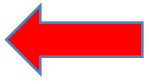
COMP4442 Service and Cloud Computing

Lab 5

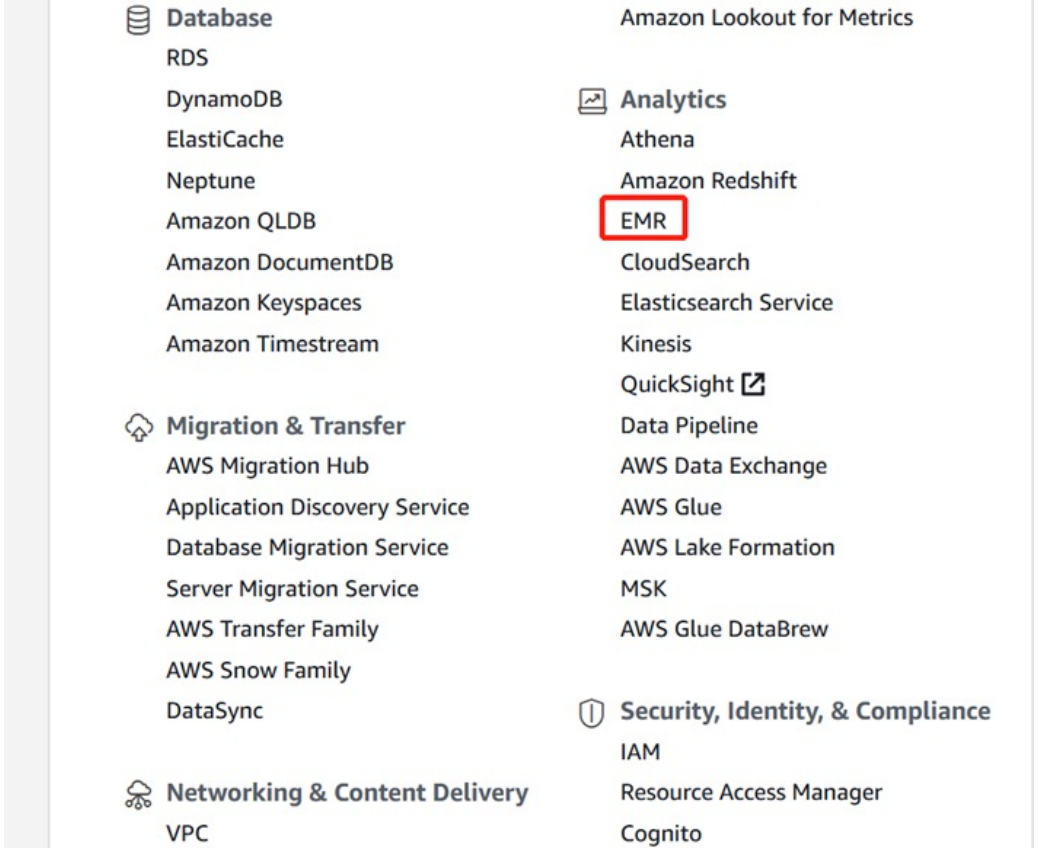
Outline

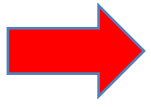
- Operations of Spark Resilient Distributed Datasets (RDD)

