Distributed Computing

DIANE WOODBRIDGE, PH.D

Distributed Computing

Course Objectives

Understand needs and concepts of distributed computing.

Understand the Spark and its stack.

Being competent to work with Spark on a distributed computing environment.

- Programming with RDDs.
- Work with key/value pairs.
- Work with DataFrame.
- Work with SparkSQL, MLlib, ML, Spark Streaming, (and GarphX).
- Work on Amazon AWS.

Spark Interview Questions

What is Apache Spark?

Explain the key features of Spark.

~~**What is RDD?**~~

~~**How to create RDD.**~~

~~**What is "partitions"?**~~

~~**Types or RDD operations?**~~

~~**What is "transformation"?**~~

~~**What is "action"?**~~

~~**Functions of "spark core"?**~~

~~**What is "spark context"?**~~

What is an "RDD lineage"?

Which file systems does Spark support?

List the various types of "Cluster Managers" in Spark.

What is "YARN"?

What is "Mesos"?

What is a "worker node"?

What is an "accumulator"?

What is "Spark SQL" (Shark)?

What is "SparkStreaming"?

What is "GraphX"?

What is "MLlib"?

https://www.edureka.co/blog/interview-questions/top-apache-spark-interview-questions-2016/

Spark Interview Questions

~~What are the advantages of using Apache Spark over Hadoop MapReduce for big data processing?~~

~~What are the languages supported by Apache Spark for developing big data applications?~~

Can you use Spark to access and analyze data stored in Cassandra databases?

Is it possible to run Apache Spark on Apache Mesos?

~~How can you minimize data transfers when working with Spark?~~

Why is there a need for broadcast variables?

Name a few companies that use Apache Spark in production.

What are the various data sources available in SparkSQL?

What is the advantage of a Parquet file?

~~What do you understand by Pair RDD?~~

Is Apache Spark a good fit for Reinforcement learning?

https://www.dezyre.com/article/top-50-spark-interview-questions-and-answers-for-2016/208

RDD Dependencies

RDD dependency (lineage).

- Every time a transformation is performed on an RDD , a dependency between an old and a new RDD is created.
  - Spark's execution model is based on directed acyclic graphs (DAG s)
  - In Spark DAG s, RDD s are vertices and dependencies are edges. Every time a transformation is performed on an RDD , a new vertex (a new RDD) and a new edge (a dependency) are created.
- Dependency types
  - Narrow – When no data shuffle between partitions is required.
  - Wide - When it requires shuffle when joining RDDs.

RDD Dependencies – Example 1

```
import random
```

```
list = [random.randrange(10) for x in range(500)]
```

```
listrdd = sc.parallelize(list, 5)
```

```
pairs = listrdd.map(lambda x : (x,x*x))
```

```
reduced = pairs.reduceByKey(lambda v1, v2 : v1+v2)
```

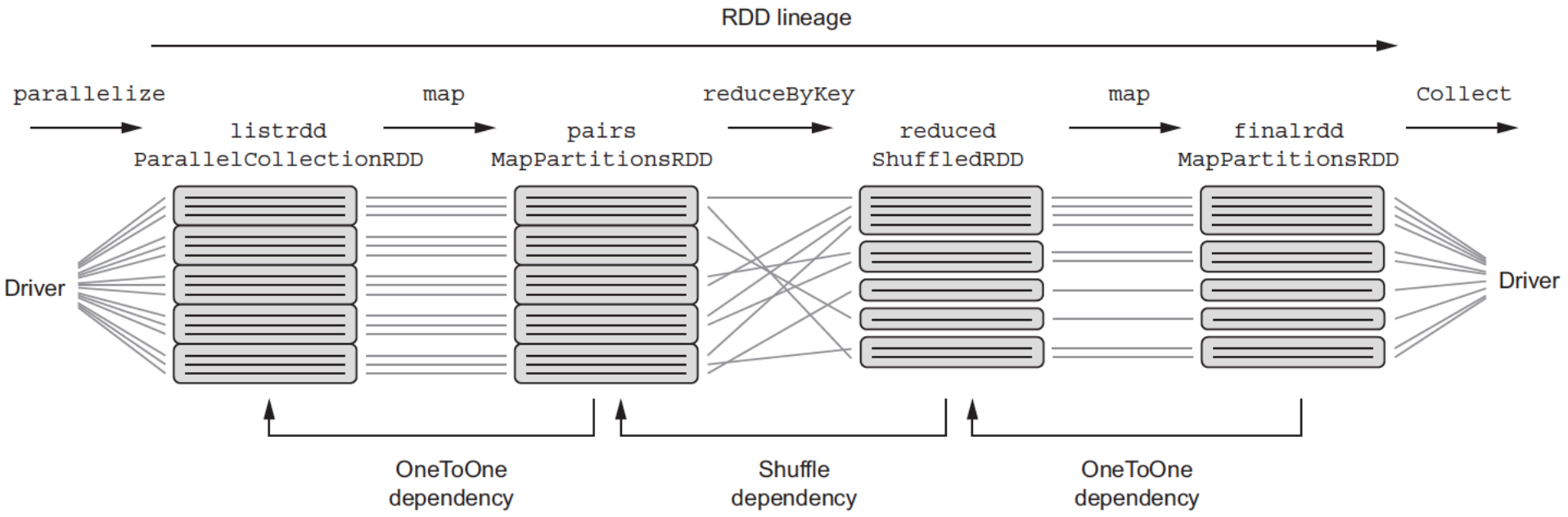
```
finalrdd = reduced.mapPartitions(lambda itr: ["K="+str(k)+",V="+str(v) for (k,v) in itr])
```

```
finalrdd.collect()
```

```
import random

list = [random.randrange(10) for x in range(500)]

listrdd = sc.parallelize(list, 5)

pairs = listrdd.map(lambda x : (x,x*x))

reduced = pairs.reduceByKey(lambda v1, v2 : v1+v2)

finalrdd = reduced.mapPartitions(lambda itr: ["K="+str(k)+",V="+str(v) for (k,v) in itr])

finalrdd.collect()
```

Output

```
['K=0,V=0',
 'K=5,V=1300',
 'K=1,V=48',
 'K=6,V=1836',
 'K=2,V=212',
 'K=7,V=2156',
 'K=8,V=3072',
 'K=3,V=432',
 'K=9,V=4455',
 'K=4,V=720']
```



RDD Dependencies – Example 1

toDebugString()

- Shows a textual representation of RDD dependencies.
- The RDDs in the output appears in reverse order.
- Useful in trying to minimize the number of shuffles.
  - The numbers in parentheses show the number of partitions of the corresponding RDD.
- Every time you see a ShuffleRDD in the lineage chain, you can be sure that a shuffle will be performed at that point.

```
print finalrdd.toDebugString()
```

```
(5) PythonRDD[41] at collect at <ipython-input-45-036dee88a049>:1 []
 |  MapPartitionsRDD[40] at mapPartitions at PythonRDD.scala:422 []
 |  ShuffledRDD[39] at partitionBy at NativeMethodAccessorImpl.java:-2 []
 +-(5) PairwiseRDD[38] at reduceByKey at <ipython-input-43-b3cb29adde6b>:1 []
    |  PythonRDD[37] at reduceByKey at <ipython-input-43-b3cb29adde6b>:1 []
    |  ParallelCollectionRDD[36] at parallelize at PythonRDD.scala:475 []
```

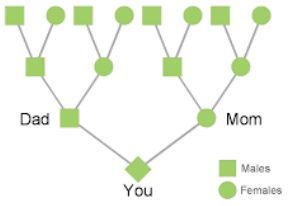
RDD Dependencies – Example 1

toDebugString()

- Shows a textual representation of RDD dependencies.
- The RDDs in the output appears in reverse order.
- Useful in trying to minimize the number of shuffles.
  - The numbers in parentheses show the number of partitions of the corresponding RDD.
- Every time you see a ShuffleRDD in the lineage chain, you can be sure that a shuffle will be performed at that point.

```
print finalrdd.toDebugString()
```

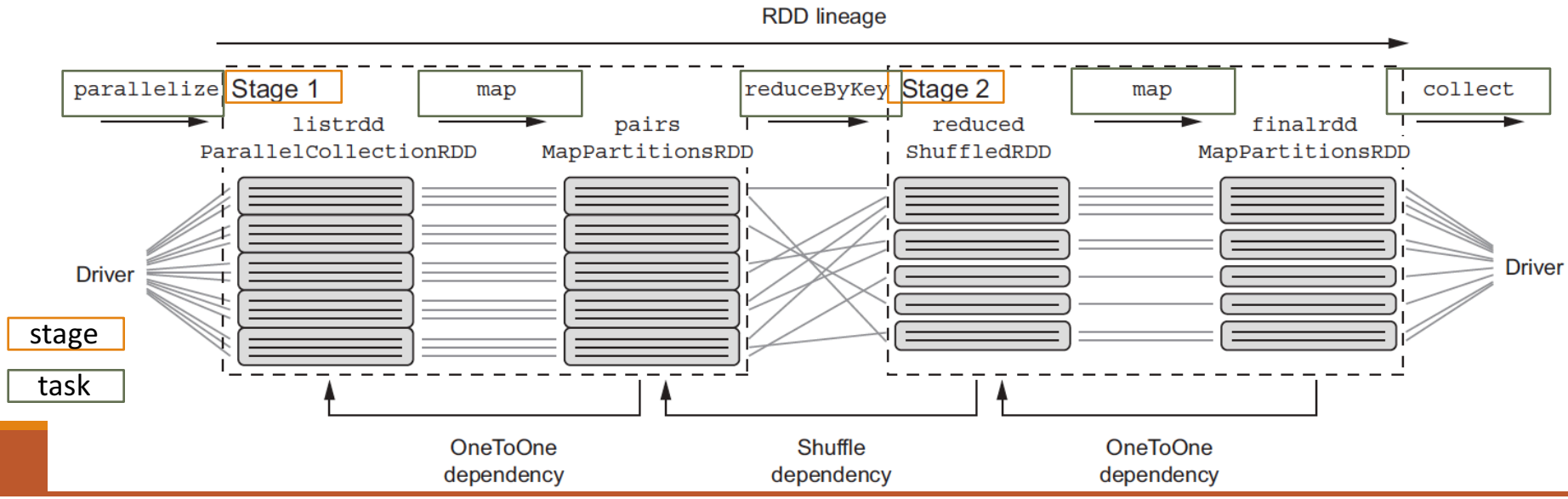
```
(5) PythonRDD[41] at collect at <ipython-input-45-036dee88a049>:1 []
 |  MapPartitionsRDD[40] at mapPartitions at PythonRDD.scala:422 []
 |  ShuffledRDD[39] at partitionBy at NativeMethodAccessorImpl.java:-2 []
 +-(5) PairwiseRDD[38] at reduceByKey at <ipython-input-43-b3cb29adde6b>:1 []
    |  PythonRDD[37] at reduceByKey at <ipython-input-43-b3cb29adde6b>:1 []
    |  ParallelCollectionRDD[36] at parallelize at PythonRDD.scala:475 []
```



RDD Dependencies

Spark stages and tasks.

