

# **COMP4442 Service and Cloud Computing**

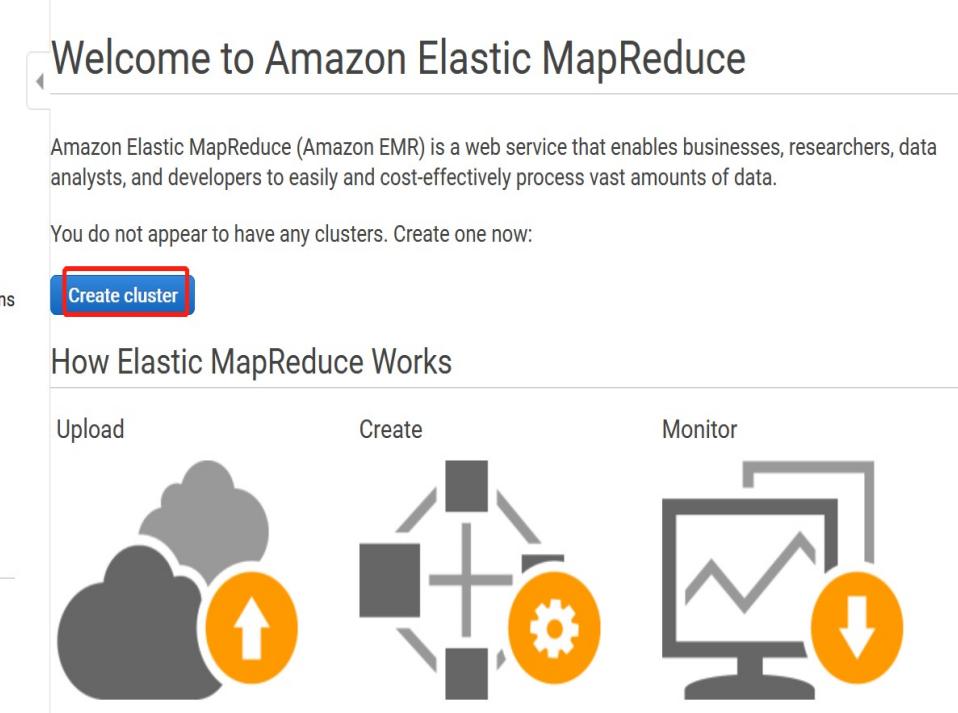