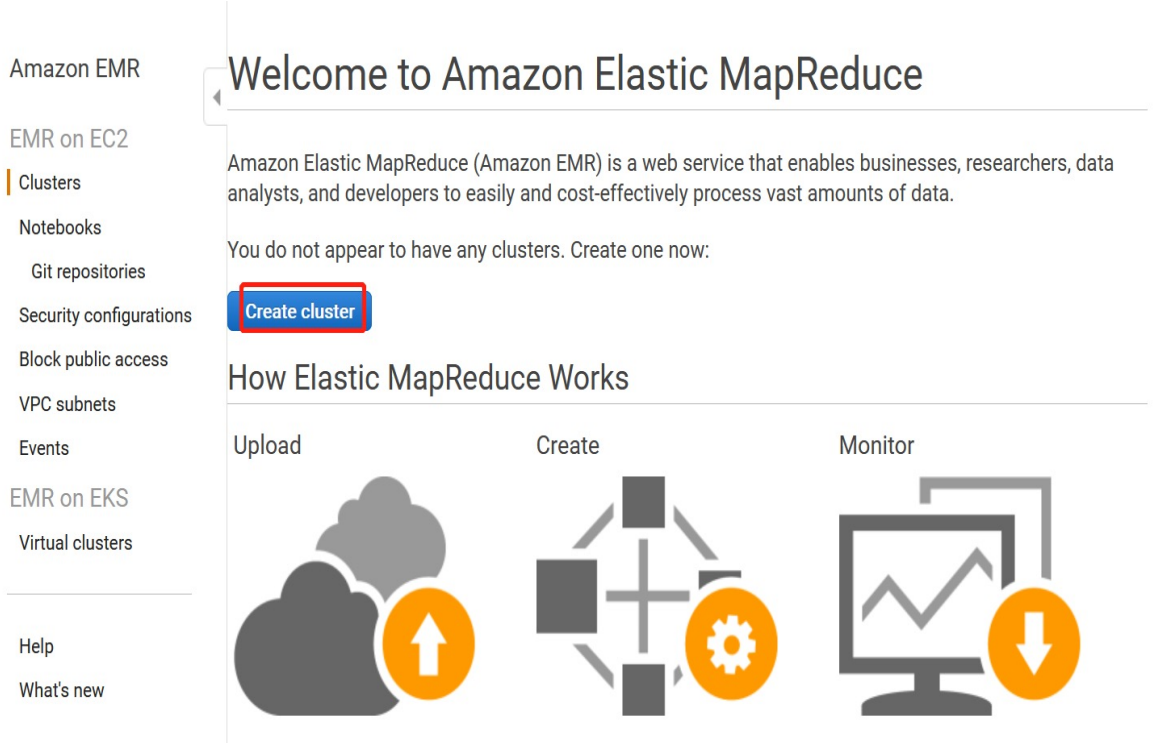
- How to use AWS Spark
  - Create Spark Cluster 
  - Develop a spark program with python
  - Submit spark job to cluster

Create Spark Cluster

- Find the EMR service on the console, then create the spark cluster.

