




# Big Data

## Spark deployment

**Stéphane Vialle & Gianluca Quercini**



1

## Spark deployment

1. DAG of lazy operations
2. Ex of Spark execution on cluster (standalone mode)
3. Ex of Spark execution on cloud

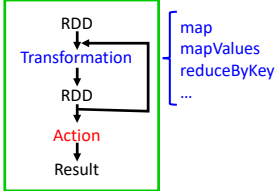
A1 - Ex of Spark execution on cluster (YARN manager)  
A2 - Ex of Spark execution on cluster (MESOS manager)

2

## DAG of lazy operations

- A **RDD** is a dataset distributed among the Spark compute nodes
- **Transformations** are **lazy** operations: saved and executed further
- **Actions** trigger the execution of the sequence of transformations

A **job** is a sequence of RDD transformations, ended by an action



A **Spark application** is a **set of jobs** to run sequentially or in parallel  
→ A DAG of tasks

3

## DAG of lazy operations

The **Spark application driver** controls the application run

- It creates the Spark context
- It analyses the Spark program
- It **creates a DAG of tasks** for each job
- It **optimizes** the DAG
  - pipelining narrow transformations
  - identifying the tasks that can be run in parallel
- It **schedules** the DAG of tasks on the available worker nodes (the **Spark Executors**) **in order to maximize parallelism** (and to reduce the execution time)

4

## Task DAG execution

### Spark job trace: on 10 Spark executors, with 3GB input file

```

DAGScheduler: Submitting 24 missing tasks from ShuffleMapStage 0 ...
TaskSchedulerImpl: Adding task set 0.0 with 24 tasks
...
TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 172.20.10.14, executor 0, partition 1, ...)
TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, 172.20.10.11, executor 7, partition 2, ...)
...
TaskSetManager: Starting task 10.0 in stage 0.0 (TID 10, 172.20.10.11, executor 7, partition 10, ...)

TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 18274 ms ... (executor 7) (1/24)
TaskSetManager: Starting task 11.0 in stage 0.0 (TID 11, 172.20.10.7, executor 8, partition 11, ...)
TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 18459 ms ... (executor 8) (2/24)
...
TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
...
    
```

Submitting the 10 first tasks on the 10 Spark executor processes

Submitting a new task when a previous one has finished

End of task graph execution

5

## Spark deployment

1. DAG of lazy operations
2. Ex of Spark execution on cluster (standalone mode)
3. Ex of Spark execution on cloud

A1 - Ex of Spark execution on cluster (YARN manager)  
A2 - Ex of Spark execution on cluster (MESOS manager)

6

### Using the Spark Master as cluster manager (standalone mode)

`spark-submit --master spark://node:port ... myApp`

**Spark cluster configuration:**

- Add the list of cluster worker nodes in the Spark Master config.
- Specify the maximum amount of memory per Spark Executor  
`spark-submit --executor-memory XX ...`
- Specify the **total amount of CPU cores** used to process **one Spark application** (through all its Spark executors)  
`spark-submit --total-executor-cores YY ...`

7

### Using the Spark Master as cluster manager (standalone mode)

`spark-submit --master spark://node:port ... myApp`

**Spark cluster configuration:**

- Default config :
  - (only) 1GB/Spark Executor
  - Unlimited nb of CPU cores per application execution
  - **The Spark Master creates one mono-core Executor on all Worker nodes to process each job ...**
- You can limit the total nb of cores per job
- You can concentrate the cores into few multi-core Executors

8

### Using the Spark Master as cluster manager (standalone mode)

`spark-submit --master spark://node:port ... myApp`

**Spark cluster configuration:**

- Default config :
  - (only) 1GB/Spark Executor
  - Unlimited nb of CPU cores per application execution
  - The Spark Master creates one **multi-core** Executor on all Worker nodes to process each job (**invading all cores!**)
  - **But you can fix the nb of cores per Spark Executor**
- You can limit the total nb of cores per job
- You can concentrate the cores into few multi-core Executors