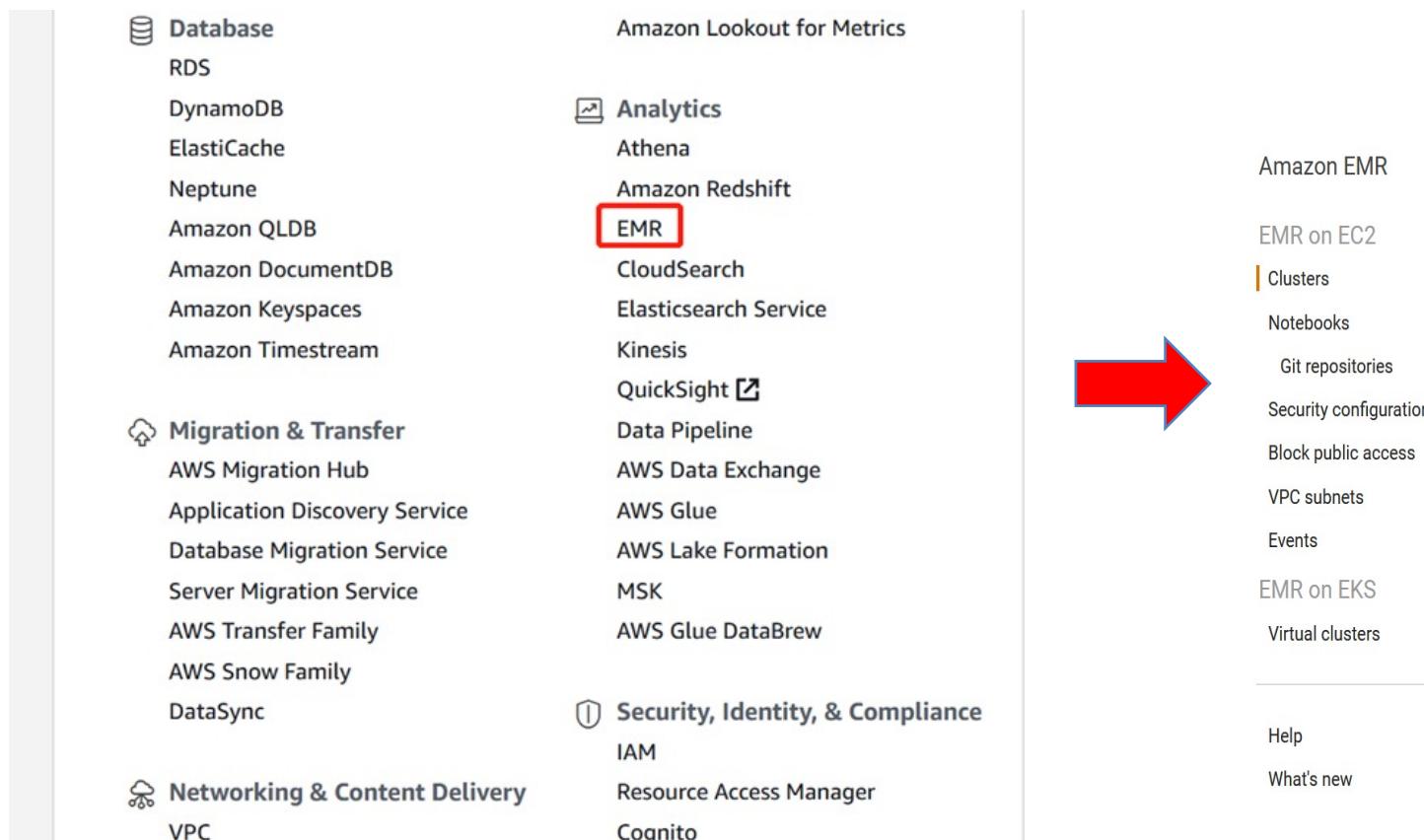
## **Lab 5**

