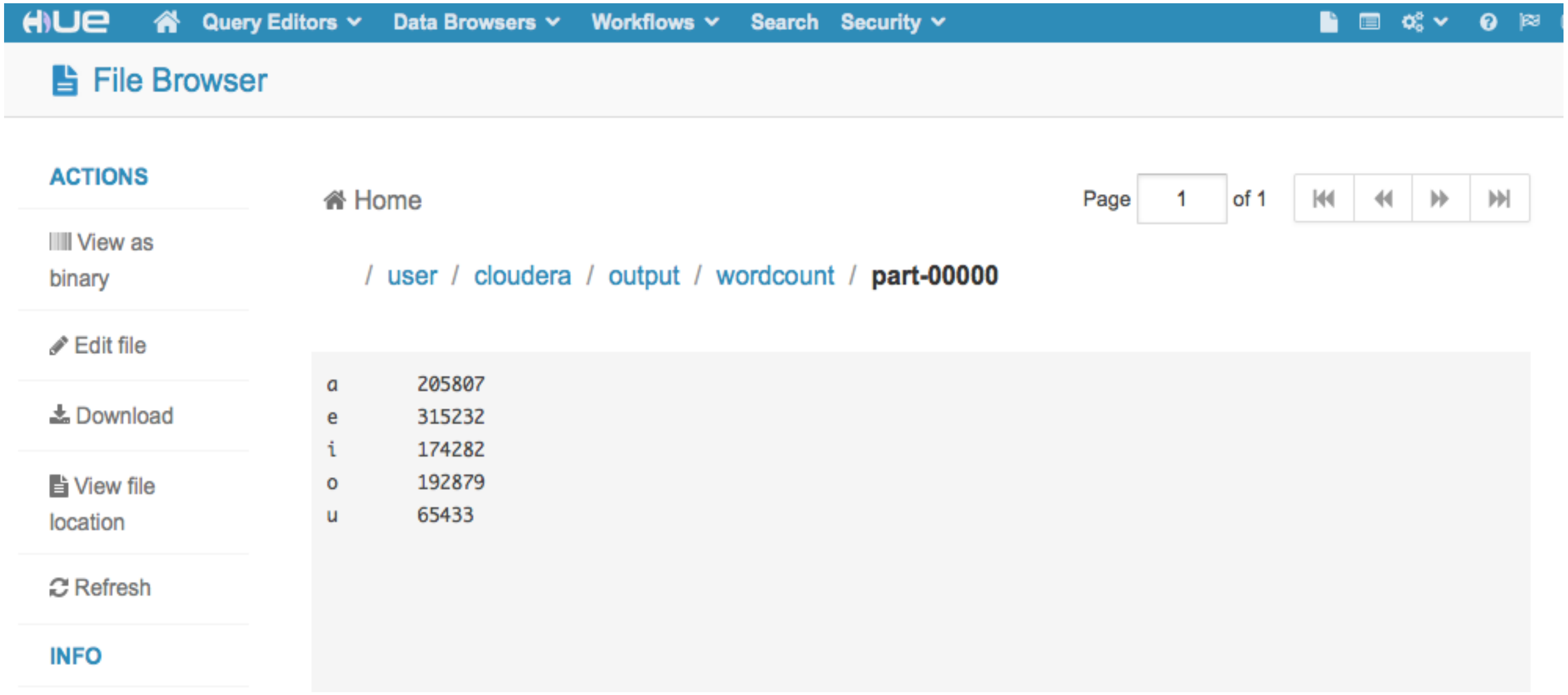
Reviewing MapReduce Output Result



/ user / cloudera / output / wordcount

	Name	Size	User	Group	Permissions	Date
	↥		cloudera	cloudera	drwxr-xr-x	June 13, 2016 09:32 PM
	.		root	cloudera	drwxr-xr-x	June 13, 2016 09:32 PM
	_SUCCESS	0 bytes	root	cloudera	-rw-r--r--	June 13, 2016 09:32 PM
	part-00000	44 bytes	root	cloudera	-rw-r--r--	June 13, 2016 09:32 PM

Reviewing MapReduce Output Result



/ user / cloudera / output / wordcount / **part-00000**

```
a       205807
e       315232
i       174282
o       192879
u       65433
```