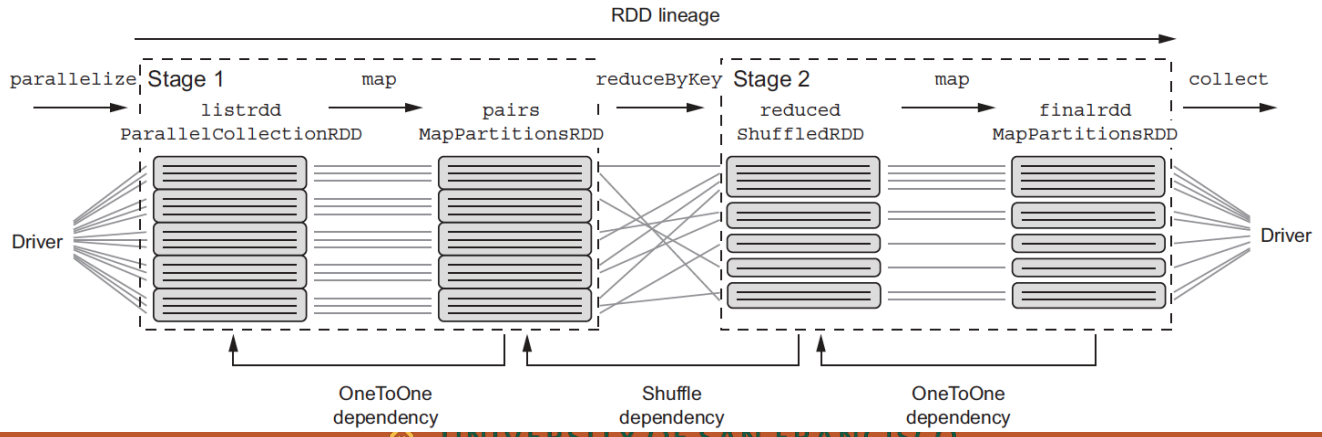
- Every job is divided into stages based on the points where shuffles occur.



RDD Dependencies

Spark stages and tasks.

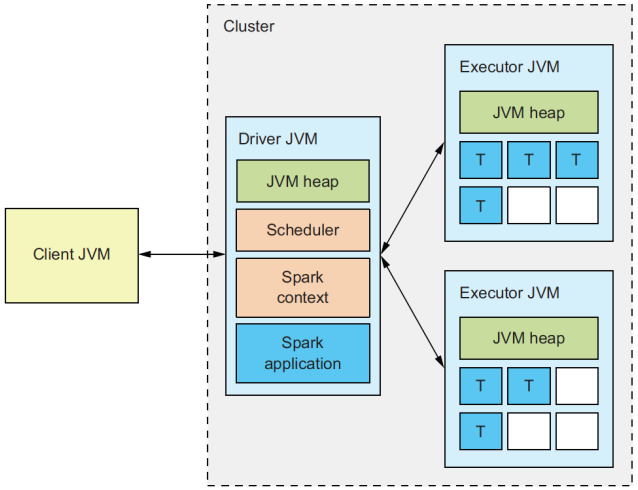
- Stage 1 encompasses transformations that result in a shuffle: parallelize , map , and reduceByKey .
- The results of Stage 1 are saved on disk as intermediate files on executor machines.

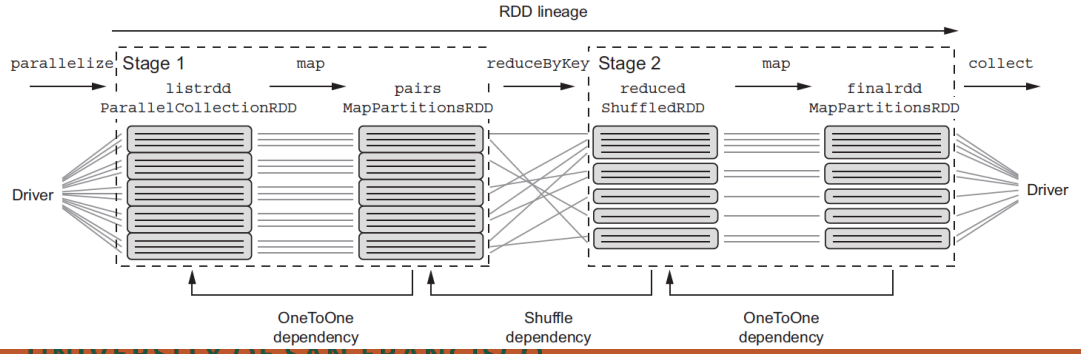


RDD Dependencies

Spark stages and tasks.

- During Stage 2, each partition receives data from these intermediate files belonging to it, and the execution is continued.
- For each stage, tasks are created and sent to the executors.
- After all tasks of a particular stage complete, the driver creates tasks for the next stage and sends them to the executors, and so on.





RDD Dependencies

RDD lineage can grow arbitrarily long, by chaining any number of transformations, Spark provides a way to persist the entire RDD to stable storage. Then, in case of node failure, Spark doesn't need to recompute the missing RDD pieces from the start. It uses a snapshot and computes the rest of the lineage from there. This feature is called checkpointing .

During checkpointing, the entire RDD is persisted to disk—not just its data, as is the case with caching, but its lineage, too. After checkpointing, the RDD 's dependencies are erased, as well as the information about its parent(s), because they won't be needed for its recomputation any more.

If the RDD is checkpointed, reading it from a file could be much quicker than rebuilding it by using all the (possibly complicated) transformations.

Word Count

Stages and tasks?

Advanced Spark Programming

Using shared variables to communicate with Spark executors.

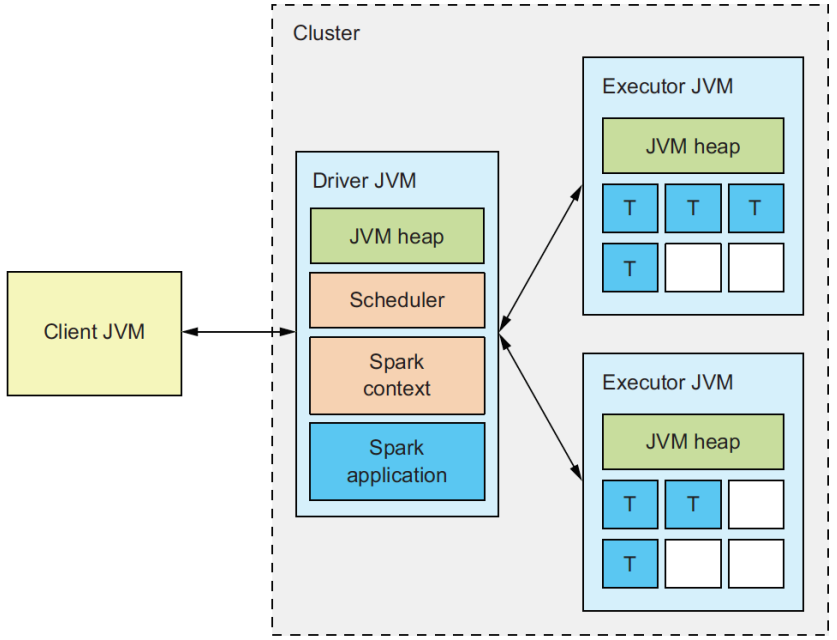
- Normally, when a function passed to a Spark operation is executed, it works on separate copies of all the variables used in the function. These variables are copied to each partition, and no updates to the variables are propagated back to the driver program.
- Solution : Shared variables
  - Help maintain a global state or hare data across tasks and partitions.
  - Accumulator : aggregate information from executor nodes to the driver node.
  - Broadcast variable : efficiently distribute large read-only values to executor nodes.

http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Advanced Spark Programming

Using shared variables to communicate with Spark executors.

- Shared variables
  - Accumulator : aggregate information from executor nodes to the driver node.
    - Is shared across executors that you can only add to.
    - Can be accessed by driver node, not from an executor node.
    - Can be used to implement **global** sums and counters.
    - Create using sc.accumulator(inital_value)
    - The executor can add to the accumulator with .add(val) method. (it is write-only .)
    - The driver can call it using .value



http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Example 2

Define a accumulator variable and initialize the value to be 0.

Generate values between 1 and 10000000 using .parallelize().

For each value, increment accumulator variable.

Example 2

```
accum = sc.accumulator(0)
```

```
list = sc.parallelize(range(1,10000000))
```

```
list.foreach(lambda x : accum.add(1))
```

```
accum.value
```

```
9999999
```

Exception occurs when you tried to access accumulator values in executor nodes.

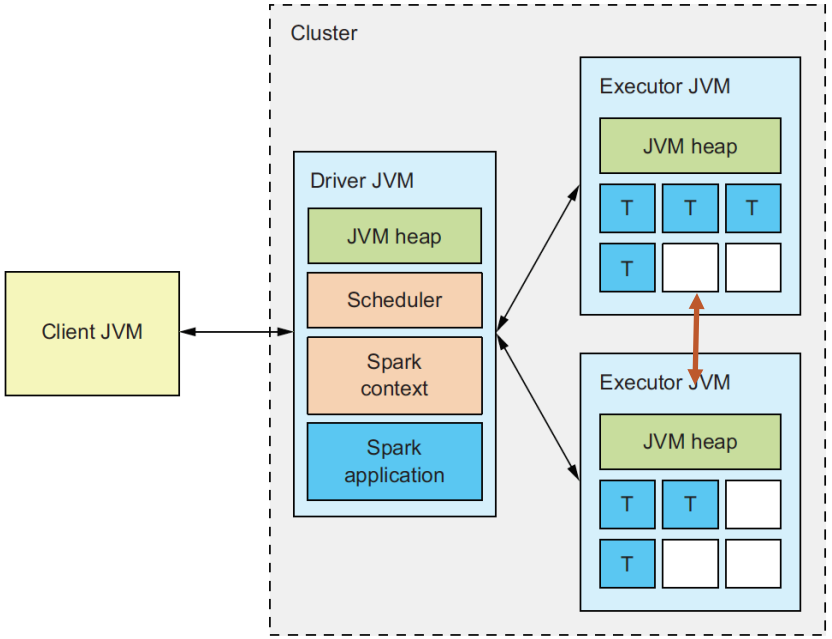
```
list.foreach(lambda x : accum.value)
```

```
---------------------------------------------------------------------------
Py4JJavaError                             Traceback (most recent call last)
<ipython-input-56-856481d5c40d> in <module>()
----> 1 list.foreach(lambda x : accum.value)

/usr/local/Cellar/apache-spark/2.0.2/libexec/python/pyspark/rdd.pyc in foreach(self, f)
    748                 f(x)
    749             return iter([])
--> 750         self.mapPartitions(processPartition).count()  # Force evaluation
    751
```

Advanced Spark Programming

Using shared variables to communicate with Spark executors.