# 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.



Welcome to Amazon Elastic MapReduce

Amazon Elastic MapReduce (Amazon EMR) is a web service that enables businesses, researchers, data analysts, and developers to easily and cost-effectively process vast amounts of data.

You do not appear to have any clusters. Create one now:

**Create cluster**

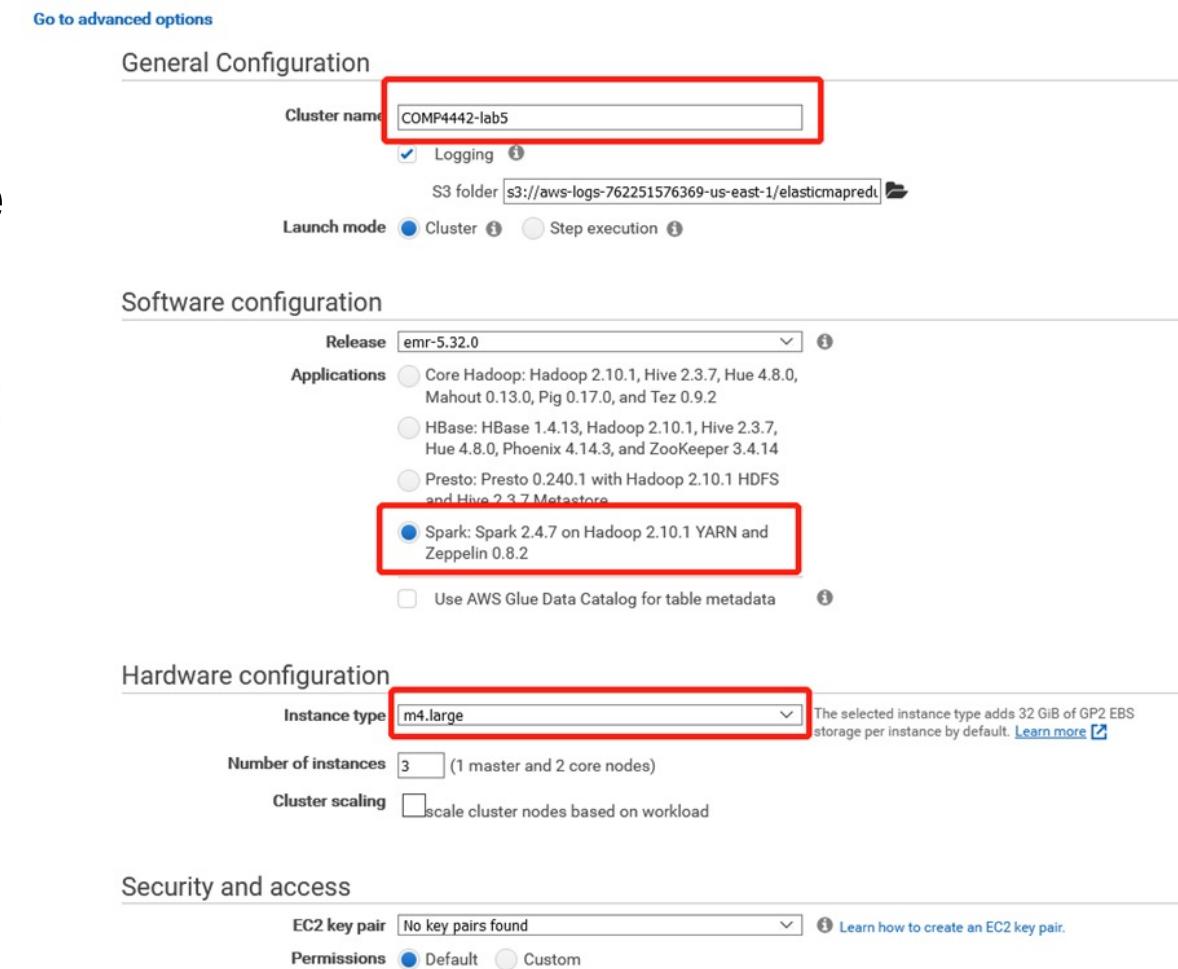
How Elastic MapReduce Works

Upload      Create      Monitor

The page includes three icons: a cloud with an upward arrow, a cluster of nodes with a gear, and a monitor with a line graph and a downward arrow.

# 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



The screenshot shows the AWS EMR Cluster Configuration interface with several sections highlighted:

- General Configuration:** Cluster name is set to "COMP4442-lab5".
- Software configuration:** Release is "emr-5.32.0". Applications section has "Spark: Spark 2.4.7 on Hadoop 2.10.1 YARN and Zeppelin 0.8.2" selected.
- Hardware configuration:** Instance type is "m4.large".
- Security and access:** EC2 key pair is "No key pairs found".