Lecture

Understanding Oozie

Introduction

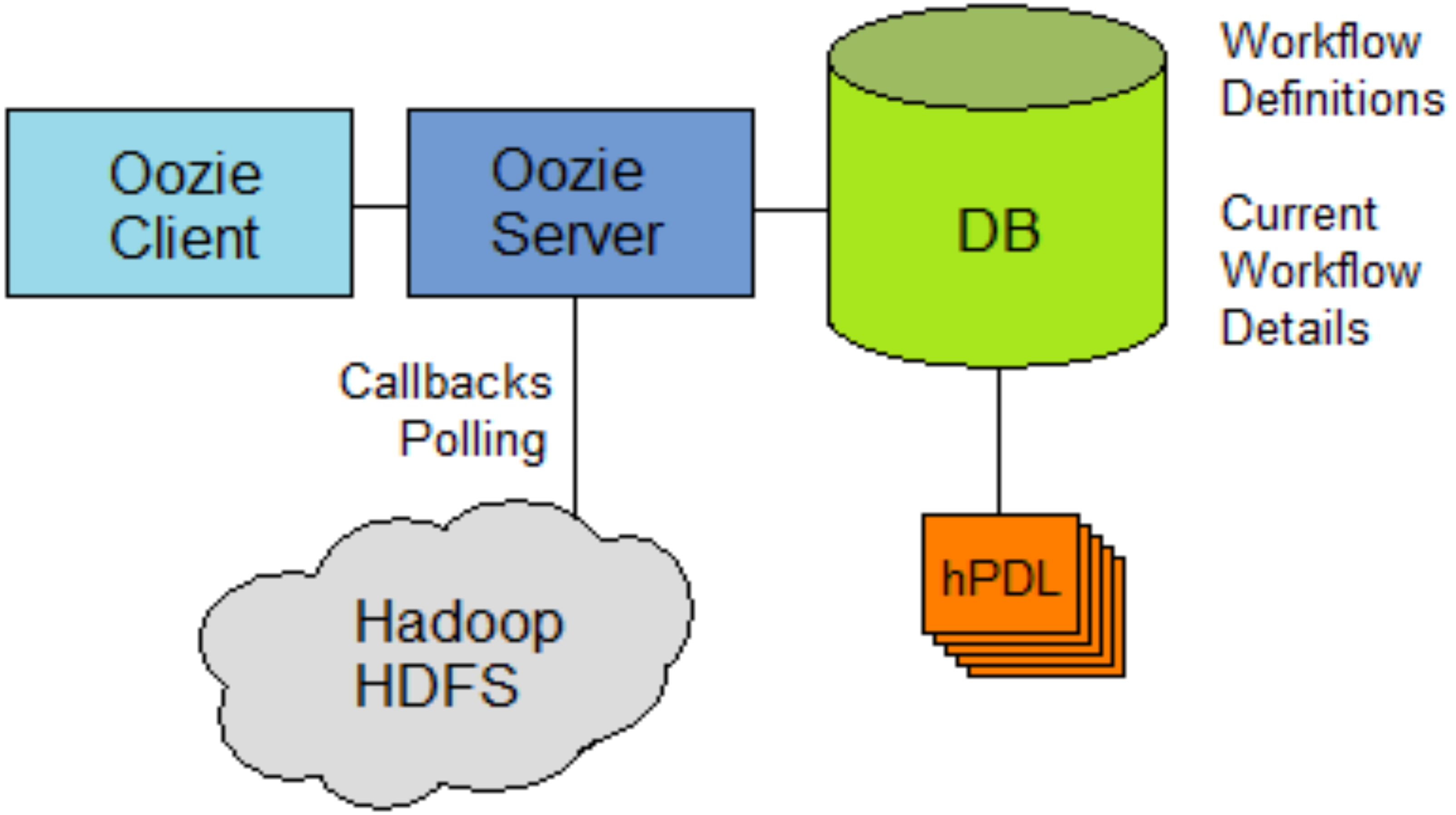
Workslow scheduler for Hadoop

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?

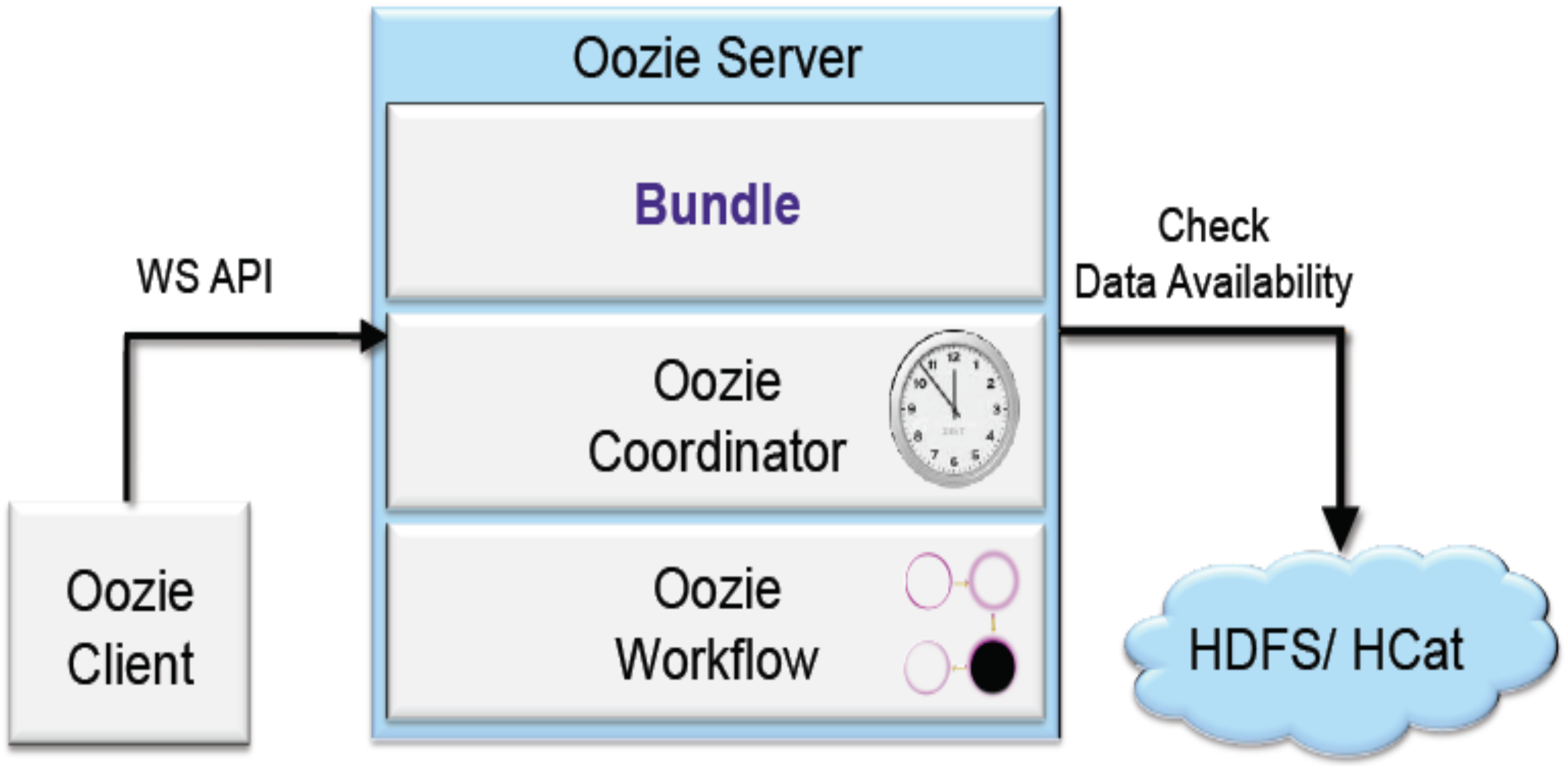
- Work flow scheduler for Hadoop
- Manages Hadoop Jobs
- Integrated with many Hadoop apps i.e. Pig, Hive
- Scaleable
- Schedule jobs
- A work flow is a collection of actions.
- A work flow is
  - Arranged as a DAG (direct acyclic graph)
  - Graph stored as hPDL (XML process definition)