- Shared variables
  - Broadcast variable : efficiently distribute large read-only values such as lookup tables to executor nodes.
    - The value is sent to each node only one using broadcast variables.
    - Create a broadcast variable using sc.broadcast(value).
    - Access the value with .value.



http://spark.apache.org/docs/latest/programming-guide.html#shared-variables

Example 3

Generate a broadcast variable access it.

```
list = sc.parallelize(range(10,100))
```

```
broadcastVar = sc.broadcast([1, 2, 3])
```

```
type(broadcastVar)
```

```
pyspark.broadcast.Broadcast
```

```
broadcastVar.value
```

```
[1, 2, 3]
```

```
add_broadcastVar = list.map(lambda x : x + broadcastVar.value[0])
```

```
add_broadcastVar.collect()
```

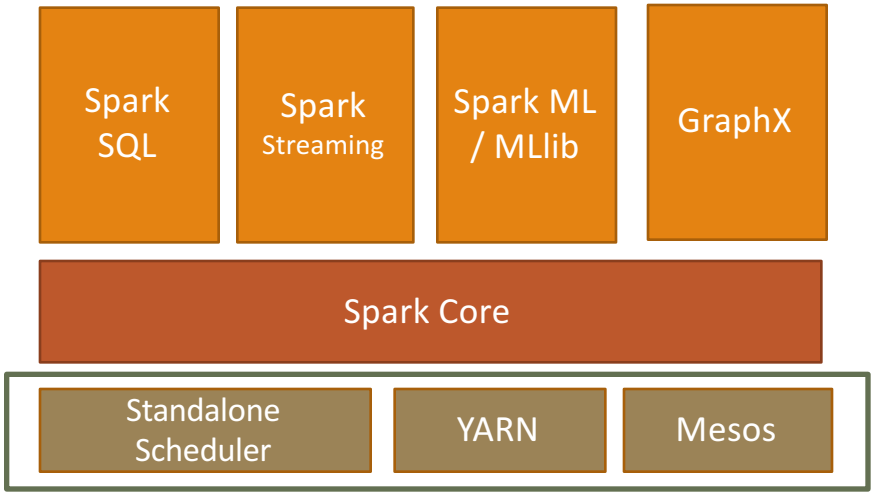
Running on a Spark Cluster

So far, we've programmed in a local mode.
(not a cluster mode!)

Spark cluster

- A set of interconnected processes, running in a distributed manner on different machines.
- Types (3)
  - Spark standalone
  - Hadoop YARN
  - Mesos

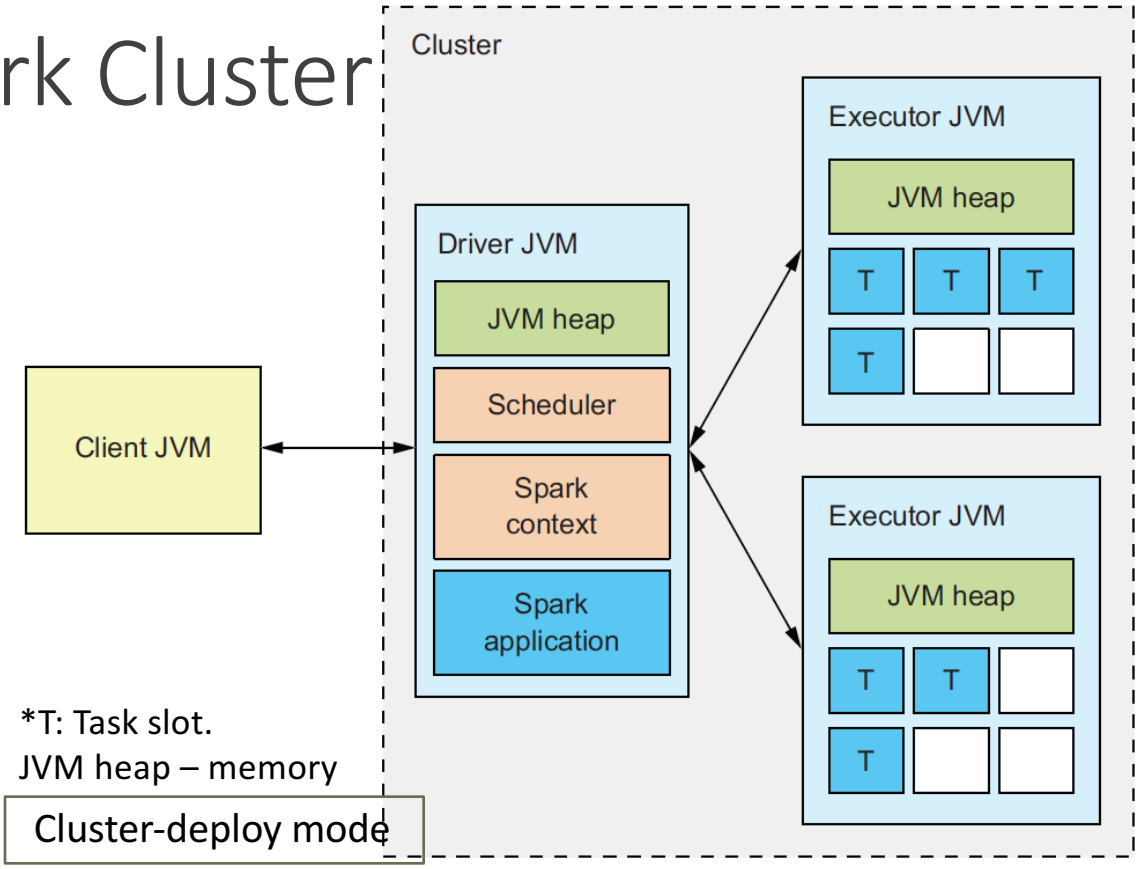
*The local mode and local cluster mode, although the easiest and quickest methods of setting up Spark, are used mainly for testing purposes.



Running on a Spark Cluster

Spark Runtime Architecture

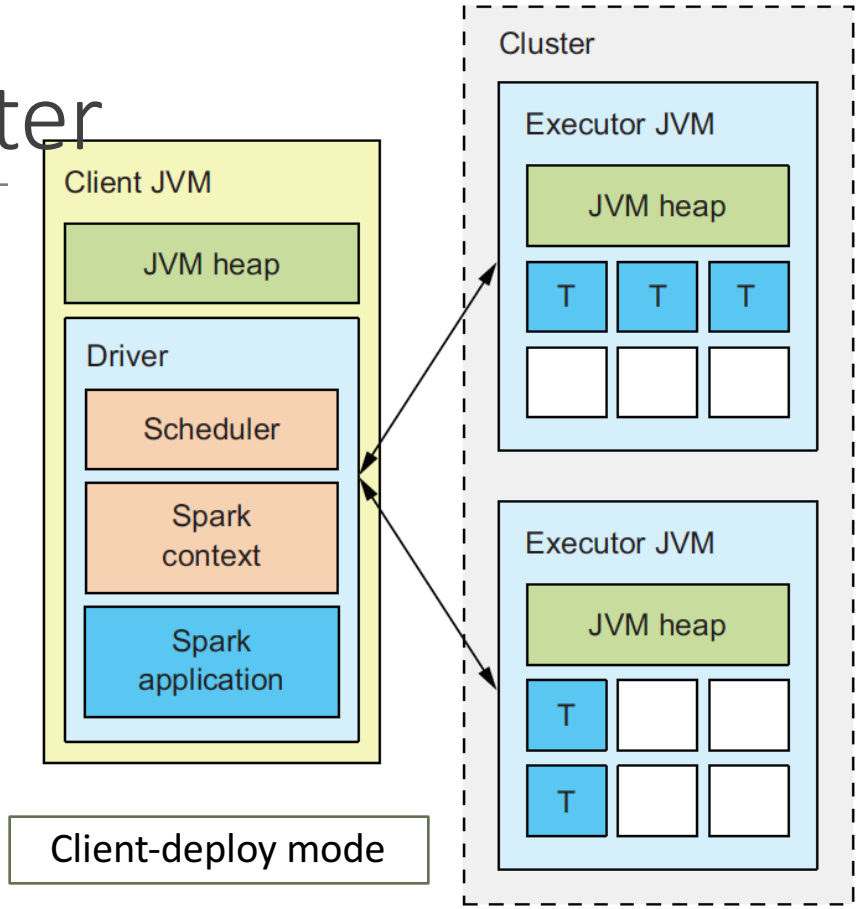
- Spark runtime components (3)
  - Client process
  - Driver
  - Executor
- Physical placement of executor and driver processes depends on the cluster types and its configuration.



Running on a Spark Cluster

Spark Runtime Architecture

- Spark runtime components (3)
  - Client process
  - Driver
  - Executor
- Physical placement of executor and driver processes depends on the cluster types and its configuration.



Running on a Spark Cluster

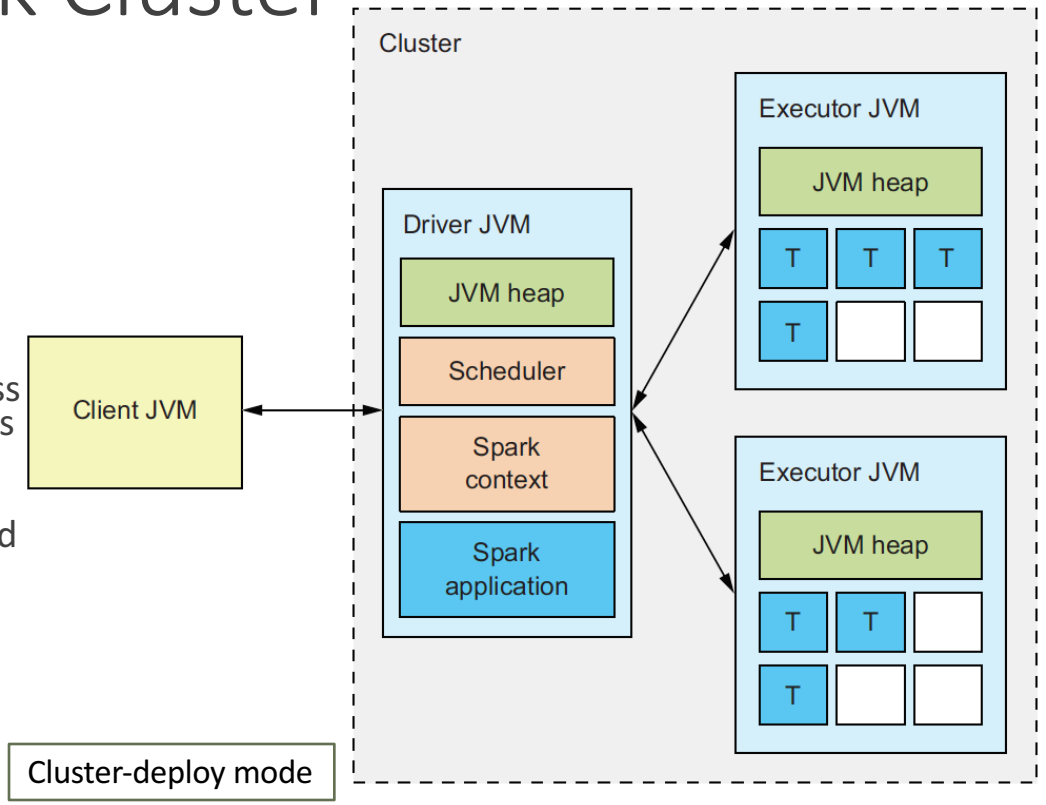
Spark Runtime Architecture

- Spark runtime components (3)
  - Client process
    - Start the driver program
    - Can be a spark-submit application, a spark-shell script, or a custom application.
    - Prepare the classpath and all configuration options for the Spark application.
  - Driver
  - Executor