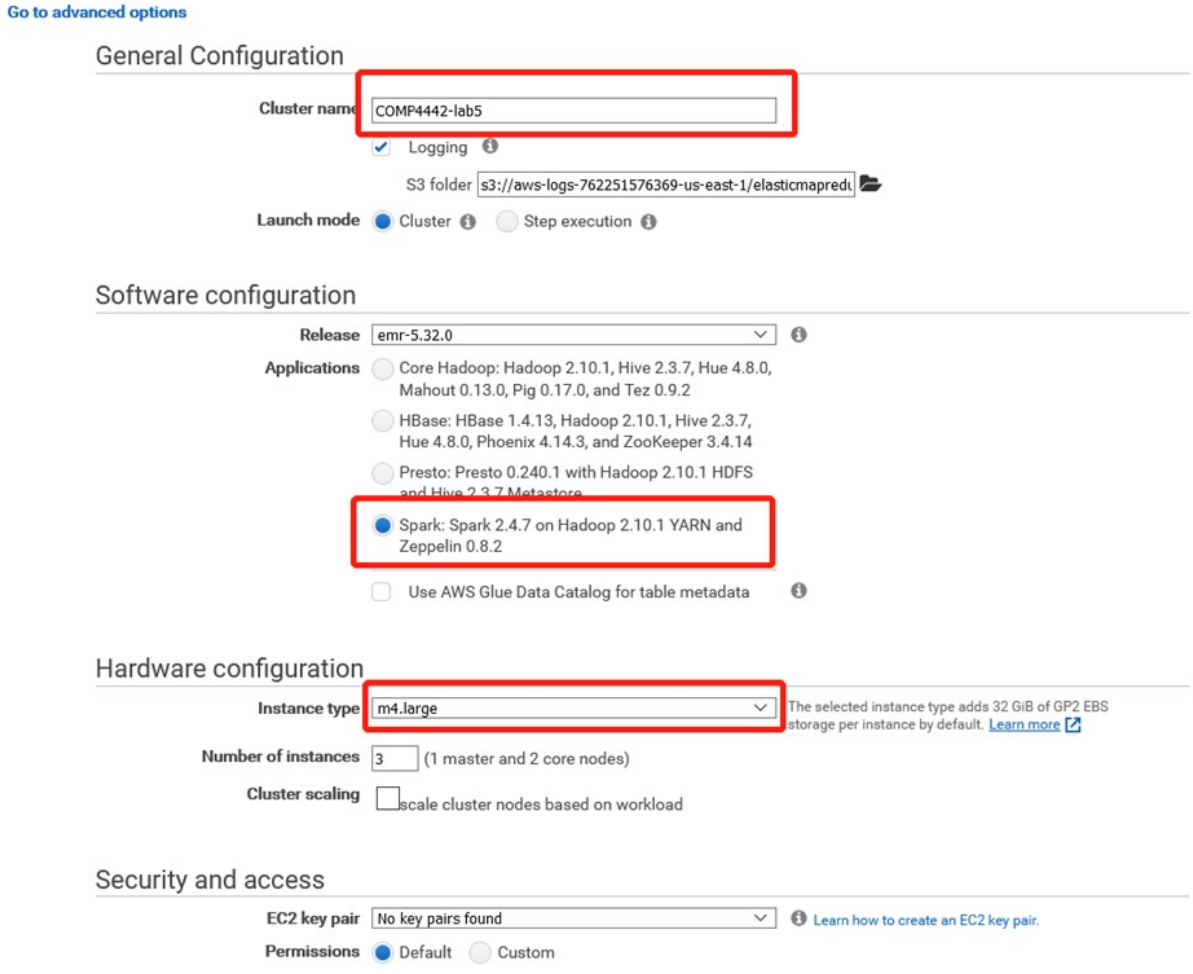






Create Spark Cluster

- Do the following configuration
  - Set cluster name
  - Choose **spark** at the cluster software
  - Choose **m4.large** instance type
  - Leave other configurations as default



Operations of Spark RDD

- RDD is considered as the core of Spark
  - immutable and distributed collection of objects.
  - fundamental data structure, containing any type of *Python, Java, or Scala* objects, including user-defined classes.

- Spark program is essentially built on operations with RDD

- Two important operations on RDDs
  - Transformations
  - Actions.

Operations of Spark RDD

- Transformations are operations on the RDDs that create a new RDD by making changes to the original RDD.
  - **map()**: returns a new RDD by applying a function to each of the elements in the original RDD.
  - **flatMap()**: returns a new RDD by applying the function to every element of the parent RDD and then flattening the result.

```
data= [2, 3, 4]
myRDD= sc.parallelize(data)

mapRDD= myRDD.map(lambda x: range(1,x))
```

→ [[1], [1,2], [1,2,3]]

```
flatMapRDD = myRDD.flatMap(lambda x: range(1,x))
```

→ [1, 1, 2, 1, 2, 3]

Operations of Spark RDD

- Transformations are operations on the RDDs that create a new RDD by making changes to the original RDD.
  - **reduceByKey()**: called on a dataset of (key, value) pairs, returns a new dataset in which the values for each of its key are aggregated.
  - **collect()**: returns a list that contains all the elements of the RDD.

```
from operator import add

myRDD = sc.parallelize([("a", 1), ("a", 2), ("a", 3), ("b", 1)])

#adds the values by keys
newRDD= myRDD.reduceByKey(add)
newRDD.collect()
```

[('a', 6), ('b', 1)]

Operations of Spark RDD

- Actions are the operations on the RDD to carry out computations and return the final result back to the driver.
  - **reduce()**: aggregates the elements of the RDD using a function that takes two elements of the RDD as input and gives the result.

```
data= [“Scala”, “Python”, “Java”, “R”]
myRDD= sc.parallelize(data)

#Concatenate the string elements
myRDD.reduce( lambda x, y: x + y)
```

‘ScalaPythonJavaR’

Operations of Spark RDD

- Actions are the operations on the RDD to carry out computations and return the final result back to the driver.
  - **countByValue()**: returns the count of each unique value in the RDD as dictionary with (value, count) pairs.

```
data= ["Python", "Scala", "Python", "R", "Python", "Java", "R", ]

myRDD= sc.parallelize(data)

#items() returns a list with all the dictionary keys and values
returned by countByValue()

myRDD.countByValue().items()
```

[('Python', 3), ('R', 2), ('Java', 1), ('Scala', 1)]