*EXP 2019*

9

### Using the Spark Master as cluster manager (standalone mode)

`spark-submit --master spark://node:port ... myApp`

**Client deployment mode:**

**Spark app. Driver**

- DAG builder
- DAG scheduler-optimizer
- Task scheduler

*Interactive control of the application: development mode*

10

### Using the Spark Master as cluster manager (standalone mode)

`spark-submit --master spark://node:port ... myApp`

**Cluster deployment mode:**

*Laptop connection can be turn off: production mode*

11

### Using the Spark Master as cluster manager (standalone mode)

`spark-submit --master spark://node:port ... myApp`

The Cluster Worker nodes should be the Data nodes, storing initial RDD values or new generated (and saved) RDD

- Will improve the global data-computations locality
- **When using HDFS: the Hadoop data nodes should be re-used as worker nodes for Spark Executors**

12

### Using the Spark Master as cluster manager (standalone mode)

```
spark-submit --master spark://node:port ... myApp
```

The Cluster Worker nodes should be the Data nodes, storing initial RDD values or new generated (and saved) RDD

**When using the Spark Master as Cluster Manager:**  
 ...there is no way to localize the Spark Executors on the data nodes hosting the right RDD blocks!

13

### Using the Spark Master as cluster manager (standalone mode)

```
spark-submit --master spark://node:port ... myApp
```

**Cluster deployment mode:**

14

### Using the Spark Master as cluster manager (standalone mode)

```
spark-submit --master spark://node:port ... myApp
```

**Strength and weakness of standalone mode:**

- Nothing more to install (included in Spark)
- Easy to configure
- Can run different jobs concurrently
- Can not share the cluster with non-Spark applications
- Can not launch Executors on the data nodes hosting input data
- Limited scheduling mechanism (unique queue)

15

### Spark deployment

1. DAG of lazy operations
2. Ex of Spark execution on cluster (standalone mode)
3. Ex of Spark execution on cloud

A1 - Ex of Spark execution on cluster (YARN manager)  
 A2 - Ex of Spark execution on cluster (MESOS manager)

16

### Using Amazon Elastic Compute Cloud « EC2 »

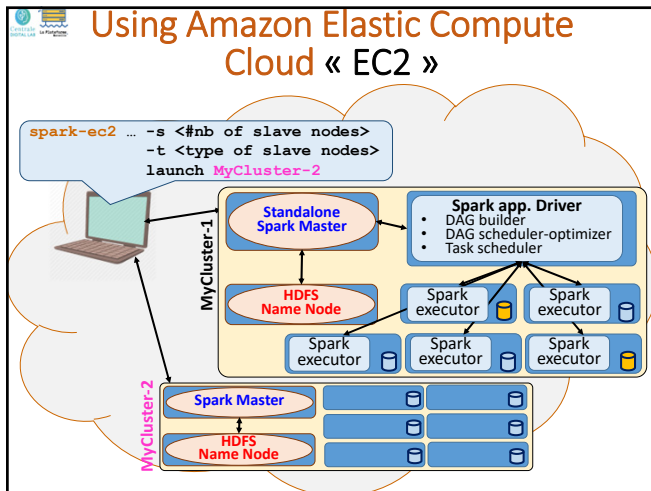
```
spark-ec2 ... -s <#nb of slave nodes> -t <type of slave nodes> launch MyCluster-1
```

17

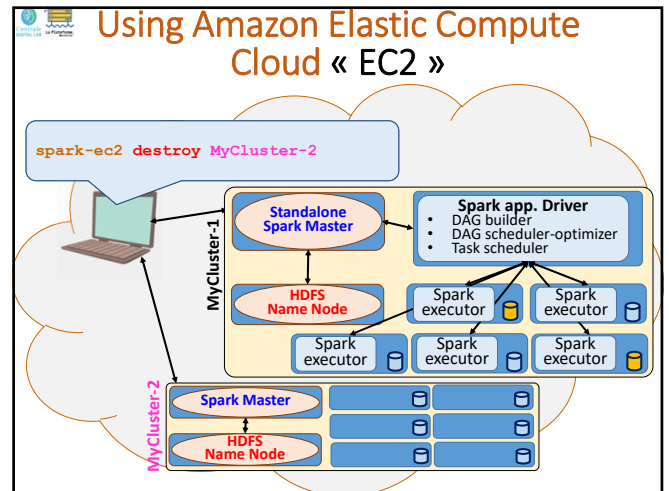
### Using Amazon Elastic Compute Cloud « EC2 »

```
spark-ec2 ... -s <#nb of slave nodes> -t <type of slave nodes> launch MyCluster-1
```