Running on a Spark Cluster

Spark Runtime Architecture

- Spark runtime components (3)
  - Client process
  - Driver
    - Two ways to run – cluster-deploy mode and client-deploy mode.
      - Cluster-deploy mode – The driver process runs as a separate JVM process in a cluster and the cluster manages its resources.
      - Client-deploy mode – The driver is running in the client's JVM process and communicates with the executors managed by the cluster.
    - Always one driver per application.
  - Executor

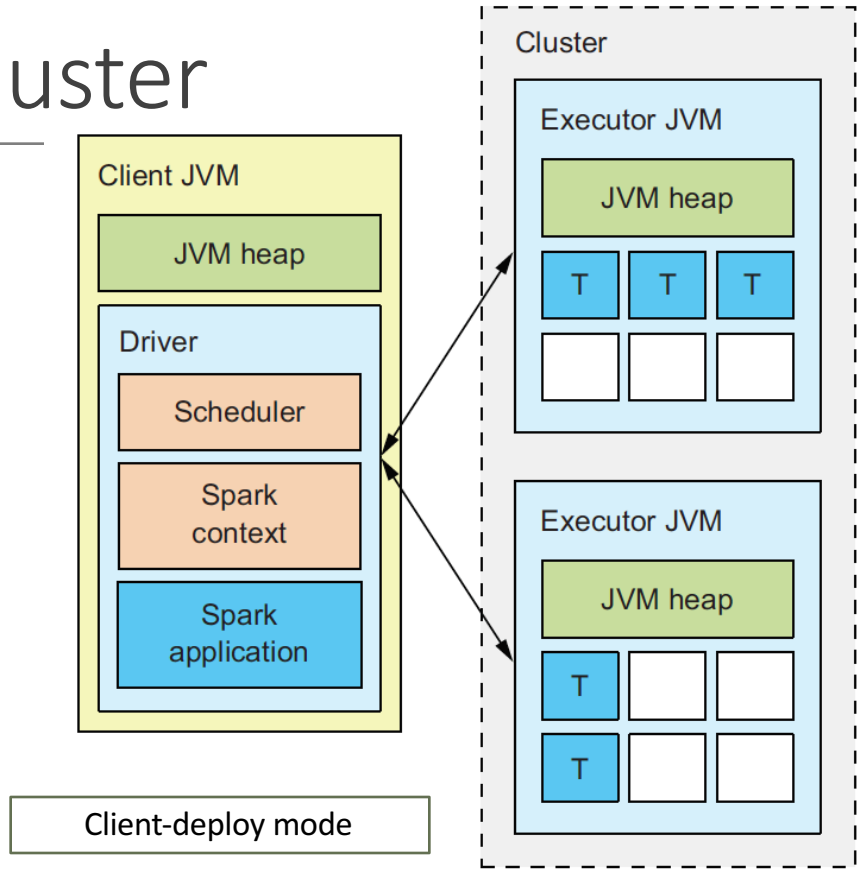


Cluster-deploy mode

Running on a Spark Cluster

Spark Runtime Architecture

- Spark runtime components (3)
  - Client process
  - Driver
    - Two ways to run – cluster-deploy mode and client-deploy mode.
      - Cluster-deploy mode – The driver process runs as a separate JVM process in a cluster and the cluster manages its resources.
      - Client-deploy mode – The driver is running in the client's JVM process and communicates with the executors managed by the cluster.
    - Always one driver per application.
  - Executor



Running on a Spark Cluster

Spark Runtime Architecture

- Spark runtime components (3)
  - Client process
  - Driver
  - Executor
    - Accept tasks from the driver, execute those tasks and return the results to the driver.
    - Each executor has several task slots for running tasks in parallel.
      - You can set the number of task slots to a value two or three times the number of CPU cores.

Running on a Spark Cluster

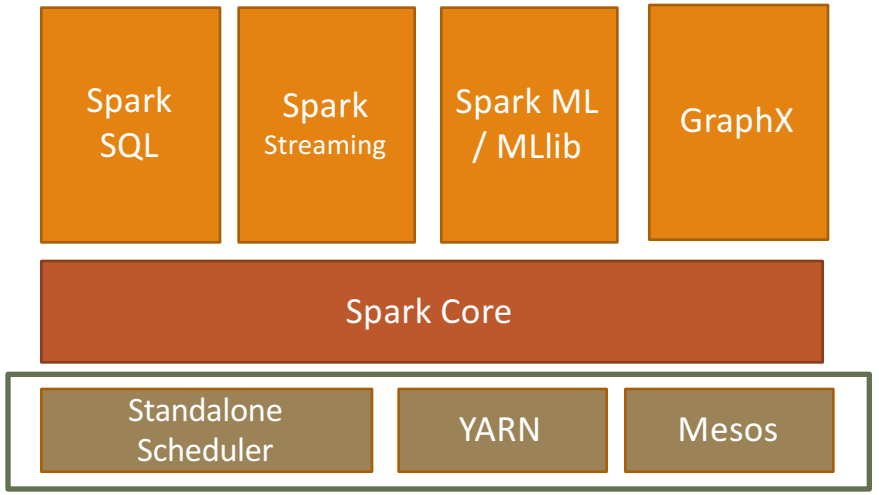
Spark Runtime Architecture

- Spark runtime components (3)
  - Client process
  - Driver
  - Executor
- Creation of the Spark context : Once the driver is started, it starts and configures an instance of SparkContext.
  - When running a Spark shell, the shell is the driver program and the spark context is already preconfigured and available as an sc variable. There can be only one Spark context per JVM.

Running on a Spark Cluster

Spark Runtime Architecture

- Spark Cluster Types (3)
  - Spark standalone cluster
  - YARN
  - Mesos



Running on a Spark Cluster

Spark Runtime Architecture

- Spark Cluster Types (3)
  - Spark standalone cluster
    - Spark-specific cluster.
    - Doesn't support communication with an HDFS secured with the Kerberos authentication protocol.
    - Known as Map Reduce 2 because it superseded the MapReduce engine in Hadoop 1 that only supported only MapReduce Jobs.
    - Provide faster job startup.

Running on a Spark Cluster

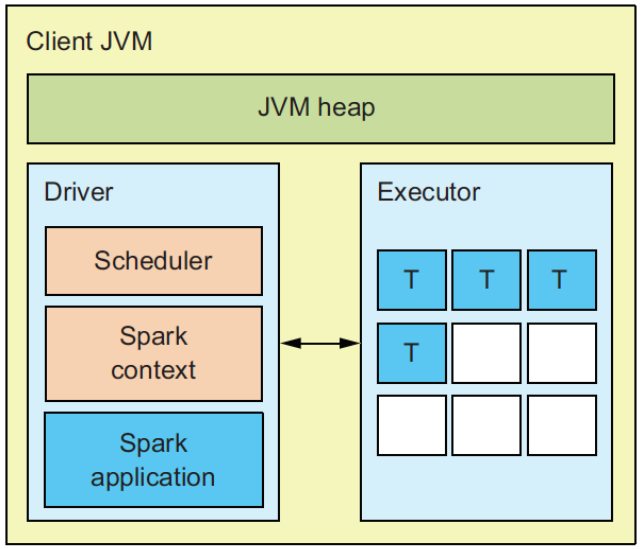
Spark Runtime Architecture

- Spark Cluster Types (3)
  - Spark standalone cluster
    - Spark local modes
      - Special cases of a Spark standalone cluster running on a single machine.
      - Easy to set up and use for quick tests. (Not for a production environment)
      - Two types
        - Local mode : A pseudo-cluster running on a single machine.
        - Local cluster mode : Spark standalone cluster that is also confined to a single machine.

Running on a Spark Cluster

Spark Runtime Architecture

- Spark Cluster Types (3)
  - Spark standalone cluster
    - Spark local modes
      - Two types
        - Local mode : A pseudo-cluster running on a single machine.
          - Only one executor in the same client JVM as the driver, but this executor can spawn several threads to run tasks.
            - Spark uses the client process as the single executor and the number of threads specified determines how many tasks can be executed in parallel.
          - Good for testing.
        - Local cluster mode : Spark standalone cluster that is also confined to a single machine.



Spark running in local mode.

Running on a Spark Cluster

Spark Runtime Architecture

- Spark Cluster Types (3)
  - Spark standalone cluster
    - Spark local modes
      - Two types
        - Local mode : A pseudo-cluster running on a single machine.
        - Local cluster mode : Spark standalone cluster that is also confined to a single machine.
          - It is a mode that a full Spark standalone cluster running on the local machine.
          - The difference with local mode is that the master process is a separate process but runs in the client JVM.
          - Good for Spark internal tests requiring inter-process communication.

Running on a Spark Cluster

Spark Runtime Architecture

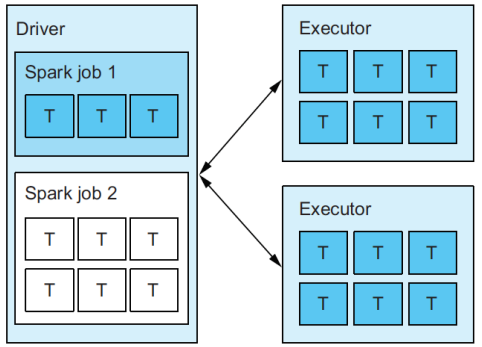
- Spark Cluster Types (3)
  - YARN
    - Hadoop's resource manger and execution system.
    - Pros
      - Many organizations already have YARN clusters of a significant size, along with the technical know-how, tools, and procedures for managing and monitoring them.
      - YARN lets you run different types of Java applications, not just Spark, so you can mix legacy Hadoop and Spark applications with ease.
      - YARN provides methods for isolating and prioritizing applications among users and organizations, functionality the standalone cluster doesn't have.
      - It's the only cluster type that supports Kerberos-secured HDFS (Secure HDFS mode).

Running on a Spark Cluster

Spark Runtime Architecture

- Spark Cluster Types (3)
  - Mesos
    - A scalable and fault-tolerant distributed systems kernel written in C++.
    - Mesos supports C++ and Python applications.
    - Mesos provides scheduling of memory and other types of resources including CPU , disk space, and ports.
    - You can run YARN on top of Mesos

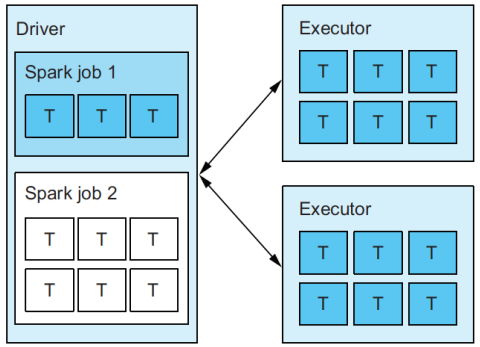
Running on a Spark Cluster



Resource and Job Scheduling

- Resources are scheduled as executors (JVM processes) and CPU (task slots) and then memory is allocated to them.
- Resource scheduling
  - If several Spark applications run in a single cluster, they compete for the cluster's resources.
  - Two levels of resource scheduling (Details in the next slides…).
    - Cluster resource scheduling
      - Allocate resources for Spark executors of different Spark applications.
      - Divide cluster resources among several applications running in a single cluster.
      - Provide requested resource for each application and free up the resources when the application closes.
    - Spark resource scheduling – schedule CPU and memory resources within a single application.
- Job scheduling
  - Once the application's driver and executors are running, the Spark scheduler communicates with them directly and decides which executors will run which tasks.