Oozie Architecture



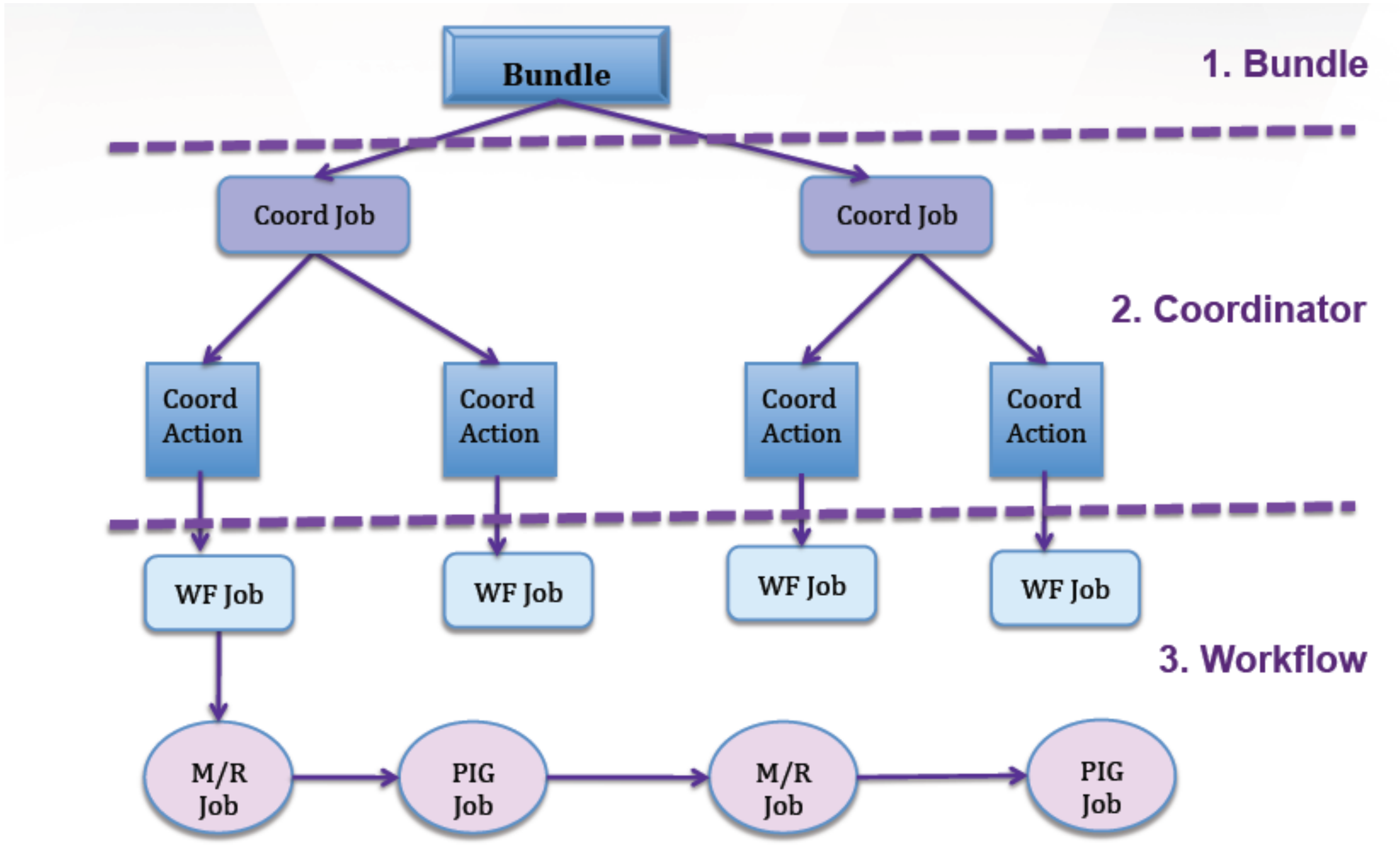
Source: info@semtech-solutions.co.nz

Oozie Server



Source: Oozie – Now and Beyond, Yahoo, 2013

Layer of Abstraction in Oozie



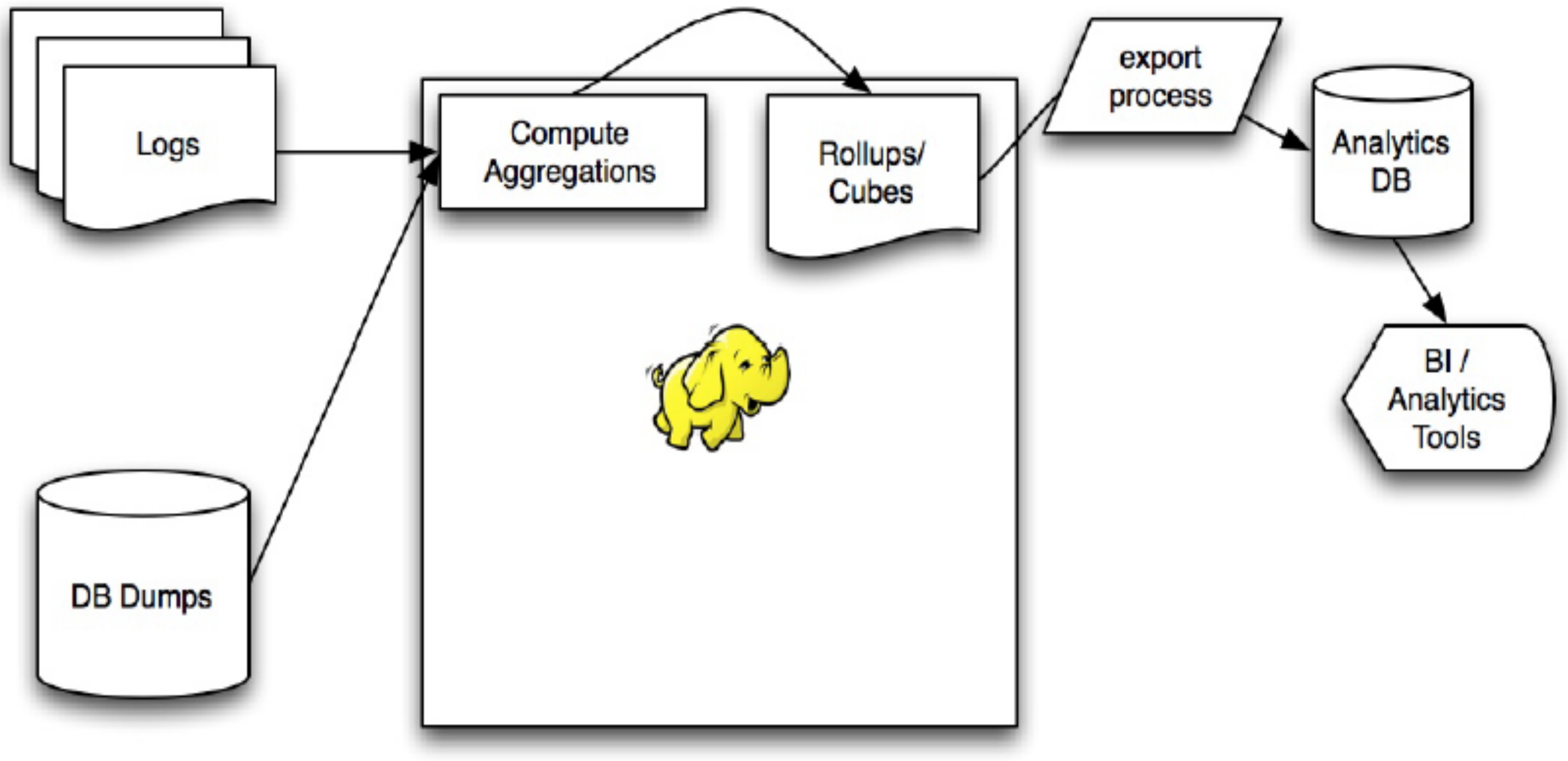
Source: Oozie – Now and Beyond, Yahoo, 2013

Workflow Example: Data Analytics

- Logs => fact table(s)
- Database backup => Dimension tables
- Complete rollups/cubes
- Load data into a low-latency storage (e.g. Hbae, HDFS)
- Dashboard & BI tools