# 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)]
```

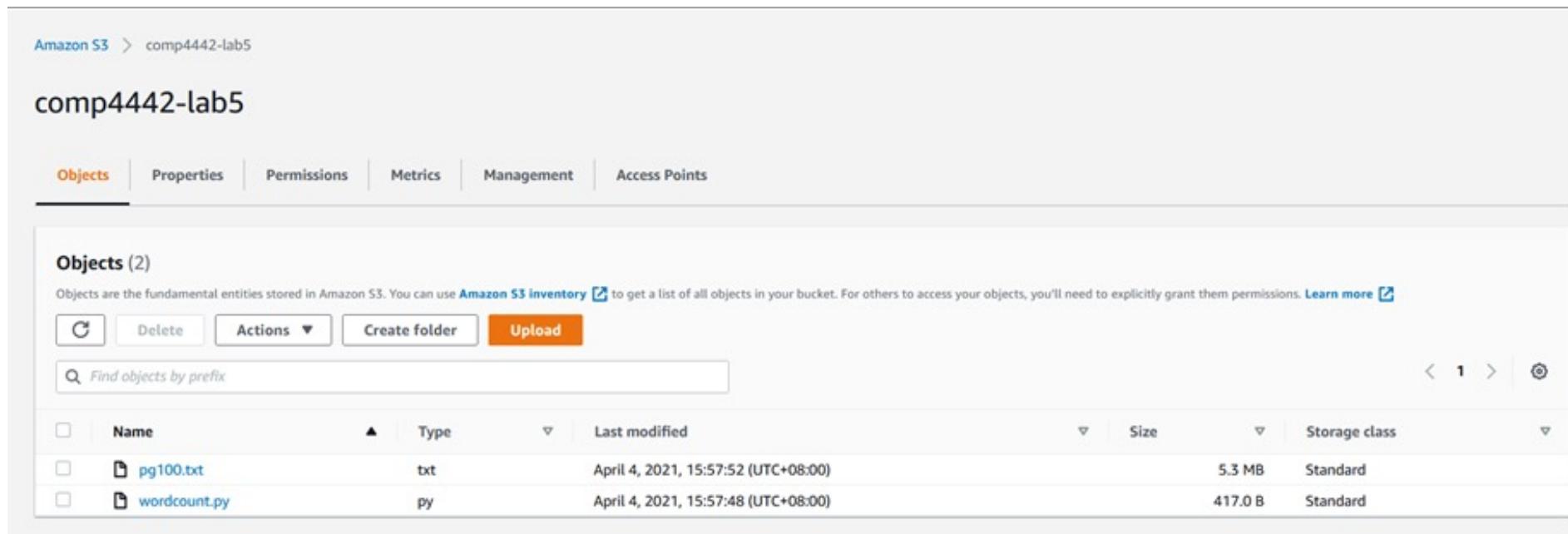
# Operations of Spark RDD

<b>Transformations</b>	$map(f : T \Rightarrow U)$ : $RDD[T] \Rightarrow RDD[U]$ $filter(f : T \Rightarrow Bool)$ : $RDD[T] \Rightarrow RDD[T]$ $flatMap(f : T \Rightarrow Seq[U])$ : $RDD[T] \Rightarrow RDD[U]$ $sample(fraction : Float)$ : $RDD[T] \Rightarrow RDD[T]$ (Deterministic sampling) $groupByKey()$ : $RDD[(K, V)] \Rightarrow RDD[(K, Seq[V])]$ $reduceByKey(f : (V, V) \Rightarrow V)$ : $RDD[(K, V)] \Rightarrow RDD[(K, V)]$ $union()$ : $(RDD[T], RDD[T]) \Rightarrow RDD[T]$ $join()$ : $(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (V, W))]$ $cogroup()$ : $(RDD[(K, V)], RDD[(K, W)]) \Rightarrow RDD[(K, (Seq[V], Seq[W]))]$ $crossProduct()$ : $(RDD[T], RDD[U]) \Rightarrow RDD[(T, U)]$ $mapValues(f : V \Rightarrow W)$ : $RDD[(K, V)] \Rightarrow RDD[(K, W)]$ (Preserves partitioning) $sort(c : Comparator[K])$ : $RDD[(K, V)] \Rightarrow RDD[(K, V)]$ $partitionBy(p : Partitioner[K])$ : $RDD[(K, V)] \Rightarrow RDD[(K, V)]$
<b>Actions</b>	$count()$ : $RDD[T] \Rightarrow Long$ $collect()$ : $RDD[T] \Rightarrow Seq[T]$ $reduce(f : (T, T) \Rightarrow T)$ : $RDD[T] \Rightarrow T$ $lookup(k : K)$ : $RDD[(K, V)] \Rightarrow Seq[V]$ (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



Amazon S3 > comp4442-lab5

comp4442-lab5

Objects Properties Permissions Metrics Management Access Points

**Objects (2)**

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	pg100.txt	txt	April 4, 2021, 15:57:52 (UTC+08:00)	5.3 MB	Standard
<input type="checkbox"/>	wordcount.py	py	April 4, 2021, 15:57:48 (UTC+08:00)	417.0 B	Standard

# 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()
```

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)