Running on a Spark Cluster



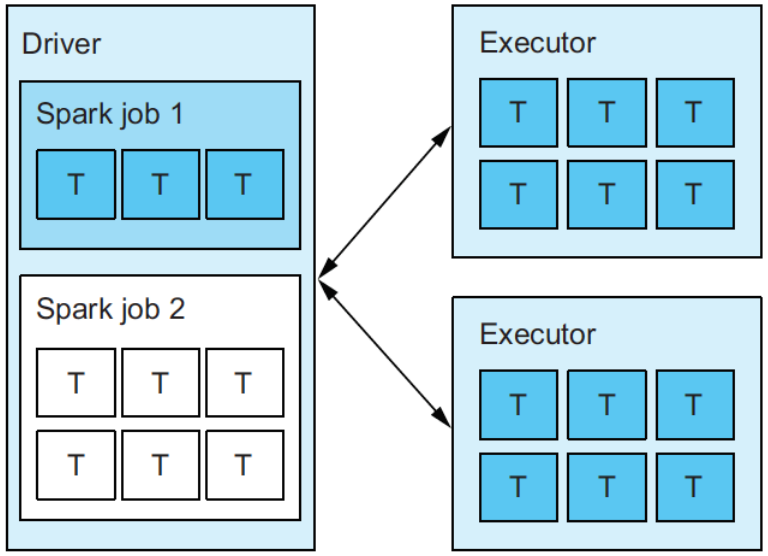
Resource and Job Scheduling

- Resources are scheduled as executors (JVM processes) and CPU (task slots) and then memory is allocated to them.
- Resource scheduling
- Job scheduling
  - Once the cluster manager allocates CPU and memory resource for the executors, scheduling of jobs occurs within the Spark application.
  - Decide how to split jobs into tasks and how to choose which executors will execute them.
  - Two scheduling modes.
    - FIFO scheduling
    - Fair scheduling

Running on a Spark Cluster

Resource and Job Scheduling

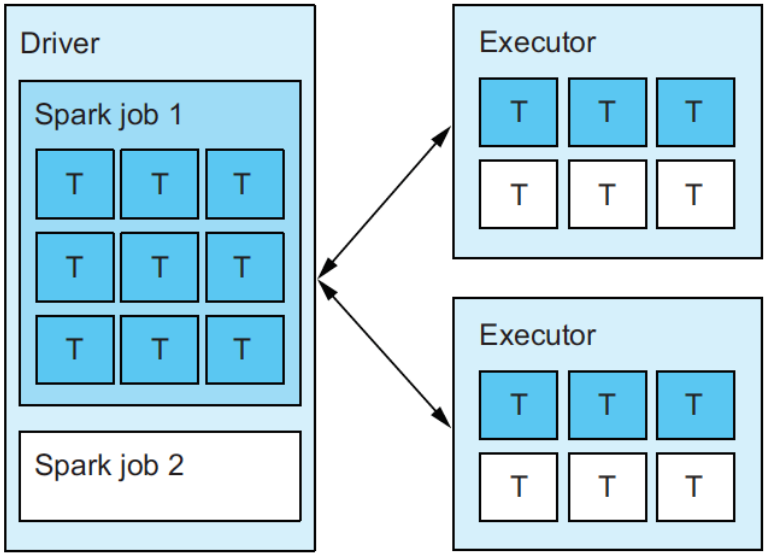
- Resources are scheduled as executors (JVM processes) and CPU (task slots) and then memory is allocated to them.
- Resource scheduling
- Job scheduling
  - Two scheduling modes.
    - FIFO scheduling : The first job that requests resource takes up all the necessary and available executor task slots.
      - Works best if your application is a single-user application that is running only one job at a time.
    - Fair scheduling



Running on a Spark Cluster

Resource and Job Scheduling

- Resources are scheduled as executors (JVM processes) and CPU (task slots) and then memory is allocated to them.
- Resource scheduling
- Job scheduling
  - Two scheduling modes.
    - FIFO scheduling
    - Fair scheduling: evenly distribute available resources.
      - Better option for multiuser application running several jobs simultaneously.

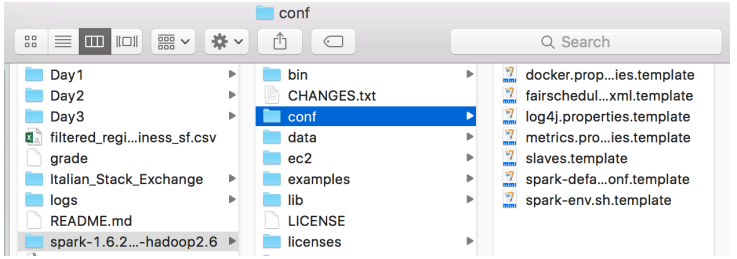


Running on a Spark Cluster

Resource and Job Scheduling

- Data locality : Spark tries to run tasks as close to the data location as possible.
  - Spark maintains a list of preferred hostnames or executors where the partition's data resides for each partition so that computation can be moved closer to the data.
  - If Spark obtains a list of preferred locations, the Spark scheduler tries to run tasks on the executors where the data is physically present.
  - 5 levels of locality
    1. PROCESS_LOCAL—Execute a task on the executor that cached the partition.
    2. NODE_LOCAL—Execute a task on the node where the partition is available.
    3. RACK_LOCAL—Execute the task on the same rack as the partition if rack information is available in the cluster (currently only on YARN).
    4. NO_PREF—No preferred locations are associated with the task.
    5. ANY—Default if everything else fails.
  - If a task slot with the best locality for the task can’t be obtained, the scheduler waits a certain amount of time and then tries a location with the second-best locality.

Running on a Spark Cluster



Spark Configuration

- List of configuration parameters. : http://spark.apache.org/docs/latest/configuration.html
  - Application properties, runtime environment, shuffle behavior, Spark UI, compression and serialization, memory management, execution behavior, networking, scheduling, etc.
- You can configure parameters using on the command line, in Spark configuration files, as system environment variables, and from within user program.
  - Spark configuration file – specify parameters in the <spark_home>/conf/spark-defaults.conf
  - Command line parameters – use as arguments to the pyspark, spark-shell and spark-submit commands.

```
ML-ITS-603436:~ dwoodbridge$ pyspark --help
Usage: ./bin/pyspark [options]

Options:
  --master MASTER_URL         spark://host:port, mesos://host:port, yarn, or local.
  --deploy-mode DEPLOY_MODE   Whether to launch the driver program locally ("client") or
                              on one of the worker machines inside the cluster ("cluster")
                              (Default: client).
  --class CLASS_NAME          Your application's main class (for Java / Scala apps).
  --name NAME                 A name of your application.
  --jars JARS                 Comma-separated list of local jars to include on the driver
                              and executor classpaths.
  --packages                  Comma-separated list of maven coordinates of jars to include
                              on the driver and executor classpaths. Will search the local
                              maven repo, then maven central and any additional remote
                              repositories given by --repositories. The format for the
                              coordinates should be groupId:artifactId:version.
  --exclude-packages          Comma-separated list of groupId:artifactId, to exclude while
                              resolving the dependencies provided in --packages to avoid
                              dependency conflicts.
  --repositories              Comma-separated list of additional remote repositories to
                              search for the maven coordinates given with --packages.
  --py-files PY_FILES         Comma-separated list of .zip, .egg, or .py files to place
                              on the PYTHONPATH for Python apps.
  --files FILES               Comma-separated list of files to be placed in the working
                              directory of each executor.

  --conf PROP=VALUE           Arbitrary Spark configuration property.
  --properties-file FILE      Path to a file from which to load extra properties. If not
                              specified, this will look for conf/spark-defaults.conf.
```

Running on a Spark Cluster

Spark Configuration

- List of configuration parameters. : http://spark.apache.org/docs/latest/configuration.html
  - Application properties, runtime environment, shuffle behavior, Spark UI, compression and serialization, memory management, execution behavior, networking, scheduling, etc.
- You can configure parameters using on the command line, in Spark configuration files, as system environment variables, and from within user program.
  1. Spark configuration file – specify parameters in the <spark_home>/conf/spark-defaults.conf
  2. Command line parameters – use as arguments to the pyspark, spark-shell and spark-

```
ML-ITS-603436:~ dwoodbridge$ pyspark --driver-memory 16g
[W 08:18:38.779 NotebookApp] Unrecognized JSON config file version, assuming version 1
[I 08:18:43.431 NotebookApp] [nb_conda_kernels] enabled, 1 kernels found
[I 08:18:43.814 NotebookApp] The port 8888 is already in use, trying another port.
[I 08:18:44.791 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 08:18:44.792 NotebookApp] ✗ nbpresent PDF export DISABLED: No module named nbbrowserpdf.exporters.pdf
[I 08:18:44.805 NotebookApp] [nb_conda] enabled
[I 08:18:45.094 NotebookApp] [nb_anacondacloud] enabled
[I 08:18:45.103 NotebookApp] Serving notebooks from local directory: /Users/dwoodbridge
[I 08:18:45.104 NotebookApp] 0 active kernels
[I 08:18:45.104 NotebookApp] The Jupyter Notebook is running at: http://localhost:8889/
[I 08:18:45.104 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

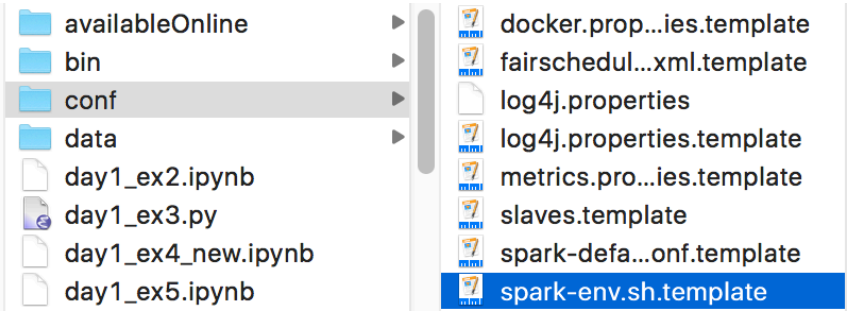
Running on a Spark Cluster

Spark Configuration

- List of configuration parameters. : http://spark.apache.org/docs/latest/configuration.html
  - Application properties, runtime environment, shuffle behavior, Spark UI, compression and serialization, memory management, execution behavior, networking, scheduling, etc.
- You can configure parameters using on the command line, in Spark configuration files, as system environment variables, and from within user program.
  - Spark configuration file – specify parameters in the <spark_home>/conf/spark-defaults.conf
  - Command line parameters – use as arguments to the pyspark, spark-shell and spark-submit commands.