Source: Workflow Engines for Hadoop, Joe Crobak, 2013

Workflow Example: Data Analytics



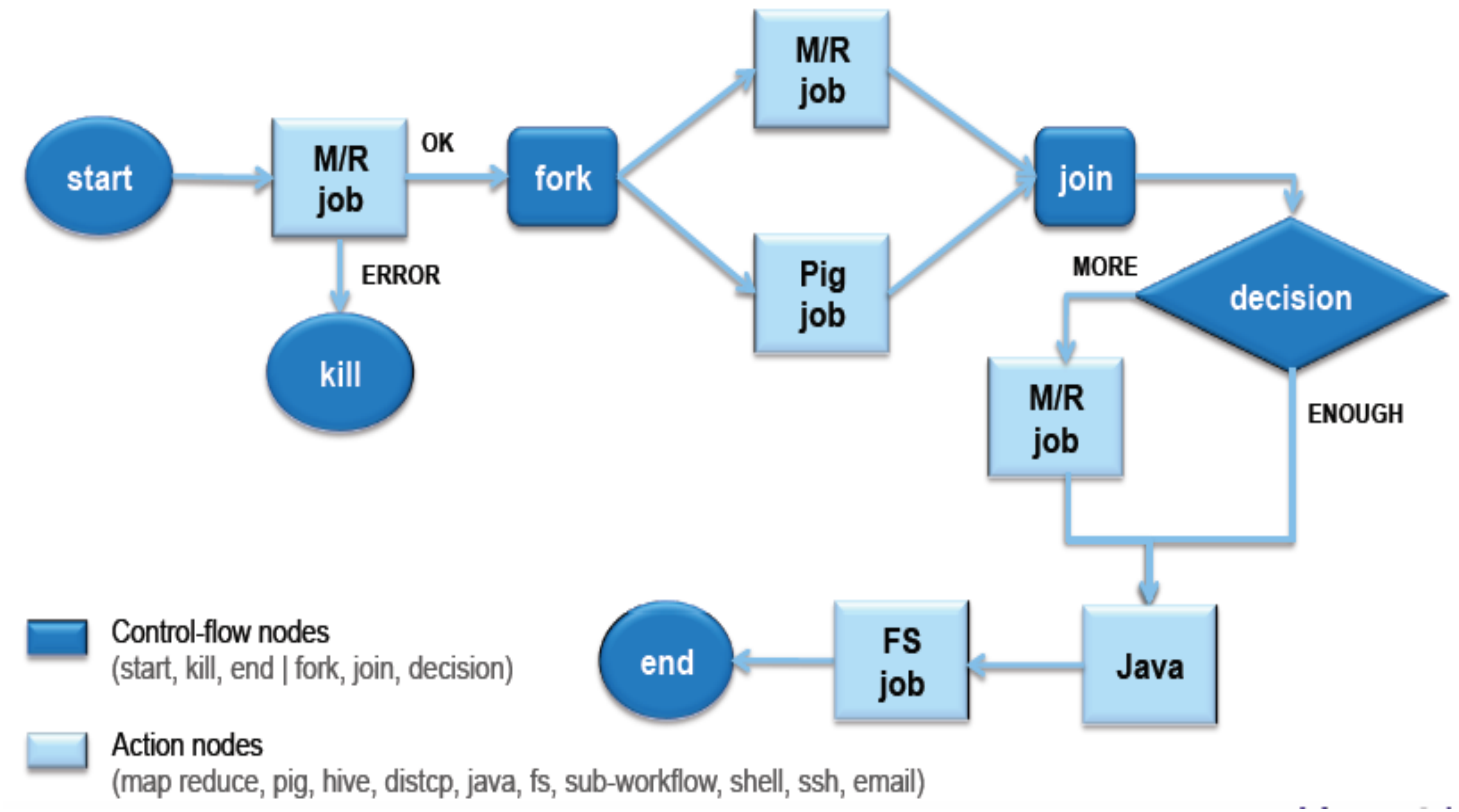
Source: Workflow Engines for Hadoop, Joe Crobak, 2013

Workflow Example: Data Analytics

- What happens if there is a failure?
  - Rebuild the failed day
  - .. and any downstream datasets
- With Hadoop Workflow
  - Possible OK to skip a day
  - Workflow tends to be self-contained, so you do not need to run downstream.
  - Sanity check your data before pushing to production.

Source: Workflow Engines for Hadoop, Joe Crobak, 2013

Oozie Workflow


start
M/R job
OK
ERROR
kill
fork
M/R job
Pig job
join
decision
MORE
ENOUGH
M/R job
Java
FS job
end
Control-flow nodes
(start, kill, end | fork, join, decision)
Action nodes
(map reduce, pig, hive, distcp, java, fs, sub-workflow, shell, ssh, email)