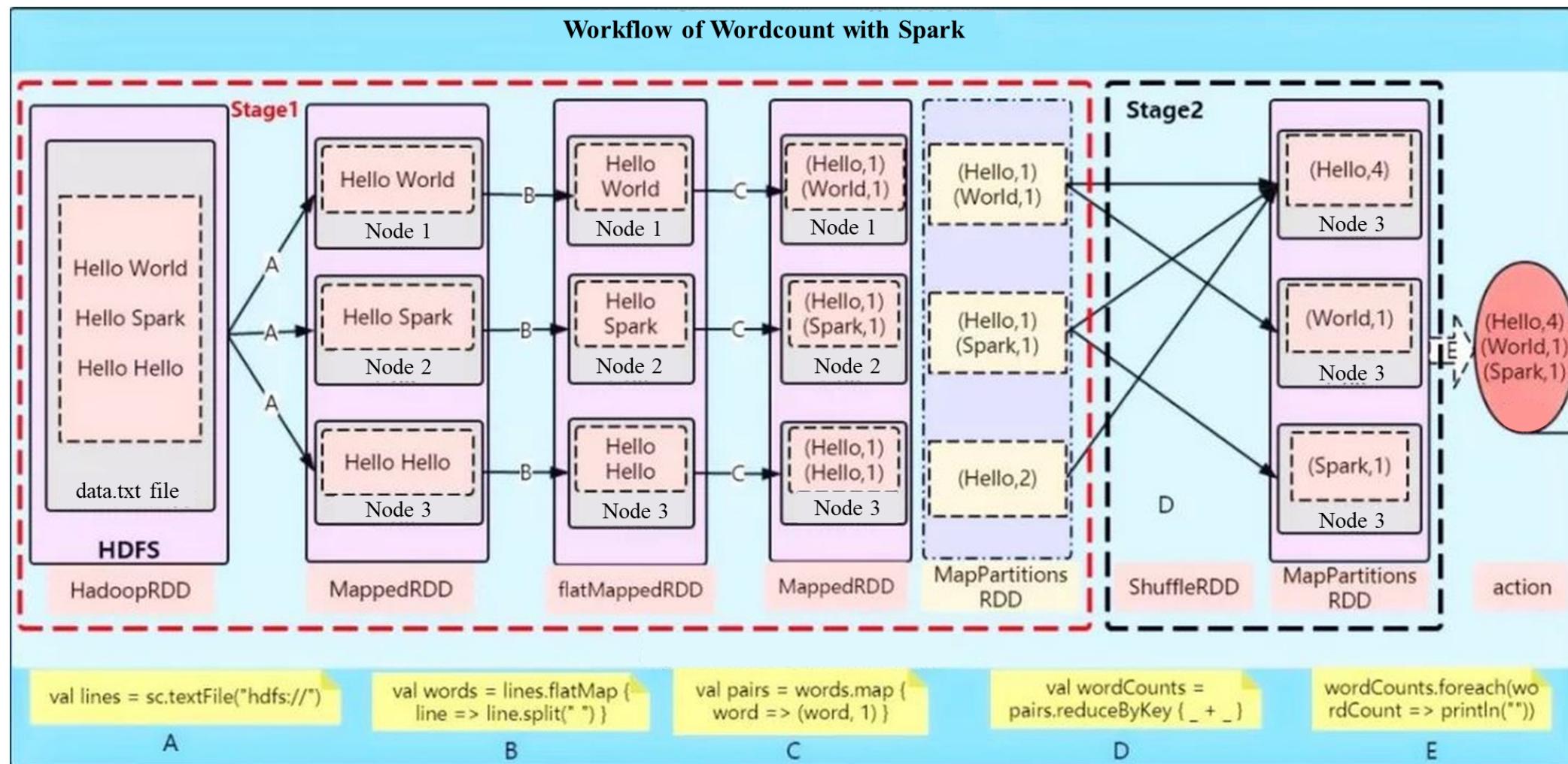
counts.saveAsTextFile(out)

sc.stop()