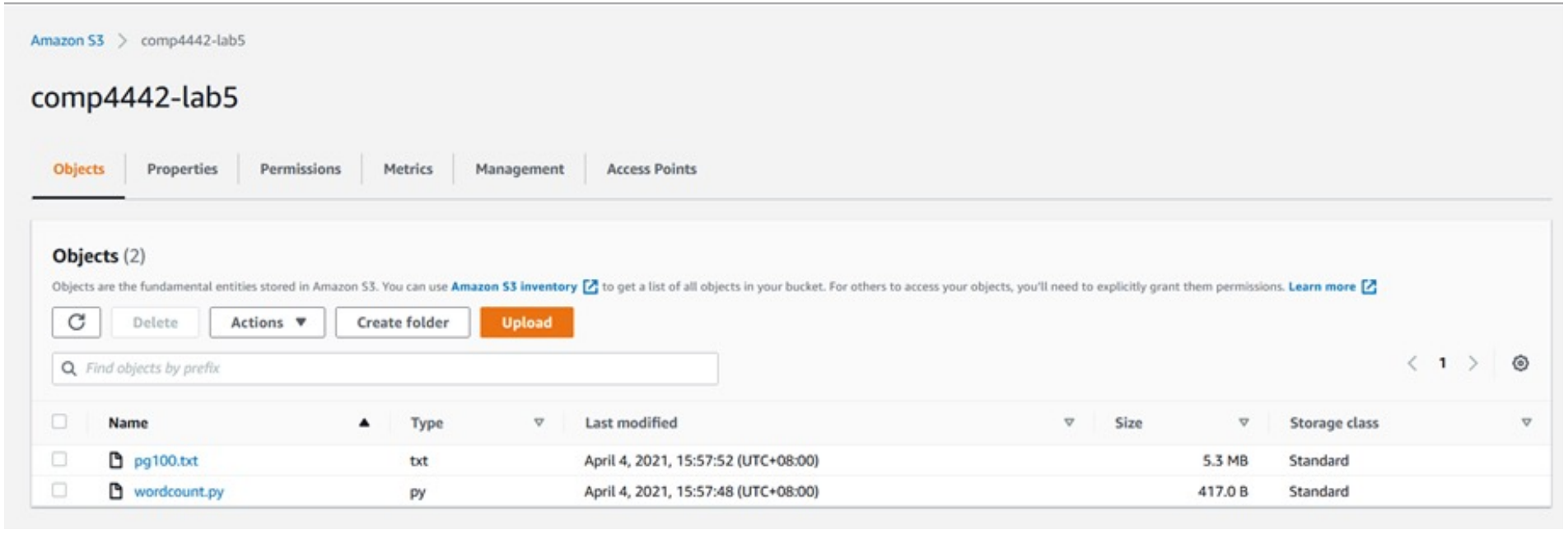
Operations of Spark RDD

Transformations	*map*(*f* : T ⇒ U)	:	RDD[T] ⇒ RDD[U]
	filter(*f* : T ⇒ Bool)	:	RDD[T] ⇒ RDD[T]
	flatMap(*f* : T ⇒ Seq[U])	:	RDD[T] ⇒ RDD[U]
	sample(*fraction* : Float)	:	RDD[T] ⇒ RDD[T] (Deterministic sampling)
	groupByKey()	:	RDD[(K, V)] ⇒ RDD[(K, Seq[V])]
	reduceByKey(*f* : (V, V) ⇒ V)	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	union()	:	(RDD[T], RDD[T]) ⇒ RDD[T]
	join()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (V, W))]
	cogroup()	:	(RDD[(K, V)], RDD[(K, W)]) ⇒ RDD[(K, (Seq[V], Seq[W]))]
	crossProduct()	:	(RDD[T], RDD[U]) ⇒ RDD[(T, U)]
	mapValues(*f* : V ⇒ W)	:	RDD[(K, V)] ⇒ RDD[(K, W)] (Preserves partitioning)
	sort(*c* : Comparator[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
	partitionBy(*p* : Partitioner[K])	:	RDD[(K, V)] ⇒ RDD[(K, V)]
Actions	*count*()	:	RDD[T] ⇒ Long
	collect()	:	RDD[T] ⇒ Seq[T]
	reduce(*f* : (T, T) ⇒ T)	:	RDD[T] ⇒ T
	lookup(*k* : K)	:	RDD[(K, V)] ⇒ Seq[V] (On hash/range partitioned RDDs)
	save(*path* : String)	:	Outputs RDD to a storage system, *e.g.*, HDFS

https://medium.com/analytics-vidhya/understanding-spark-rdds-part-3-3b1b9331652a

Develop Spark Program with Python

- The *wordcount* example aims at counting the frequency of words in the input file
- Upload the input file to AWS S3