Source: Oozie – Now and Beyond, Yahoo, 2013

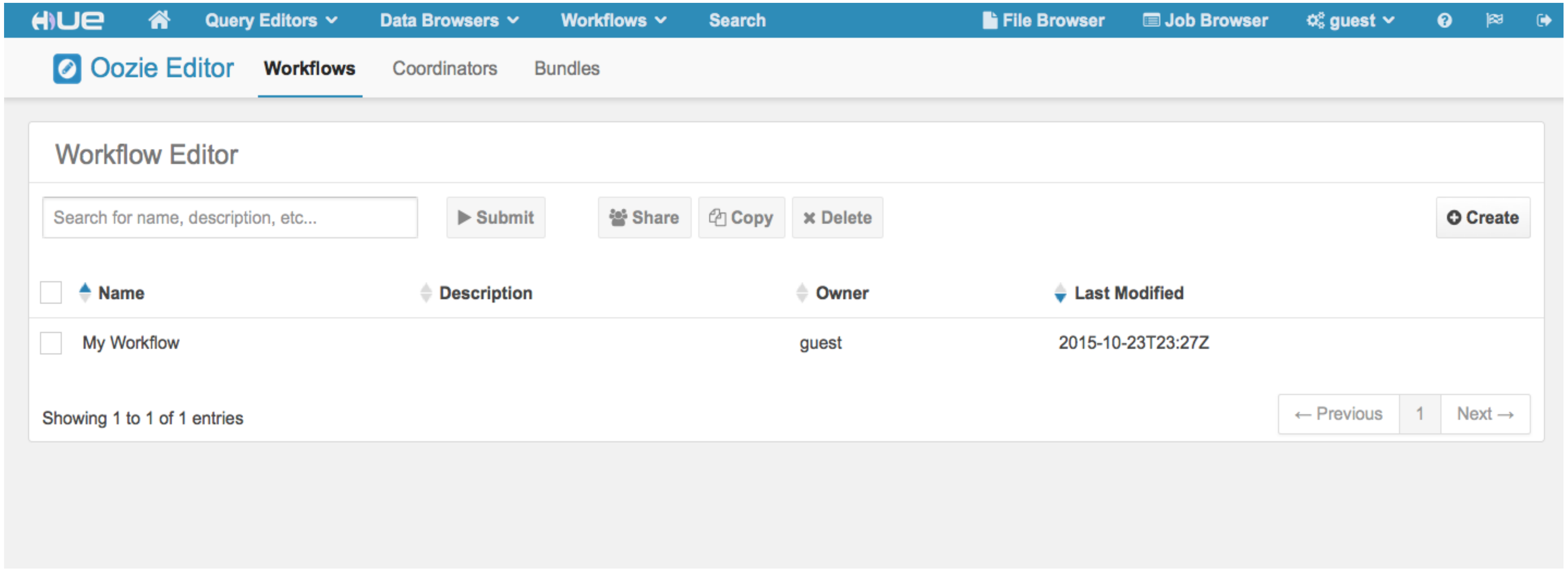
Oozie Use Cases

- Time Triggers
  - Execute your workflow every 15 minutes
- Time and Data Triggers
  - Materialize your workflow every hour, but only run them when the input data is ready (that is loaded to the grid every hour)
- Rolling Window
  - Access 15 minute datasets and roll them up into hourly datasets

Source: Oozie – Now and Beyond, Yahoo, 2013

Hands-On: Running Map Reduce using Oozie workflow

Using Hue: select WorkFlows >> Editors >> Workflows



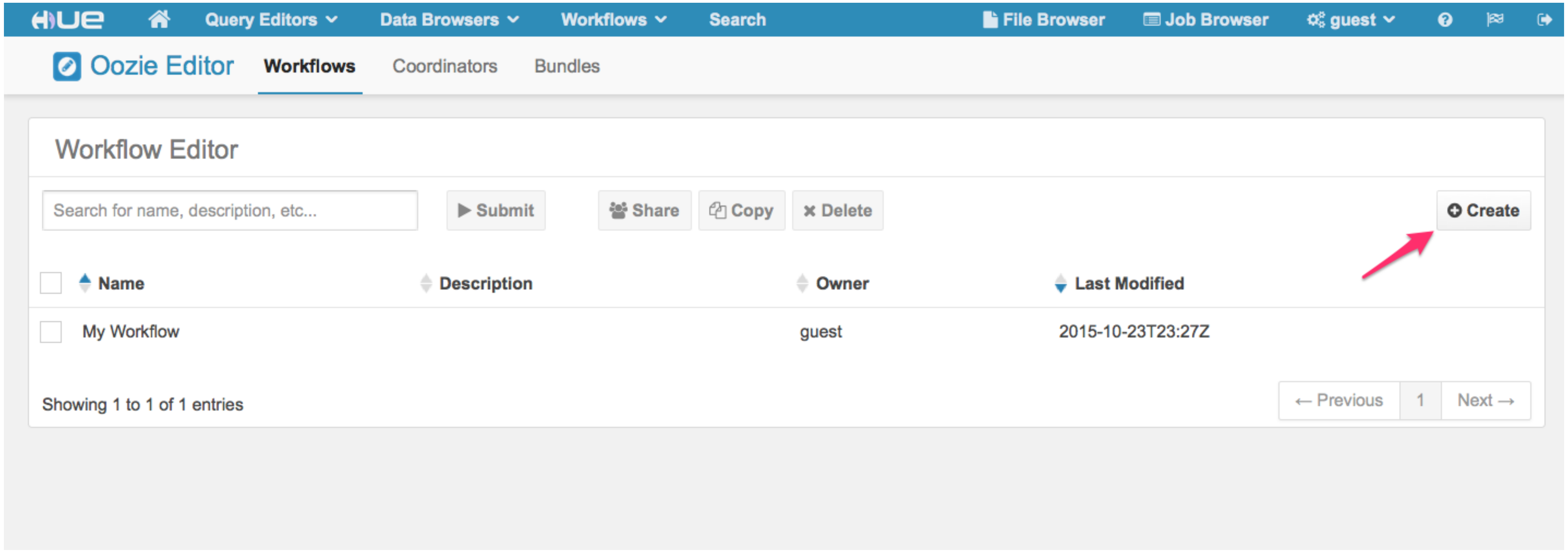
	Name	Description	Owner	Last Modified
	My Workflow		guest	2015-10-23T23:27Z