```
ML-ITS-603436:~ dwoodbridge$ pyspark --conf spark.driver.memory=16g
[W 08:21:37.472 NotebookApp] Unrecognized JSON config file version, assuming version 1
[I 08:21:38.138 NotebookApp] [nb_conda_kernels] enabled, 1 kernels found
[I 08:21:38.193 NotebookApp] The port 8888 is already in use, trying another port.
[I 08:21:38.530 NotebookApp] ✓ nbpresent HTML export ENABLED
[W 08:21:38.530 NotebookApp] ✗ nbpresent PDF export DISABLED: No module named nbbrowserpdf.exporters.pdf
[I 08:21:38.534 NotebookApp] [nb_conda] enabled
[I 08:21:38.599 NotebookApp] [nb_anacondacloud] enabled
[I 08:21:38.607 NotebookApp] Serving notebooks from local directory: /Users/dwoodbridge
```

Running on a Spark Clus



Spark Configuration

- List of configuration parameters. : http://spark.apache.org/docs/latest/configuration.html
  - Application properties, runtime environment, shuffle behavior, Spark UI, compression and serialization, memory management, execution behavior, networking, scheduling, etc.
- You can configure parameters using on the command line, in Spark configuration files, as system environment variables, and from within user program.
  1. Spark configuration file – specify parameters in the <spark_home>/conf/spark-defaults.conf
  2. Command line parameters – use as arguments to the pyspark, spark-shell and spark-submit commands.
  3. System environment variables – Specifying the spark-env.sh file in the <spark_home>/conf.
  4. User program – You can set Spark configuration parameters directly in your program by using the SparkConf class. Spark configuration cannot be changed at runtime, so make sure set up the SparkConf object with all the configuration options before creating the SparkContext object.

Running on a Spark Cluster – Example 4

Viewing configured parameters

```
sc.getConf()
```

```
<pyspark.conf.SparkConf at 0x10b9f1890>
```

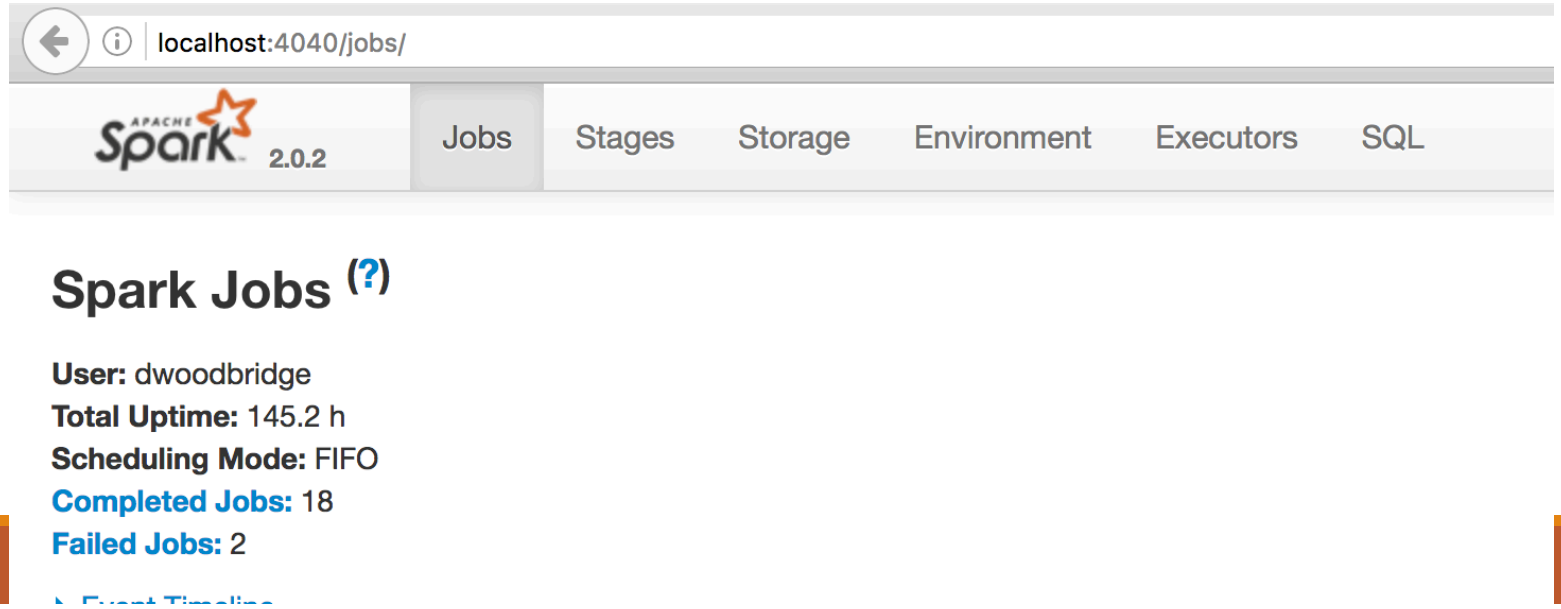
```
sc.getConf().getAll()
```