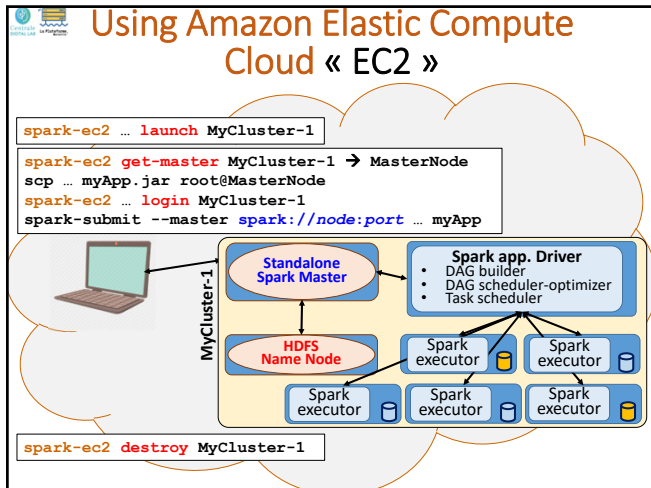
18



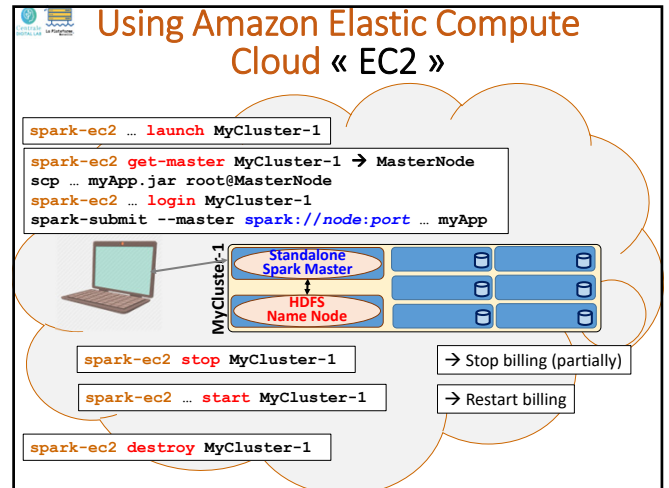
19



20



21



22

### Using Amazon Elastic Compute Cloud « EC2 »

Start to learn to deploy HDFS and Spark architectures  
 Then, learn to deploy these architecture in a CLOUD  
 ... or use a "Spark Cluster service": ready to use in a CLOUD!

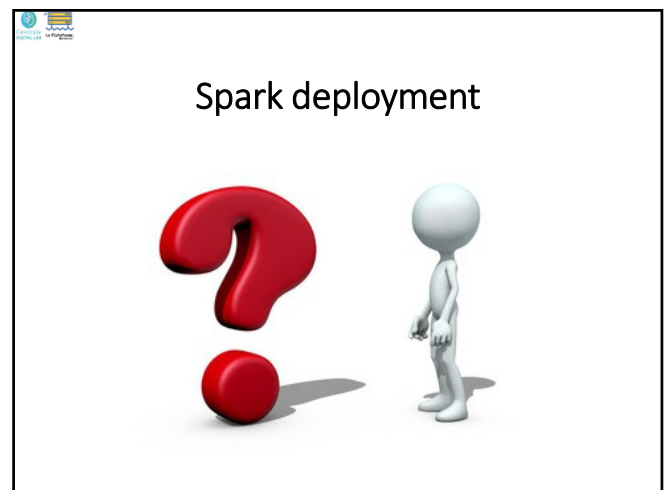
Learn to minimize the cost (€) of a Spark cluster:

- Allocate the right number of nodes
- Stop Spark cluster when you do not use, and re-start further
- Do not forget to release the allocated machines at the end

Choose to allocate reliable or preemptible machines:

- Reliable machines during all the session (standard)
- Preemptibles machines (5x less expensive!)  
 → require to support to loose some tasks, or to checkpoint...