Showing 1 to 1 of 1 entries

← Previous 1 Next →

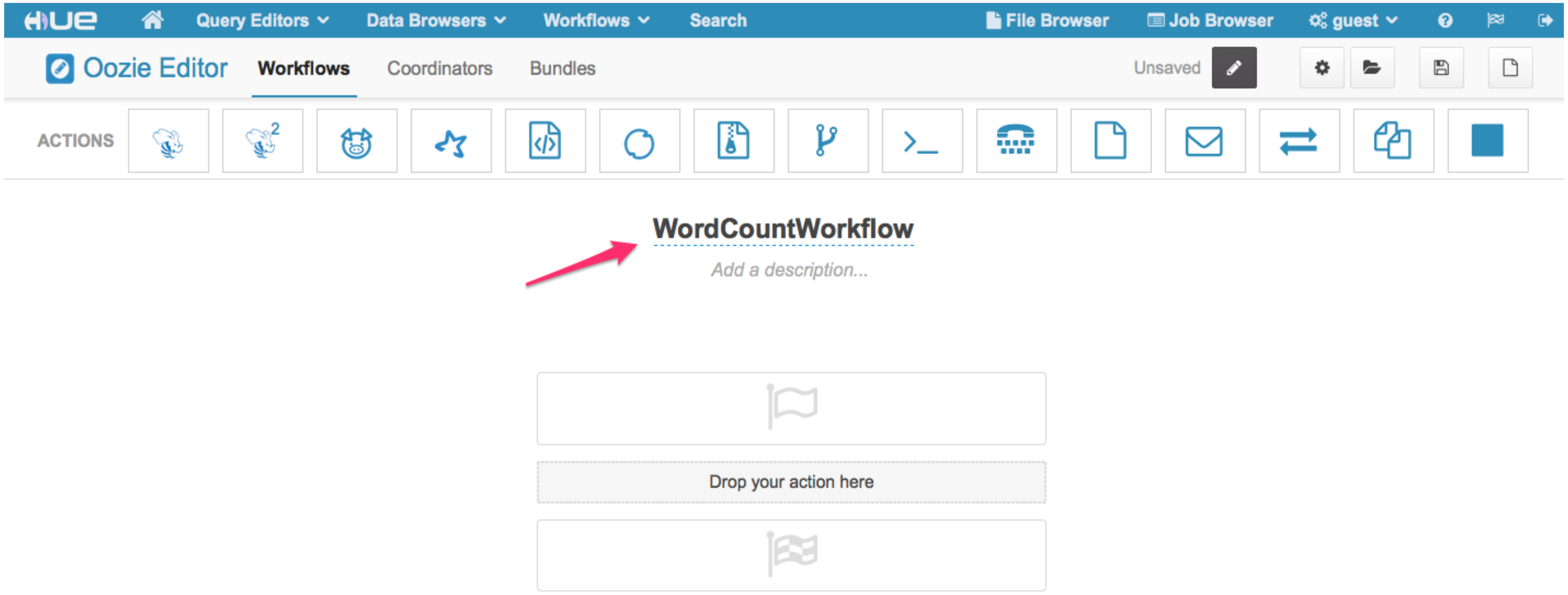
Create a new workflow

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



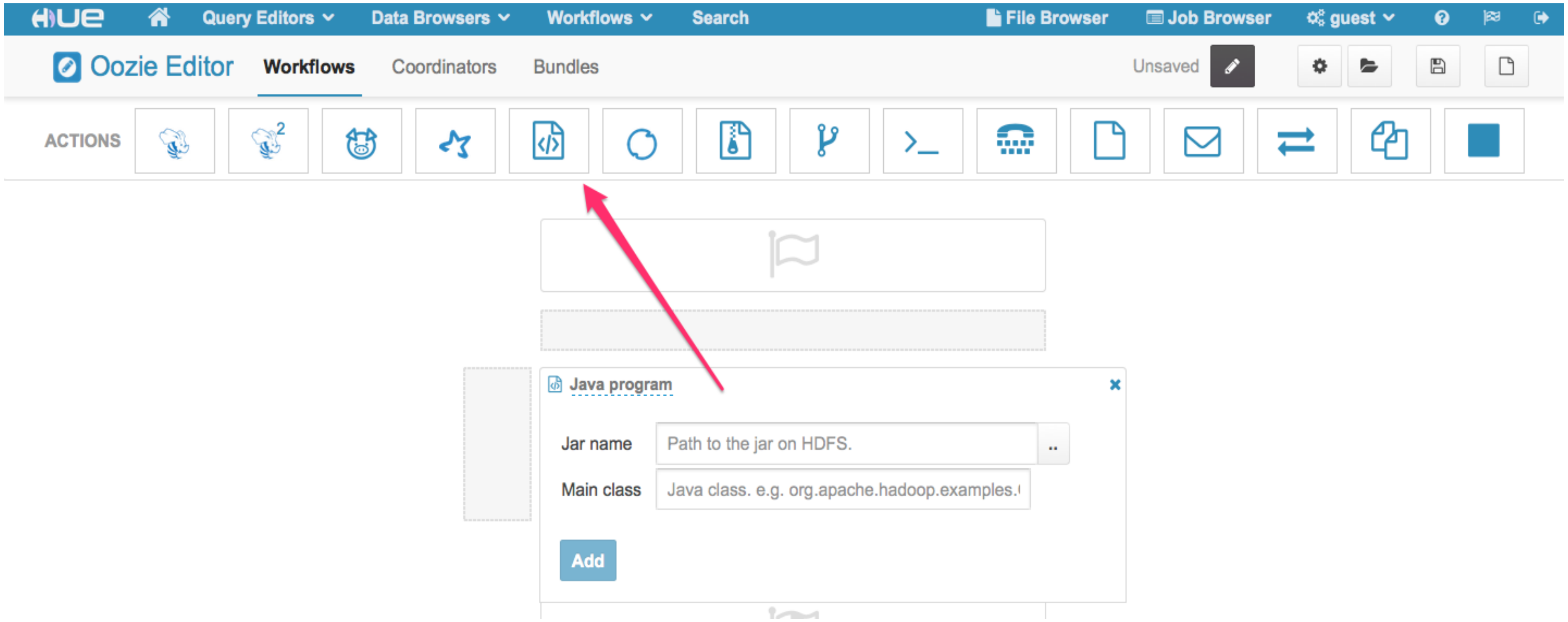
	Name	Description	Owner	Last Modified
	My Workflow		guest	2015-10-23T23:27Z