```
[(u'spark.driver.host', u'10.91.221.200'),
 (u'spark.executor.extraClassPath',
  u':/usr/local/Cellar/hadoop/2.7.3/libexec/etc/aws-java-sdk-1.11.72.jar:/usr/local/Cellar/hadoop/2.7.3/libexec/etc/hadoop-aws-2.6.0.jar:/usr/local/Cellar/hadoop/2.7.3/libexec/etc/aws-java-sdk-1.11.72.jar:/usr/local/Cellar/hadoop/2.7.3/libexec/etc/hadoop-aws-2.6.0.jar'),
 (u'spark.app.id', u'local-1483029568412'),
 (u'spark.executor.id', u'driver'),
 (u'spark.driver.memory', u'16g'),
 (u'spark.app.name', u'PySparkShell'),
 (u'spark.sql.catalogImplementation', u'hive'),
 (u'spark.rdd.compress', u'True'),
 (u'spark.serializer.objectStreamReset', u'100'),
 (u'spark.driver.extraClassPath',
  u':/usr/local/Cellar/hadoop/2.7.3/libexec/etc/aws-java-sdk-1.11.72.jar:/usr/local/Cellar/hadoop/2.7.3/libexec/etc/hadoop-aws-2.6.0.jar:/usr/local/Cellar/hadoop/2.7.3/libexec/etc/aws-java-sdk-1.11.72.jar:/usr/local/Cellar/hadoop/2.7.3/libexec/etc/hadoop-aws-2.6.0.jar'),
 (u'spark.master', u'local[*]'),
 (u'hive.metastore.warehouse.dir',
  u'file:/Users/dwoodbridge/spark-2.0.0-bin-hadoop2.7/spark-warehouse'),
 (u'spark.submit.deployMode', u'client'),
 (u'spark.driver.port', u'58055')]
```

Running on a Spark Cluster

Spark Web UI

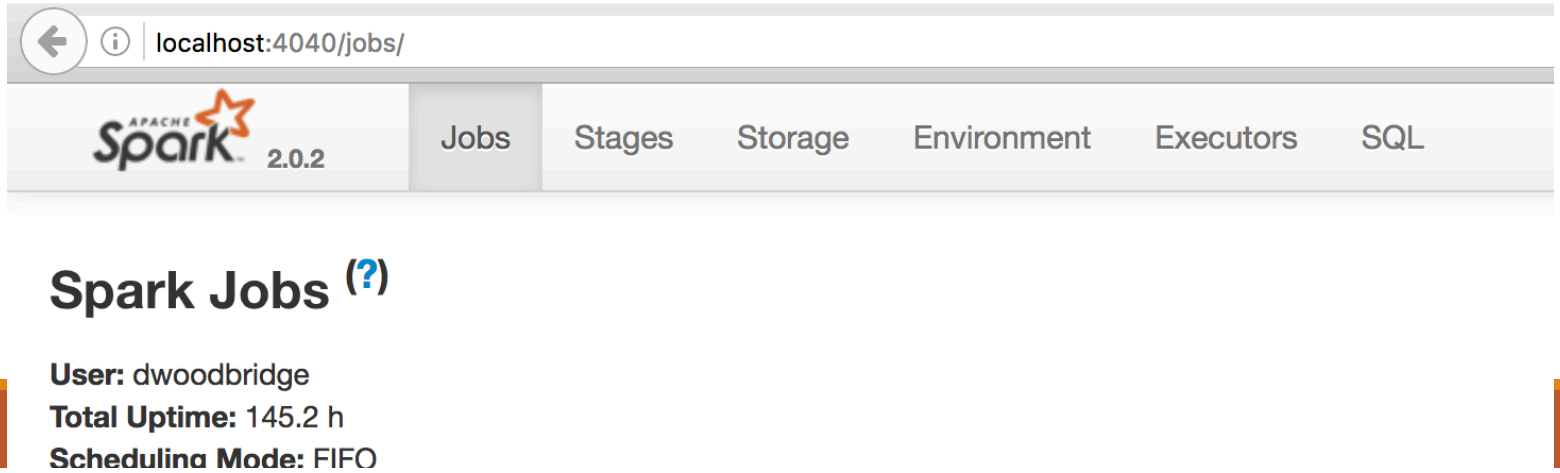
- Each time a SparkContext object is initialized, Spark starts a web UI providing Spark environment and job execution stats.
- Use http://localhost:4040.



Running on a Spark Cluster

Spark Web UI

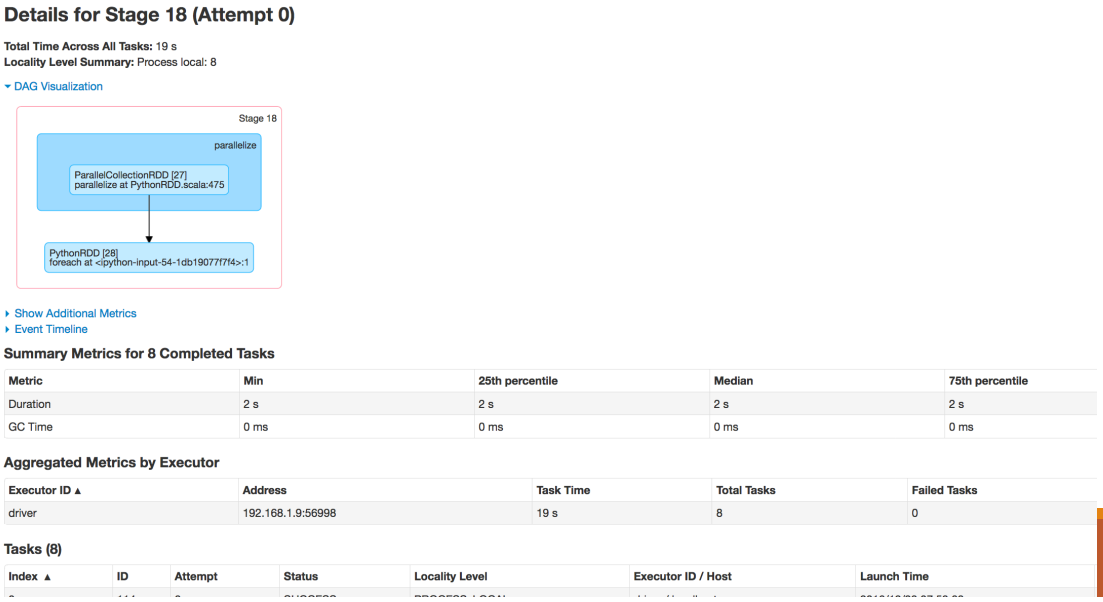
- Jobs
- Stages
- Storage
- Environment
- Executors



Running on a Spark Cluster

Spark Web UI

- Jobs
  - Provide statistics about running, completed and failed jobs.
  - If you click a job description, you can see its completed and failed stages.



Details for Stage 18 (Attempt 0)

Total Time Across All Tasks: 19 s
Locality Level Summary: Process local: 8

DAG Visualization

Show Additional Metrics
Event Timeline

Summary Metrics for 8 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile
Duration	2 s	2 s	2 s	2 s
GC Time	0 ms	0 ms	0 ms	0 ms

Aggregated Metrics by Executor

Executor ID ▲	Address	Task Time	Total Tasks	Failed Tasks
driver	192.168.1.9:56998	19 s	8	0

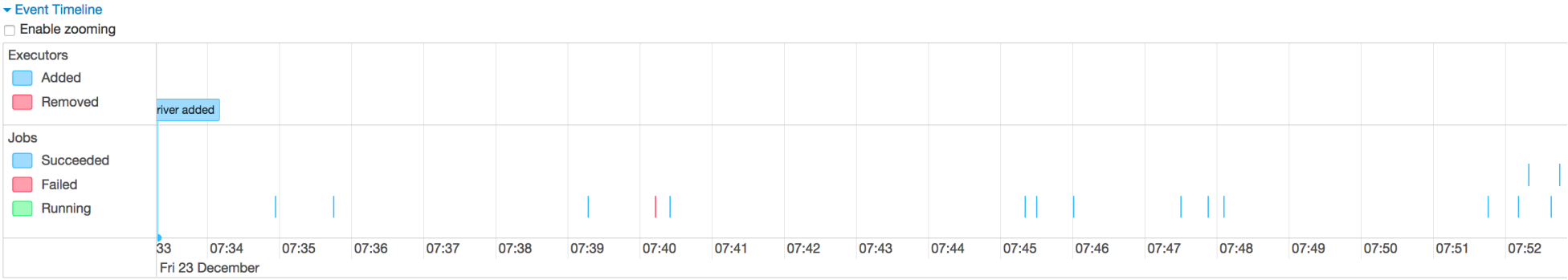
Tasks (8)

Index ▲	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time

Running on a Spark Cluster

Spark Web UI

- Jobs
  - Provide statistics about running, completed and failed jobs.
  - If you click a job description, you can see its completed and failed stages.
  - If you click the event timeline link, you get a graphical representation of jobs as they executed over time.



Running on a Spark Cluster

Spark Web UI

- Jobs
- Stages
  - Show when each stage started, how long it ran, whether it is still running, size of input and output and shuffle reads and writes, etc.
  - **Can be used to drill down to the problematic stages and tasks and narrow the problem.**
- Storage
  - Provide information about your cached RDDs and memory, storage, etc. that the data is consuming.
- Environment
  - Provide Java and Scala version, Spark and system properties, etc.
- Executors
  - Provide all executors in your cluster and its available and used memory (default is 54% of heap).

Running on a Spark Standalone Cluster

Spark standalone cluster components

Starting a Spark standalone cluster

Running applications in a standalone cluster

Running on AWS EC2

Running on a Spark Standalone Cluster

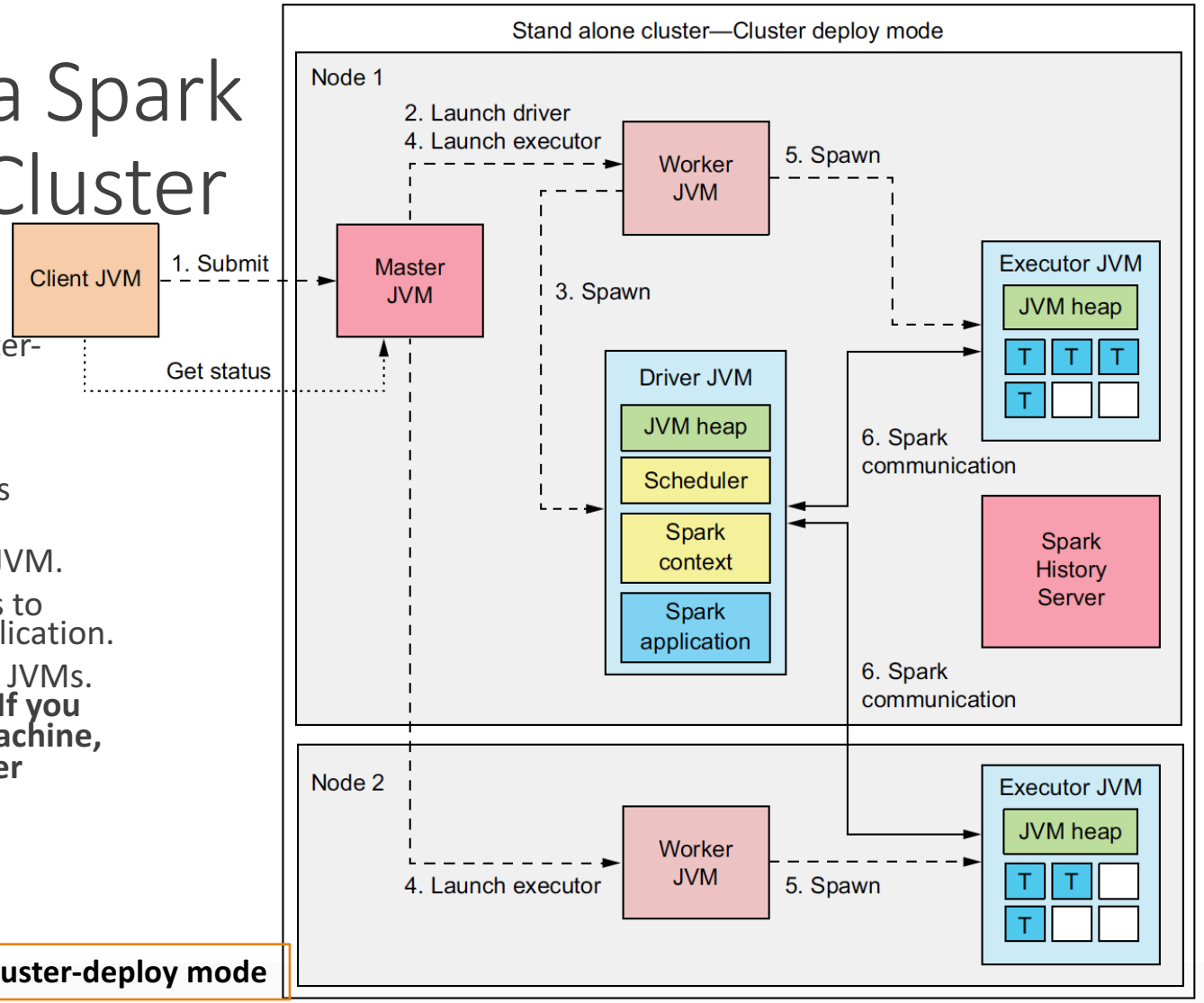
Spark standalone cluster components

- Spark standalone cluster : easy to install and configure as it was built and optimized for Spark.
- Additional Components
  - Master process
    - Act as the cluster manager.
    - Accept applications to be run and schedule worker resources.
  - Worker (slave) process
    - Launch application executors and driver for application execution.
    - Normally need to distribute workers across several nodes to avoid reaching the limits of a single machine's resources.

Running on a Spark Standalone Cluster

Spark standalone cluster

- Example of an application in cluster-deploy mode
  1. Client process submits an application to the master.
  2. The master instruct one of its workers to launch a driver.
  3. The worker spawns a driver JVM.
  4. The master instructs workers to launch executors for the application.
  5. The workers spawn executor JVMs. **A worker has one executor. If you need more executors per machine, you can start multiple worker processes.**
  6. The driver and executors communicate.



Cluster-deploy mode

Running on a Spark Standalone Cluster

Starting the standalone cluster

- Unlike previous exercises which use local mode, you should start a Spark standalone cluster before submitting an application or prior to starting the Spark shell.
- When the cluster is running, connect your application to the cluster.
- Using shell scripts to start the standalone cluster.
  - The shell scripts are in the <spark_home>/sbin directory.
    - start-master.sh : starts the master process.
    - start-slaves.sh : starts all defined worker processes.
    - start-all.sh : starts both master and worker processes.

Running on a Spark Standalone Cluster

Starting the standalone cluster

- Using shell scripts to start the standalone clsuter.
  - The shell scripts are in the <spark_home>/sbin directory.
    - start-master.sh : starts the master process.
      - You can customize system environment parameter including master IP, port, webUI port, etc. (See start-master.sh)

```
"${SPARK_HOME}/sbin"/spark-daemon.sh start $CLASS 1 \
  --host $SPARK_MASTER_HOST --port $SPARK_MASTER_PORT --webui-port $SPARK_MASTER_WEBUI_PORT \
  $ORIGINAL_ARGS
```

Running on a Spark Standalone Cluster

Starting the standalone cluster

- Using shell scripts to start the standalone clsuter.
  - The shell scripts are in the <spark_home>/sbin directory.
    - start-slaves.sh : starts all defined worker processes.
      - Connect to all machines defined in the <spark-home>/conf/slaves file and start worker processes.
        - The slaves file should contain a list of worker hostnames, each on a separate line.
      - Spark should be installed at the same location on all machines in the cluster.
    - start-all.sh : starts both master and worker processes.
      - Call start-master.sh and then start-slaves.sh.

Running on a Spark Standalone Cluster

Standalone master high availability and recovery

- Master process : Important because…
  - Client processes connect to it to submit applications.
  - Requests resources from the workers on behalf of the clients.
  - Users view the state of the running applications, relying on it.
- Master high availability : master process will be automatically restarted if it goes down.
  - spark.deploy.recoverMode (2 options)

1. ZOOKEEPER
   - You need to install and configure ZooKeeper and then start several master processes instructing them to synch through ZooKeeper. Only one will become a ZooKeeper leader. If the leader fails, one of the other masters will take its place and restore the master's state.
   - Best for the production-level high availability.
2. FILESYSTEM
   - Provide a single-node recovery, if you just want to restart the master process when it goes down.

http://spark.apache.org/docs/latest/spark-standalone.html#high-availability

Running on a Spark Standalone Cluster

Running applications in a standalone cluster

You can run it by

1. Submit with spark-submit, spark-shell, pyspark
2. Run in a Spark shell.
3. Instantiate and configure a SparkContext object in your own application.

You need to specify a master connection URL with the hostname and port of the master process.

Additional parameters to configure

- deploy-mode : client or cluster (Python application cannot run in cluster-deploy mode) on a standalone cluster.
- executor-cores, total-executor-cores.
- Additional classpath entries and files.
  - --jars, --py-files (change configuration in your code.).

Running on a Spark Standalone Cluster

Running on Amazon EC2.

- AWS EC2 : Amazon's cloud service that lets you rent virtual servers to run your applications.
- Download key pairs (.pem).
- Specify the credentials.
- Download spark-ec2 and launch/run the cluster.
  - spark-ec2 allows you to launch, manage and shut down Spark standalone clusters on Amazon EC2. It automatically sets up Apache Spark and HDFS on the cluster for you.
  - Download/clone spark-ec2 from https://github.com/amplab/spark-ec2 and place the unzipped folder under <spark_home> or list the folder under your .profile.
  - Also download spark*.tgz .
- Deploy and run code
  - To deploy code or data within your cluster, you can log in and use the provided script ~/spark-ec2/copy-dir, which, given a directory path, RSYNCs it to the same location on all the slaves.