Develop Spark Program with Python

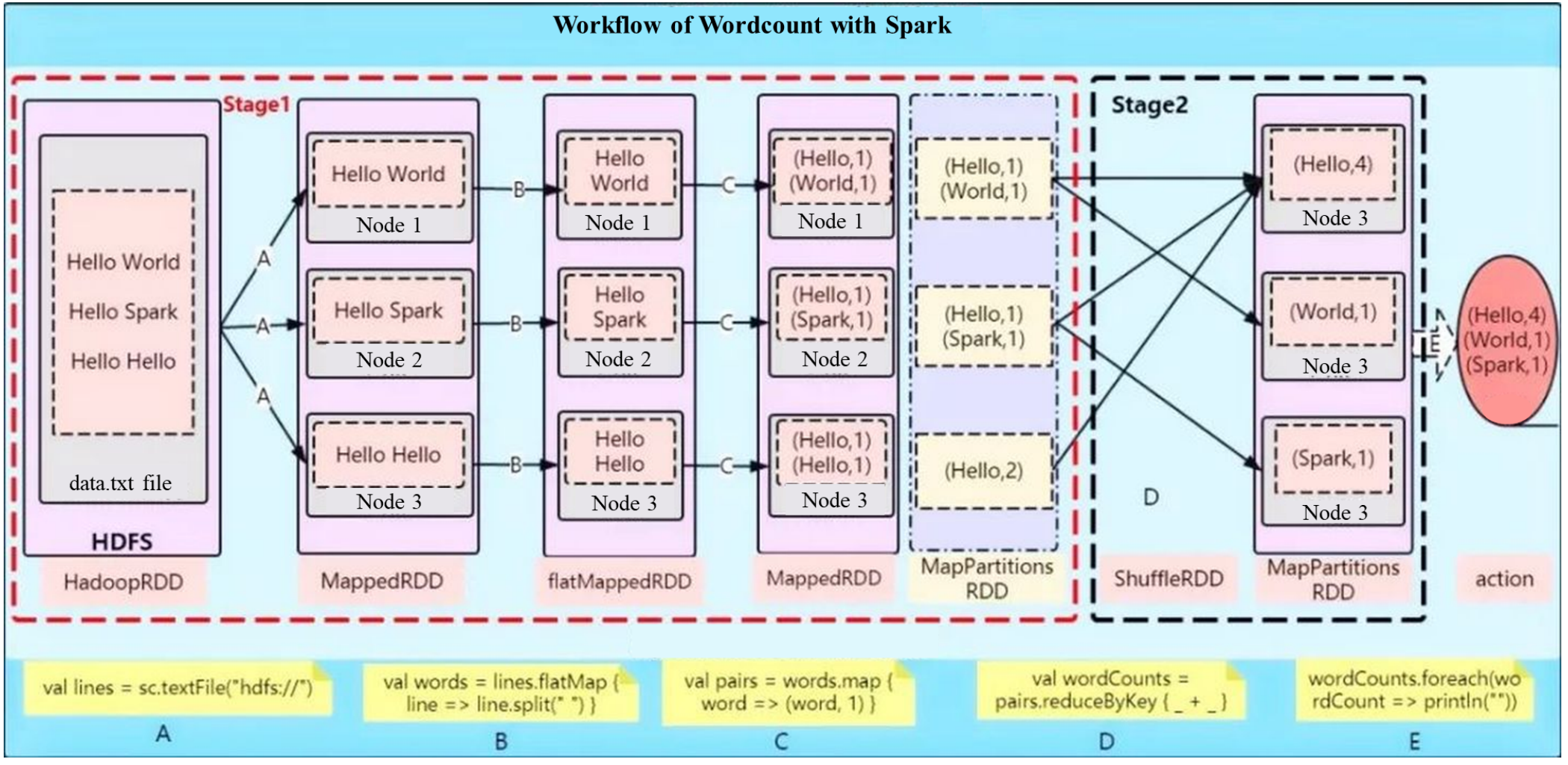
- Establish a *wordcount.py* file and upload to AWS S3

```
import os
import sys

from pyspark import SparkContext

args = sys.argv
inp = args[1]
out = args[2]

sc = SparkContext()

text_file = sc.textFile(inp)
counts = text_file.flatMap(lambda line: line.split(" ")) \
              .map(lambda word: (word, 1)) \
              .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile(out)

sc.stop()
```