Showing 1 to 1 of 1 entries


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



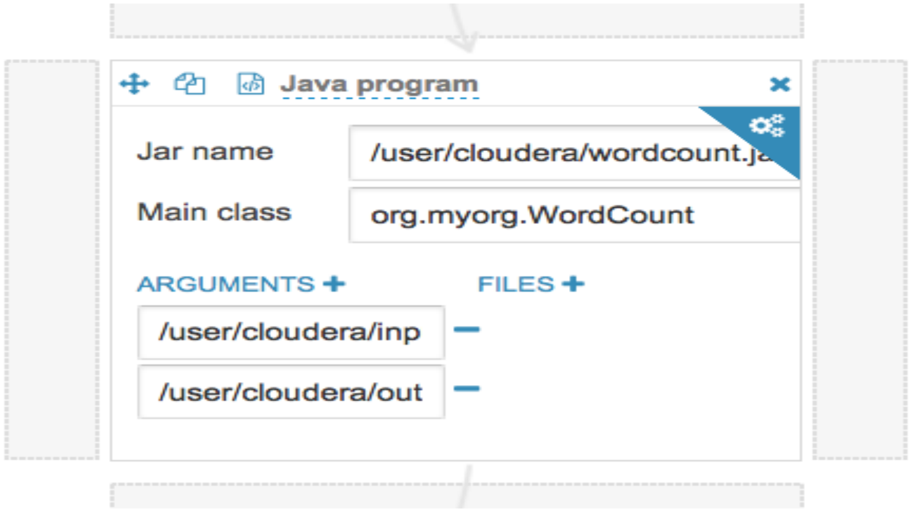
Jar name: Path to the jar on HDFS.

Main class: Java class. e.g. org.apache.hadoop.examples.

Edit the Java Job

Assign the following value

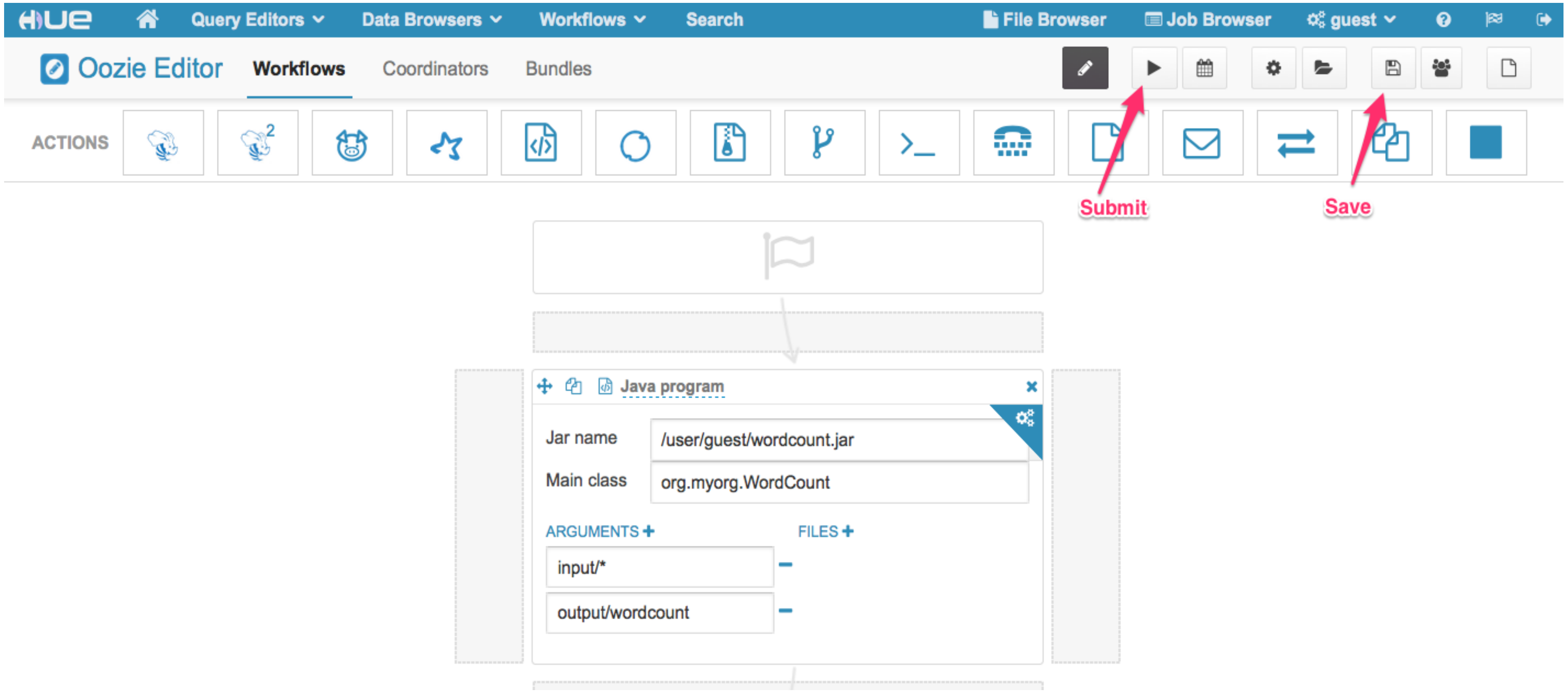
- Jar name: wordcount.jar (select … choose upload from local machine)
- Main Class: org.myorg.WordCount
- Arguments: /user/cloudera/input/*
- /user/cloudera/output/wordcount



Jar name /user/cloudera/wordcount.ja
Main class org.myorg.WordCount
/user/cloudera/inp
/user/cloudera/out

Submit the workflow

- Click Done, follow by Save
- Then click submit



Jar name	/user/guest/wordcount.jar
Main class	org.myorg.WordCount

ARGUMENTS + FILES +

input/*

output/wordcount