Read system input parameters

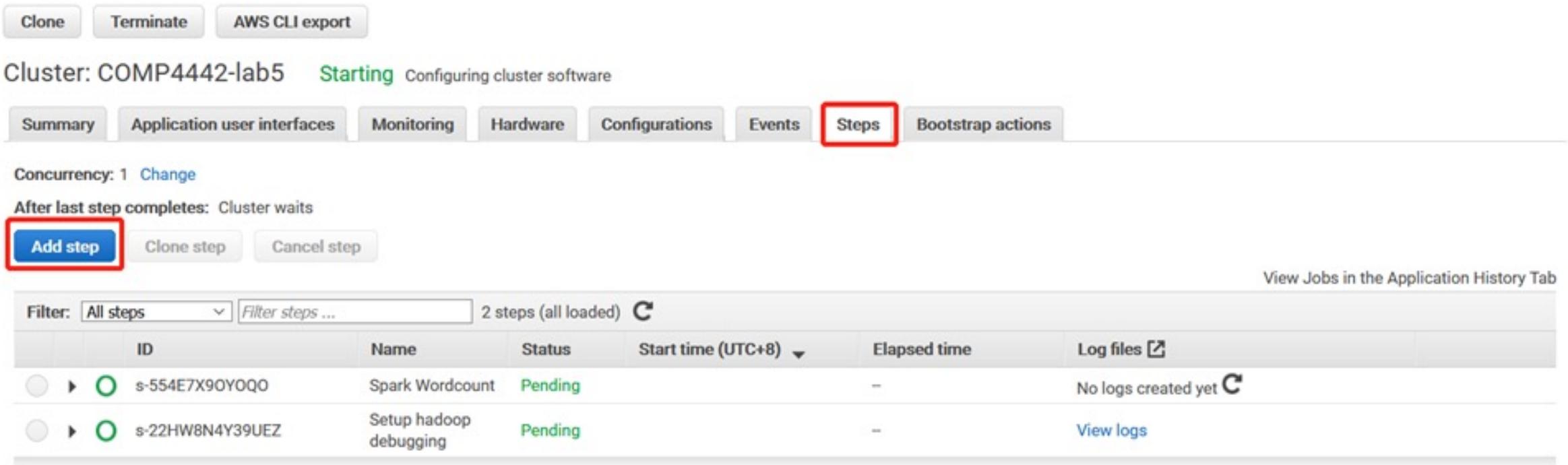
Count frequency

# Workflow of Wordcount with Spark



# Submit Spark Job to Cluster

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

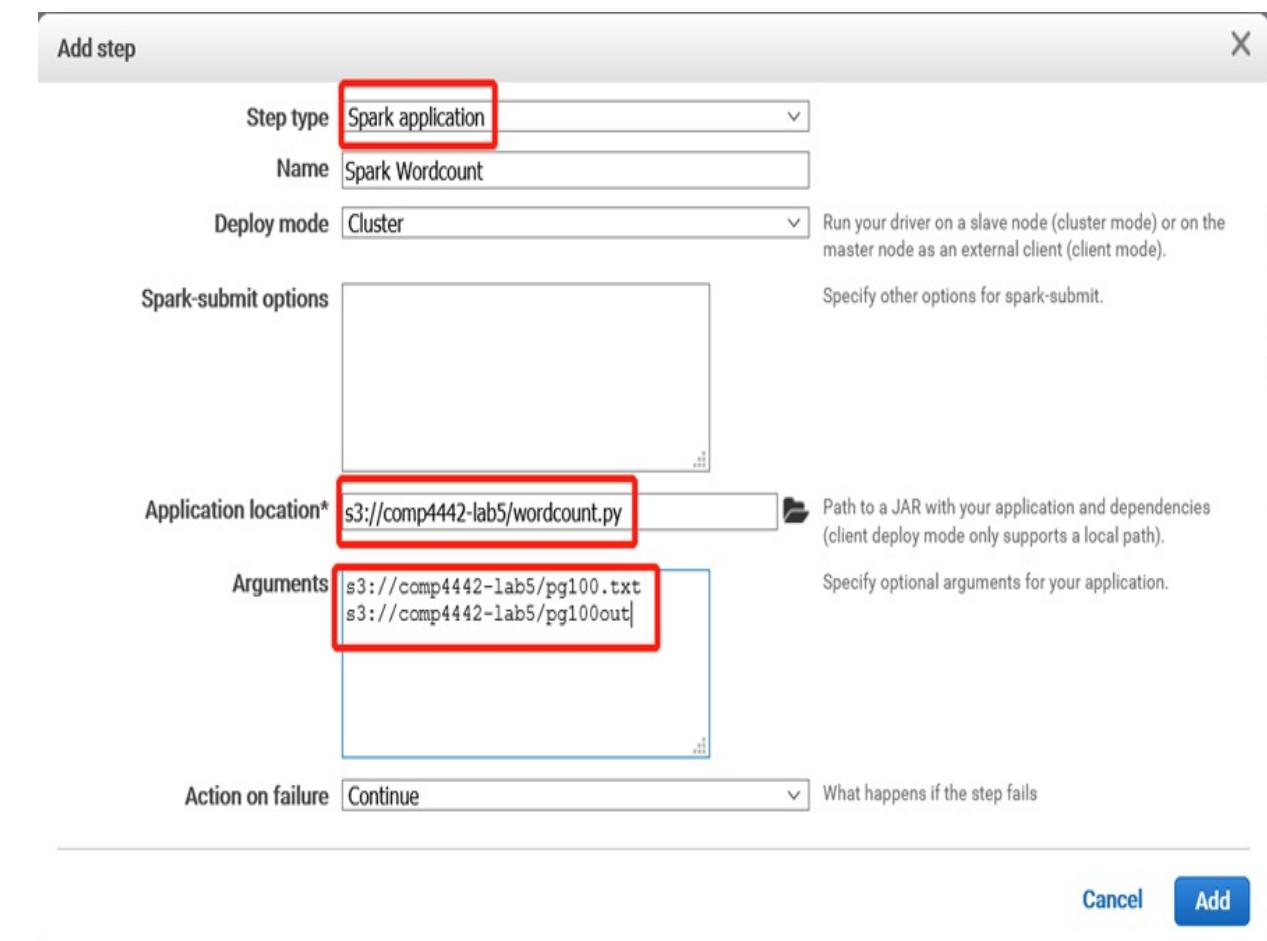


The screenshot shows the Cloudera Manager interface for a cluster named COMP4442-lab5. The status is "Starting" and it is "Configuring cluster software". The "Steps" tab is selected and highlighted with a red box. Below the tabs, there is a message about concurrency and a note that the cluster waits after the last step completes. A prominent blue "Add step" button is also highlighted with a red box. The main table displays two steps: "Spark Wordcount" and "Setup hadoop debugging", both in a "Pending" state. There is a link to view logs for the second step.

ID	Name	Status	Start time (UTC+8)	Elapsed time	Log files
s-554E7X9OYOQO	Spark Wordcount	Pending	-	-	No logs created yet