Read system input parameters

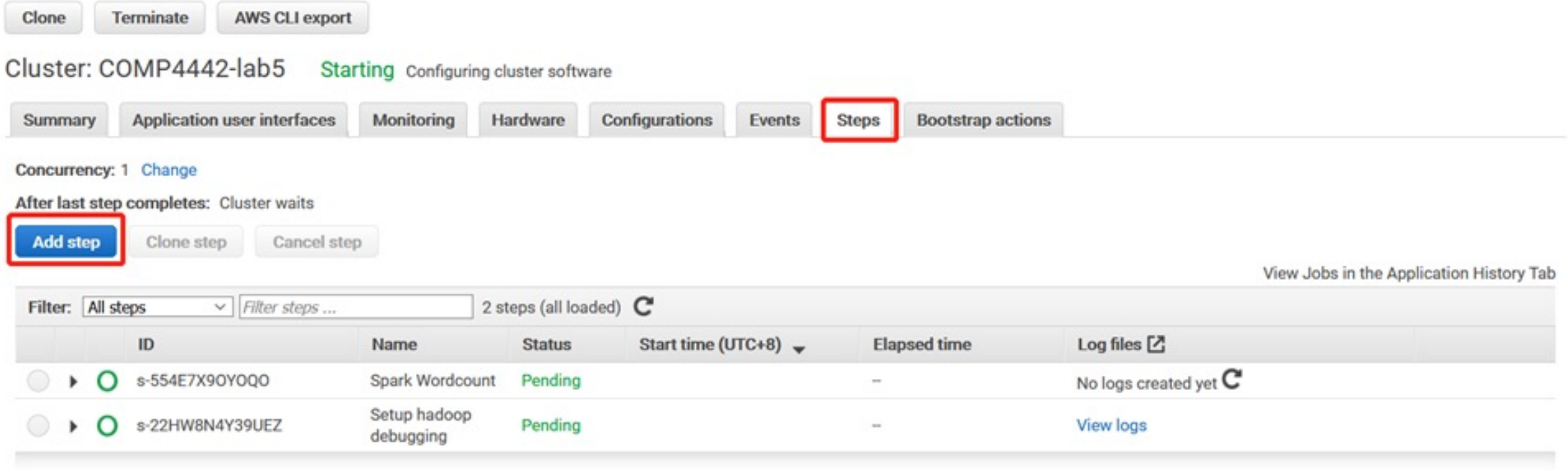
Count frequency

Workflow of Wordcount with Spark



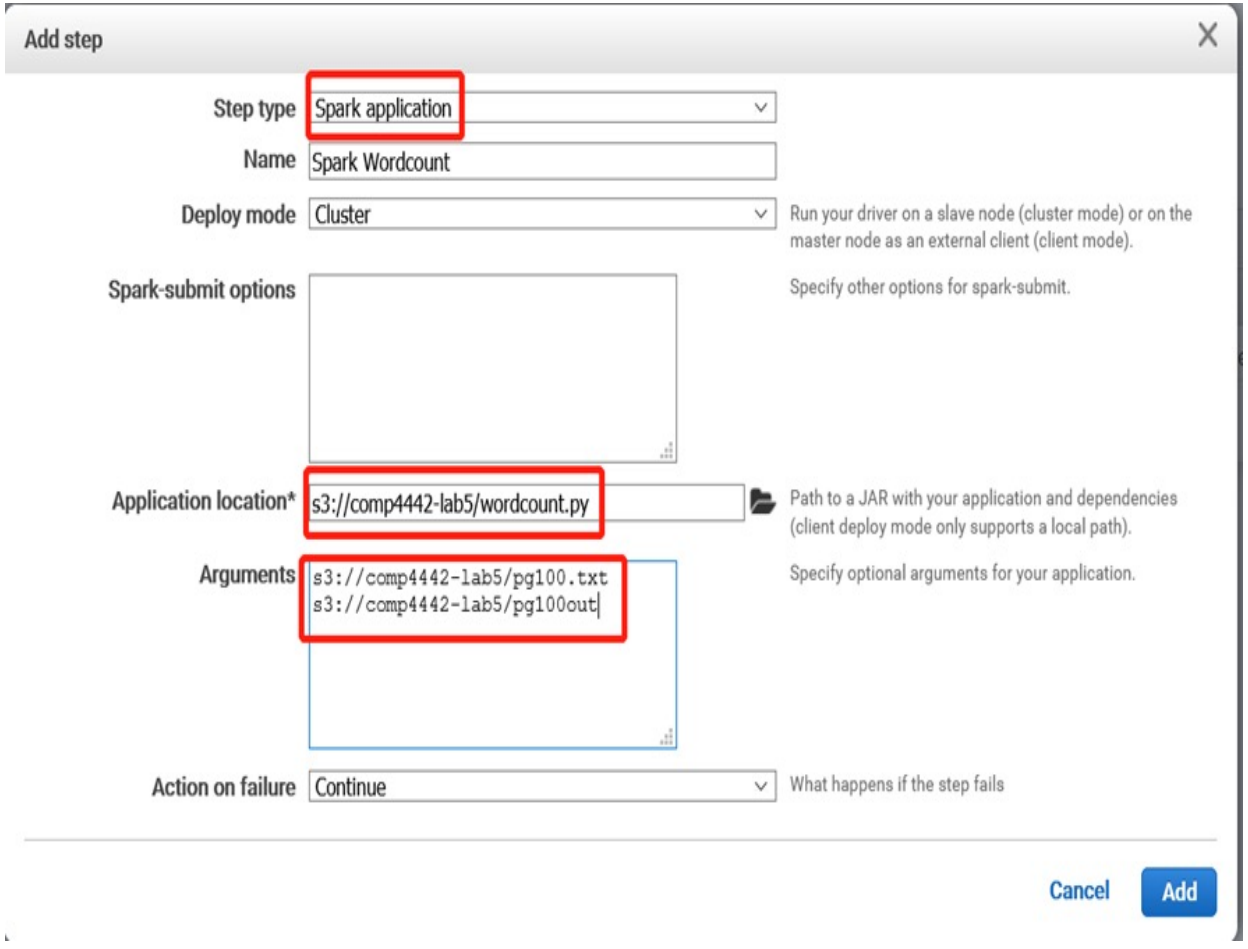
Submit Spark Job to Cluster

- On the spark cluster console, choose steps and then add step