23



24

## Spark deployment – Appendix

1. DAG of lazy operations
2. Ex of Spark execution on cluster (standalone mode)
3. Ex of Spark execution on cloud

**A1 - Ex of Spark execution on cluster (YARN manager)**  
**A2 - Ex of Spark execution on cluster (MESOS manager)**

25

## Using YARN as cluster manager

```
export HADOOP_CONF_DIR = ${HADOOP_HOME}/conf
spark-submit --master yarn ... myApp
```

**Spark cluster configuration:**

- Add an env. variable defining the path to Hadoop conf directory
- Specify the maximum amount of memory per Spark Executor
- Specify the amount of CPU cores used per Spark executor  
`spark-submit --executor-cores YY ...`
- Specify the nb of Spark Executors per job: `--num-executors`

26

## Using YARN as cluster manager

```
export HADOOP_CONF_DIR = ${HADOOP_HOME}/conf
spark-submit --master yarn ... myApp
```

**Spark cluster configuration:**

- By default:
  - (only) 1GB/Spark Executor
  - (only) 1 CPU core per Spark Executor
  - (only) 2 Spark Executors per job
- Usually better with few large Executors (RAM & nb of cores)...

27

## Using YARN as cluster manager

```
export HADOOP_CONF_DIR = ${HADOOP_HOME}/conf
spark-submit --master yarn ... myApp
```

**Spark cluster configuration:**

- Link Spark RDD meta-data « preferred locations » to HDFS meta-data about « localization of the input file blocks »

```
val sc = new SparkContext(sparkConf,
    InputFormatInfo.computePreferredLocations(
        Seq(new InputFormatInfo(conf,
            classOf[org.apache.hadoop.mapred.TextInputFormat], hdfsPath)...
    )
    Spark Context construction
```

28

## Using YARN as cluster manager

```
export HADOOP_CONF_DIR = ${HADOOP_HOME}/conf
spark-submit --master yarn ... myApp
```

**Client deployment mode:**

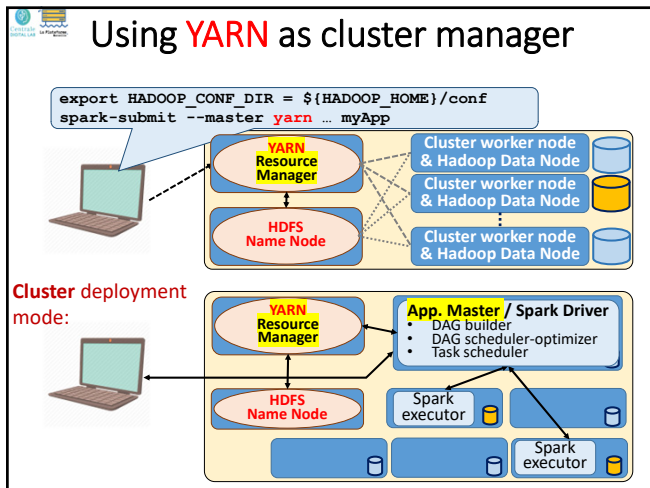
29

## Using YARN as cluster manager

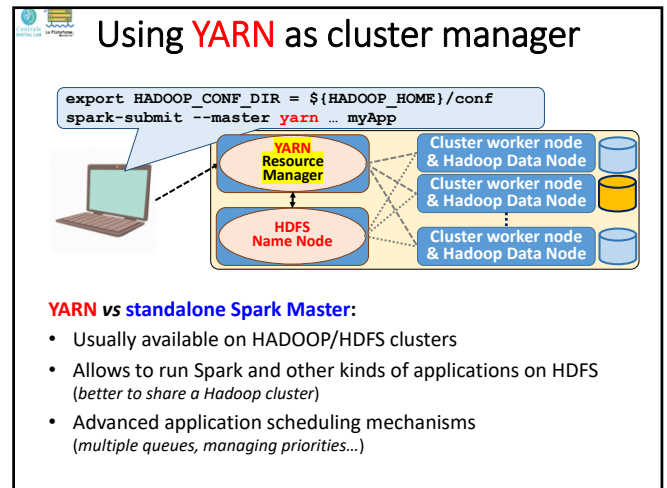
```
export HADOOP_CONF_DIR = ${HADOOP_HOME}/conf
spark-submit --master yarn ... myApp
```