s-22HW8N4Y39UEZ	Setup hadoop debugging	Pending	-	-	<a href="#">View logs</a>

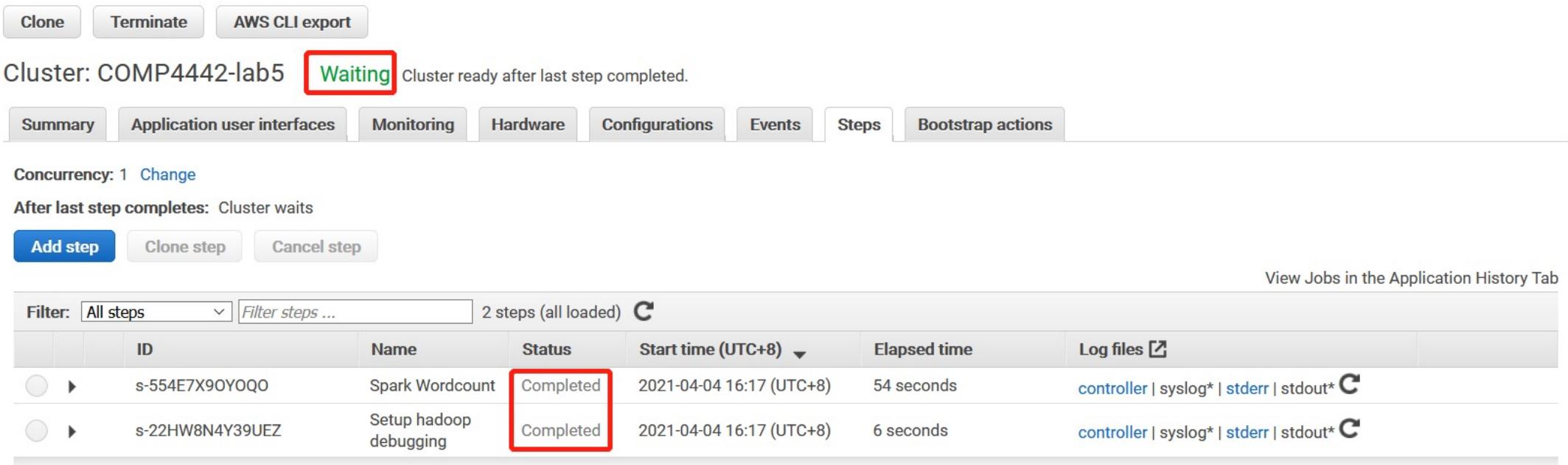
# Submit Spark Job to Cluster

- 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)



The screenshot shows the AWS CloudFormation console interface. At the top, there are buttons for 'Clone', 'Terminate', and 'AWS CLI export'. Below that, it displays 'Cluster: COMP4442-lab5' with the status 'Waiting' highlighted by a red box. A note says 'Cluster ready after last step completed.' Below the cluster name are tabs for 'Summary', 'Application user interfaces', 'Monitoring', 'Hardware', 'Configurations', 'Events', 'Steps', and 'Bootstrap actions'. The 'Steps' tab is selected. It shows 'Concurrency: 1 Change' and 'After last step completes: Cluster waits'. There are buttons for 'Add step', 'Clone step', and 'Cancel step'. A link 'View Jobs in the Application History Tab' is also present. The main table lists two steps:

ID	Name	Status	Start time (UTC+8)	Elapsed time	Log files
s-554E7X9OYOQO	Spark Wordcount	Completed	2021-04-04 16:17 (UTC+8)	54 seconds	<a href="#">controller</a>   <a href="#">syslog*</a>   <a href="#">stderr</a>   <a href="#">stdout*</a>
s-22HW8N4Y39UEZ	Setup hadoop debugging	Completed	2021-04-04 16:17 (UTC+8)	6 seconds	<a href="#">controller</a>   <a href="#">syslog*</a>   <a href="#">stderr</a>   <a href="#">stdout*</a>

# Submit Spark Job to Cluster

- Check the output
  - Download output file from S3

Amazon S3 > comp4442-lab5 > pg100out/

pg100out/

Copy S3 URI

Objects Properties

Objects (3)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

C Delete Actions ▾ Create folder Upload

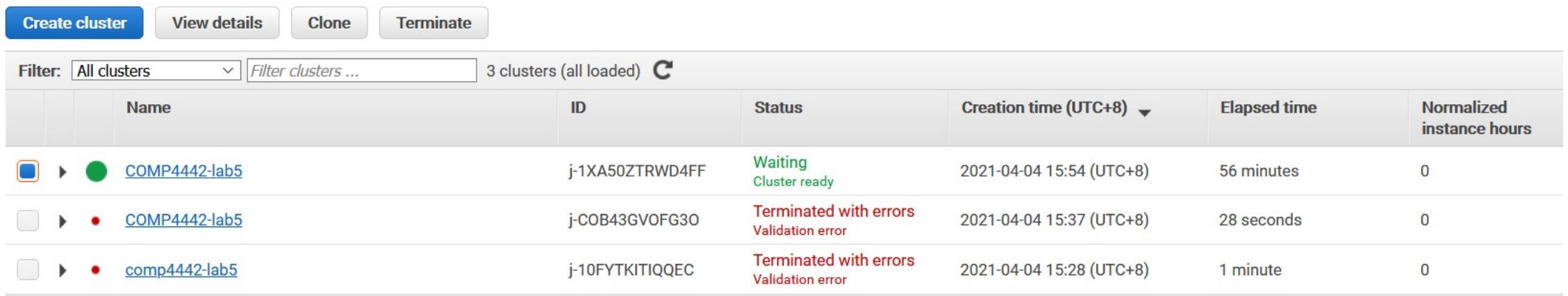
Find objects by prefix

Name	Type	Last modified	Size	Storage class
_SUCCESS	-	April 4, 2021, 16:18:47 (UTC+08:00)	0 B	Standard
part-00000	-	April 4, 2021, 16:18:47 (UTC+08:00)	516.1 KB	Standard
part-00001	-	April 4, 2021, 16:18:47 (UTC+08:00)	507.7 KB	Standard

```
1 ('Gutenberg', 7)
2 ('Complete', 3)
3 ('Works', 3)
4 ('William', 64)
5 ('Shakespeare,', 1)
6 ('by', 2791)
7 ('This', 1101)
8 ('eBook', 2)
9 ('for', 5587)
10 ('the', 23104)
11 ('cost', 32)
12 ('and', 18173)
13 ('with', 6701)
14 ('almost', 135)
15 ('You', 1508)
16 ('copy', 14)
17 ('it', 4875)
18 ('or', 1717)
19 ('under', 196)
20 ('terms', 45)
21 ('License', 1)
22 ('included', 1)
23 ('**', 4)
24 ('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



	Name	ID	Status	Creation time (UTC+8)	Elapsed time	Normalized instance hours
	<a href="#">COMP4442-lab5</a>	j-1XA50ZTRWD4FF	Waiting Cluster ready	2021-04-04 15:54 (UTC+8)	56 minutes	0
	<a href="#">COMP4442-lab5</a>	j-COB43GV0FG3O	Terminated with errors Validation error	2021-04-04 15:37 (UTC+8)	28 seconds	0
	<a href="#">comp4442-lab5</a>	j-10FYTKITIQQEC	Terminated with errors Validation error	2021-04-04 15:28 (UTC+8)	1 minute	0

# 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?