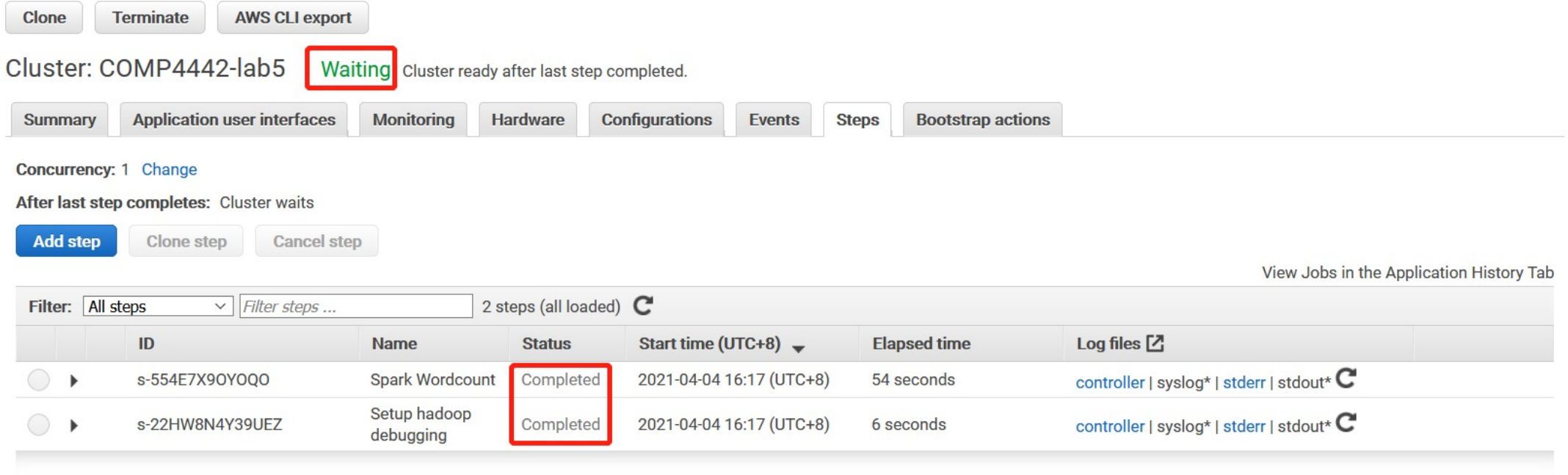
Submit Spark Job to Cluster

- Configure the spark job
  - Select Spark application as Step type
  - Indicate the application location
  - Set the application argument