**Client deployment mode:**

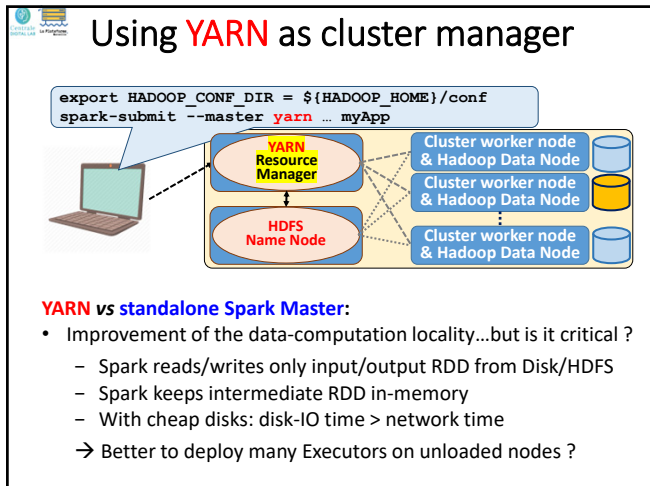
30



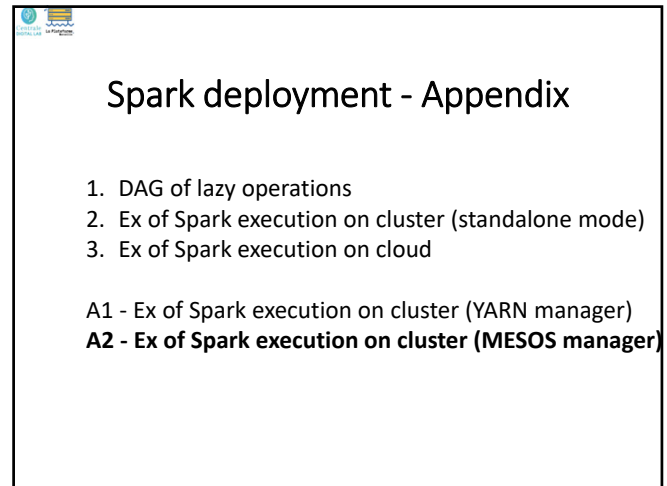
31



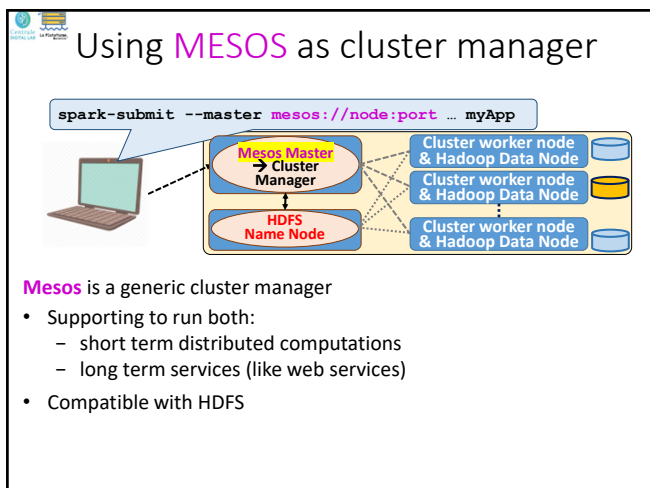
32



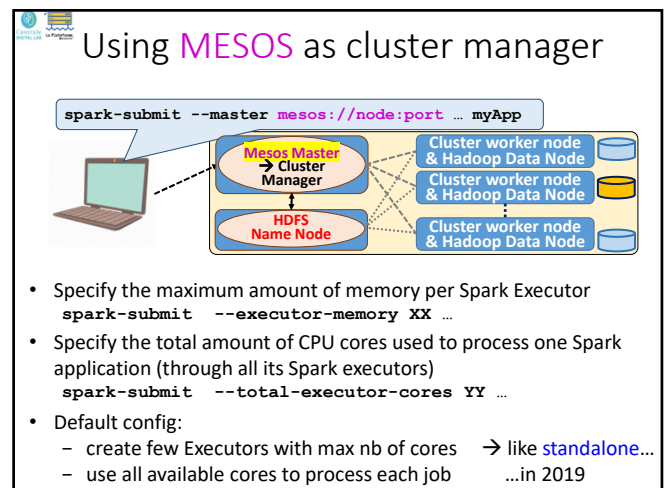
33



34

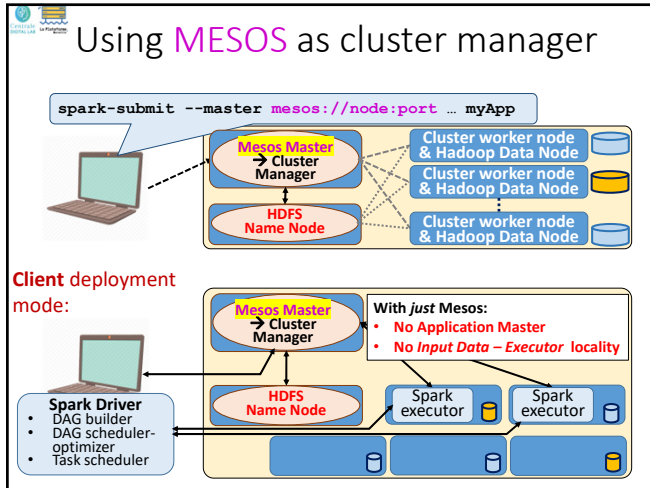


35

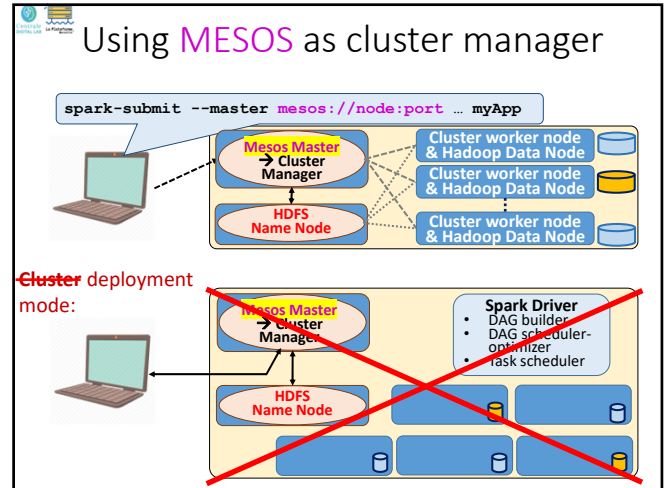


36

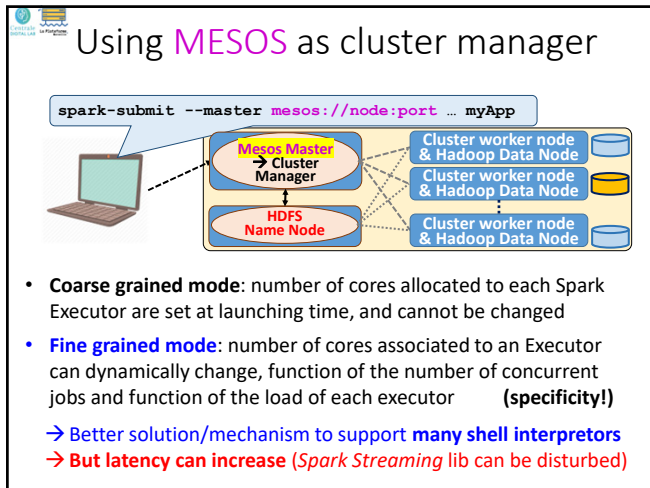




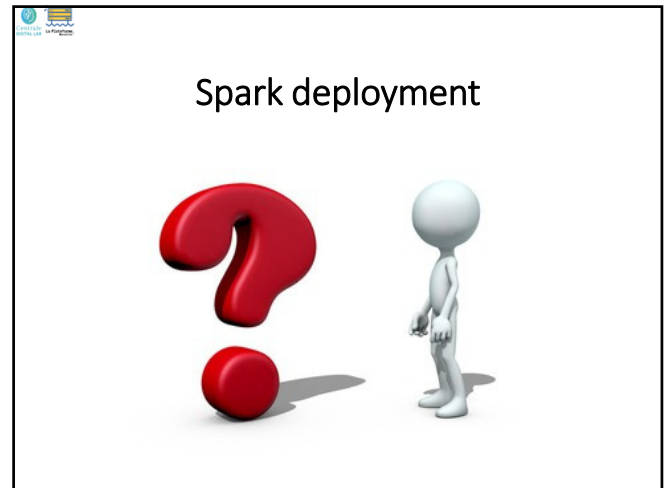
37



38



39



40