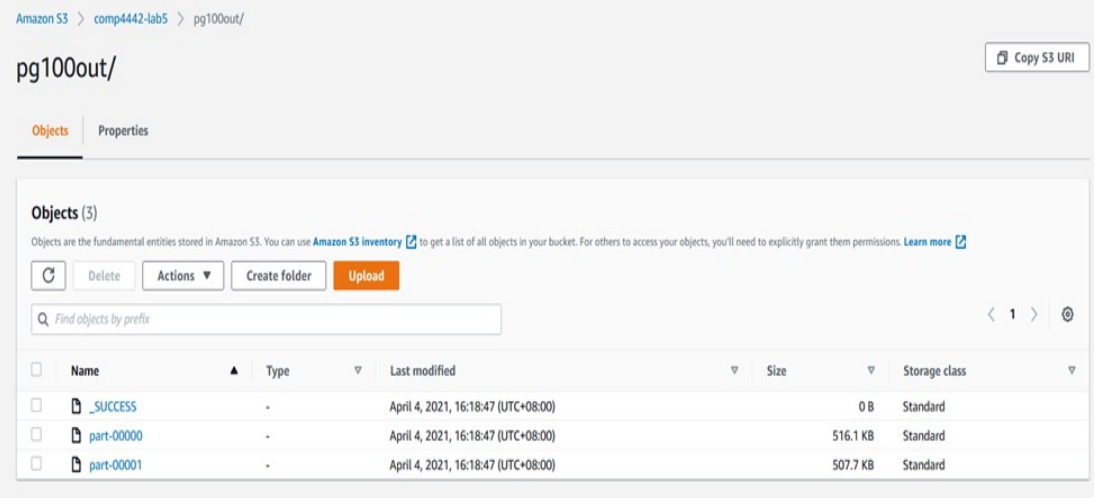
Submit Spark Job to Cluster

- Wait the cluster to run the job and save the output
  - Status of the cluster changes from starting to running, and then to wait (about 20 minutes)



Submit Spark Job to Cluster

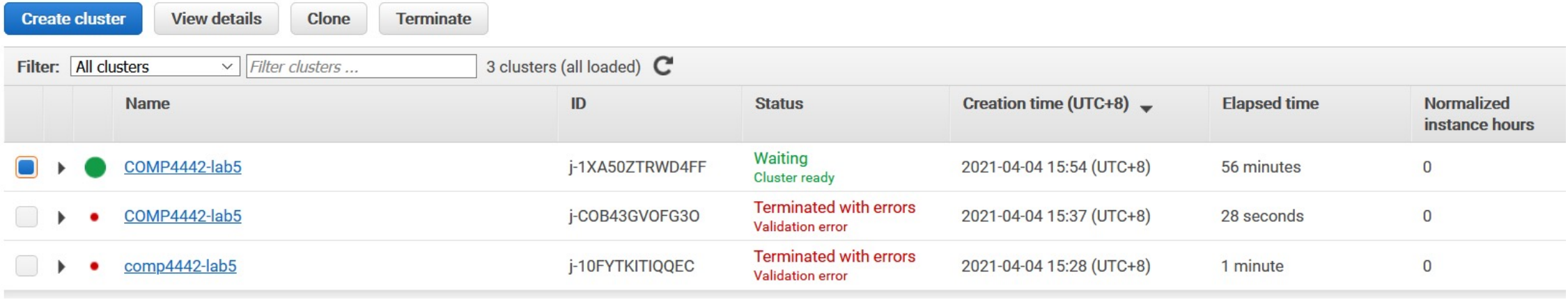
- Check the output
  - Download output file from S3




```
('Gutenberg', 7)
('Complete', 3)
('Works', 3)
('William', 64)
('Shakespeare,', 1)
('by', 2791)
('This', 1101)
('eBook', 2)
('for', 5587)
('the', 23104)
('cost', 32)
('and', 18173)
('with', 6701)
('almost', 135)
('You', 1508)
('copy', 14)
('it', 4875)
('or', 1717)
('under', 196)
('terms', 45)
('License', 1)
('included', 1)
('**', 4)
('a', 12472)
```

Stop Spark Cluster

- Do not forget to terminate the cluster
  - The AWS will save the meta-data of the cluster for about two month
  - You can easily recover the cluster by clone it



Q&A

Exercise

- What is Narrow Dependencies and Wide Dependencies of RDDs? Which RDDs operations belongs to Narrow Dependencies and Wide Dependencies, respectively?

- What is lazy evaluation of RDDs? What is the benefits of lazy evaluation?