```
ML-ITS-603436:spark-2.0.0-bin-hadoop2.7 dwoodbridge$ spark-ec2 --key-pair=msan694 --identity-file=msan694.pem --region=us-west-2 --zone=us-west-2a launch spark-cluster
Setting up security groups...
Creating security group spark-cluster-master
Creating security group spark-cluster-slaves
Searching for existing cluster spark-cluster in region us-west-2...
Spark AMI: ami-ae6e0d9e
Launching instances...
Launched 1 slave in us-west-2a, regid = r-0974105d76dd6a795
Launched master in us-west-2a, regid = r-0f077d1e870ffac33
```

https://github.com/amplab/spark-ec2

Running on YARN

YARN architecture

YARN resource scheduling

Running Spark on Amazon EMR.

Running on YARN

YARN

- Is a widely used Hadoop's MapReduce execution engine.
- Can run other types of jobs (such as Spark) other than MapReduce.

YARN Architecture

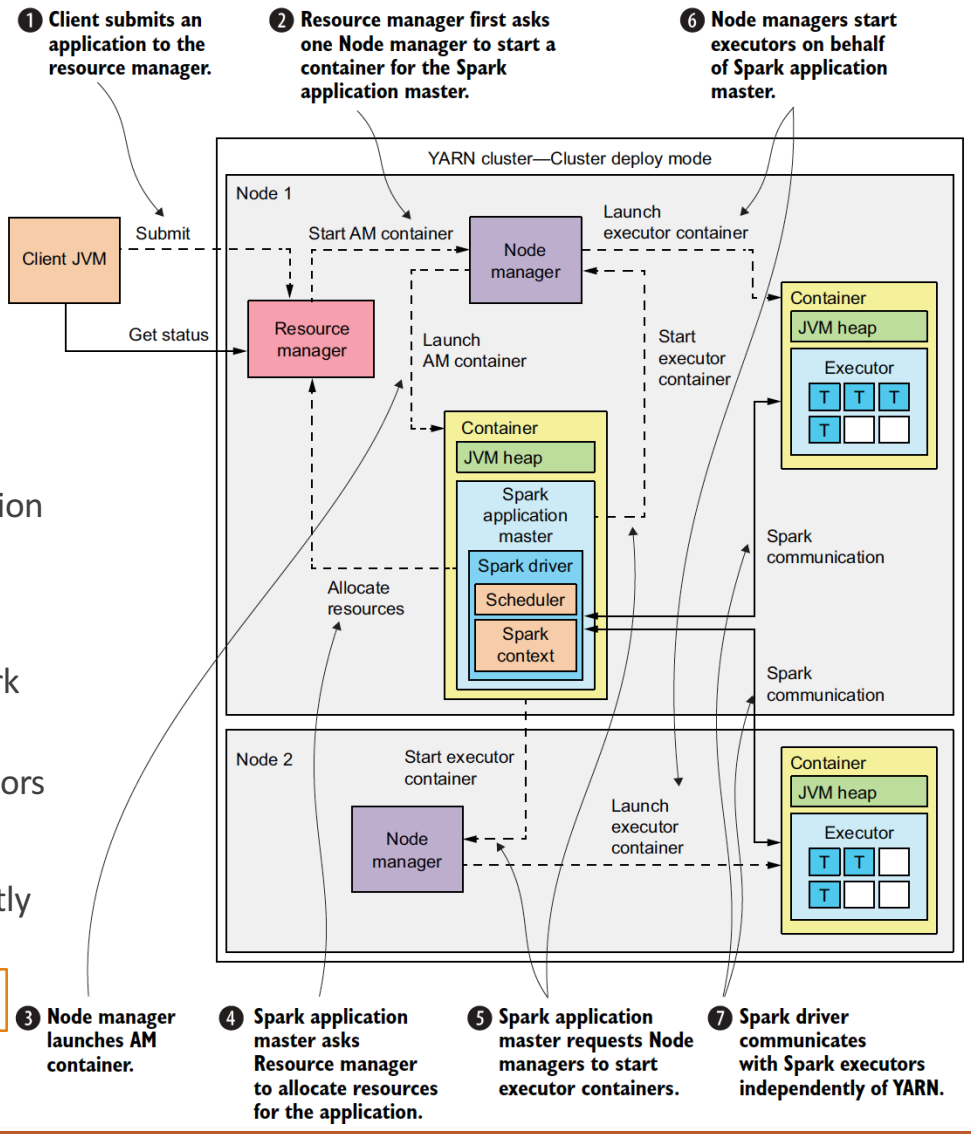
- Components
  - Resource manger (One)
  - Node manager (Several)
- Unlike running on Spark's standalone cluster, applications on YARN run in container
  - Container
  - Application master – run in its own container, it is responsible fore requesting resources from resource manager.

Running on YARN

YARN Architecture

- Example cluster of two nodes
  1. A client submit an application to the resource manager.
  2. The resource manger directs one of the node manager to allocate a container for the application master.
  3. The node manager launches a container for the application master (AM).
  4. Spark application master asks the resource manager for more containers to be used as Spark executors.
  5. When the resources are granted the application master asks the node manager to launch executors in the new container.
  6. The node manager obeys and starts executors.
  7. Driver and executors communicate independently of YARN components.

Cluster-deploy mode



Running on YARN

YARN Architecture

- YARN is installed together with Hadoop.
- Three possible modes
  - Standalone (local) mode – Runs as a single Java process.
  - Pseudo-distributed mode – Runs all Hadoop daemons (several Java processes) on a single machine.
  - Fully distributed mode – Runs on multiple machines.

Running on YARN

YARN resource scheduling (3)

- FIFO scheduler
  - The first application that requests resources will be first served.
- Capacity scheduler (Default)
  - Designed to allow for sharing a single YARN cluster by different organizations and it guarantees that each organization will always have a certain amount of resource available (guaranteed capacity). The resources scheduled by YARN is a queue.
  - Each queue's capacity determines the percentage of cluster resources that can be used by applications.
  - A hierarchy of queues can be set up to reflect a hierarchy of capacity requirements by organizations.
- Fair scheduler
  - Assign resources in a way that all applications get on average an equal share.
  - Also supports application priorities.

Running on YARN

YARN Security Consideration

- Hadoop provides the means for authorizing access to resources to certain users, using Kerberos-provided identity and access control lists in the Hadoop configuration.
- YARN knows how to handle Kerberos authentication information and pass it on to HDFS.

YARN Dynamic Resource Allocation

- Enable applications to release executors temporarily so that other applications can use the allocated resources instead of being idle.
- This is only for the YARN cluster manager.
- When you enable this, you should also enable Spark's shuffle service in order to prevent shuffle files to be recalculated which wastes resources.

Running on YARN

Running Spark on Amazon EMR.

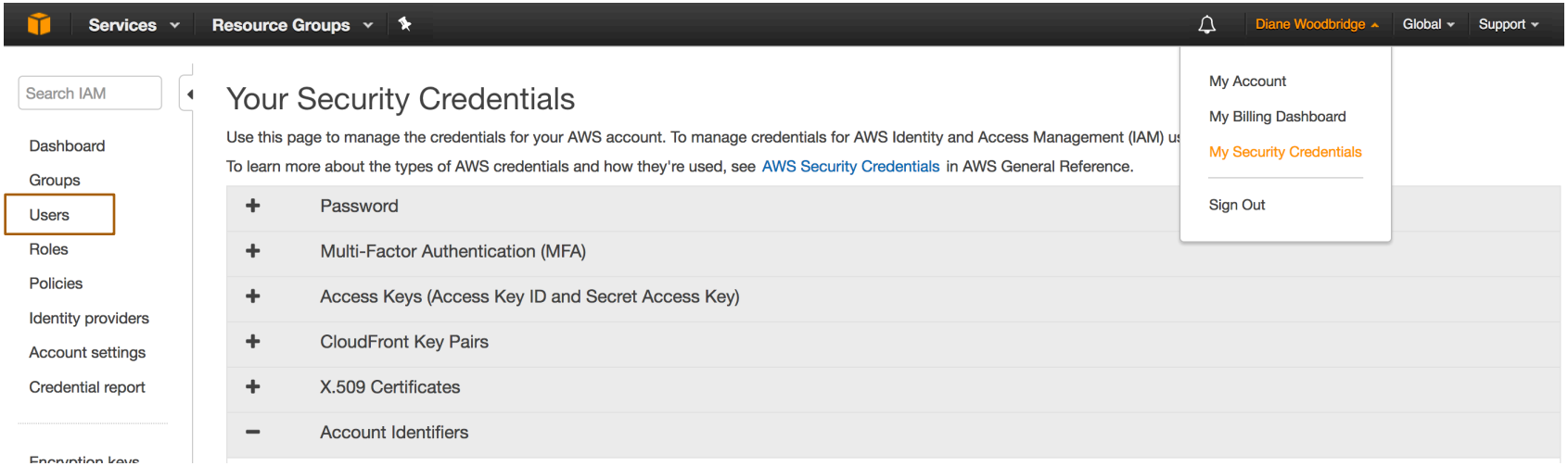
- Apache Spark on Hadoop YARN is natively supported in Amazon EMR, and you can quickly and easily create managed Apache Spark clusters from the AWS Management Console, AWS CLI, or the Amazon EMR API.
- You can leverage additional Amazon EMR features, including fast Amazon S3 connectivity using the Amazon EMR File System (EMRFS), integration with the Amazon EC2 Spot market, and Auto Scaling to add or remove instances from your cluster.
- Reference
  - http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-spark.html
  - http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-spark-shell.html

https://aws.amazon.com/emr/details/spark/

Running on YARN

Using Amazon EMR.

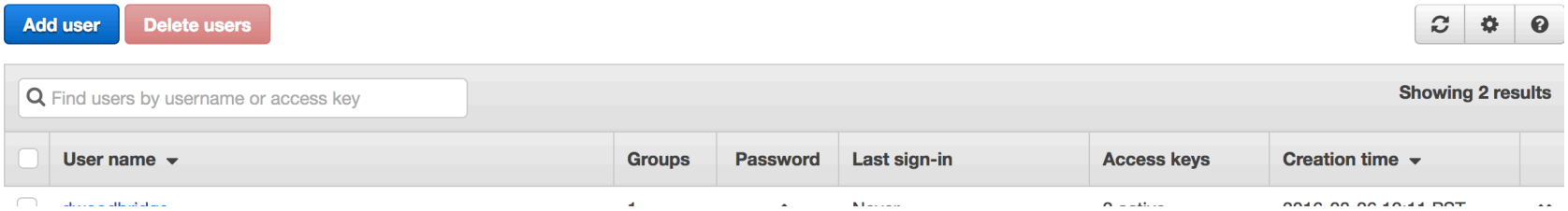
- On your AWS console, go to the user information ➔ My Security Credentials. ➔ Choose Users. ➔ Add User.



Running on YARN

Using Amazon EMR.

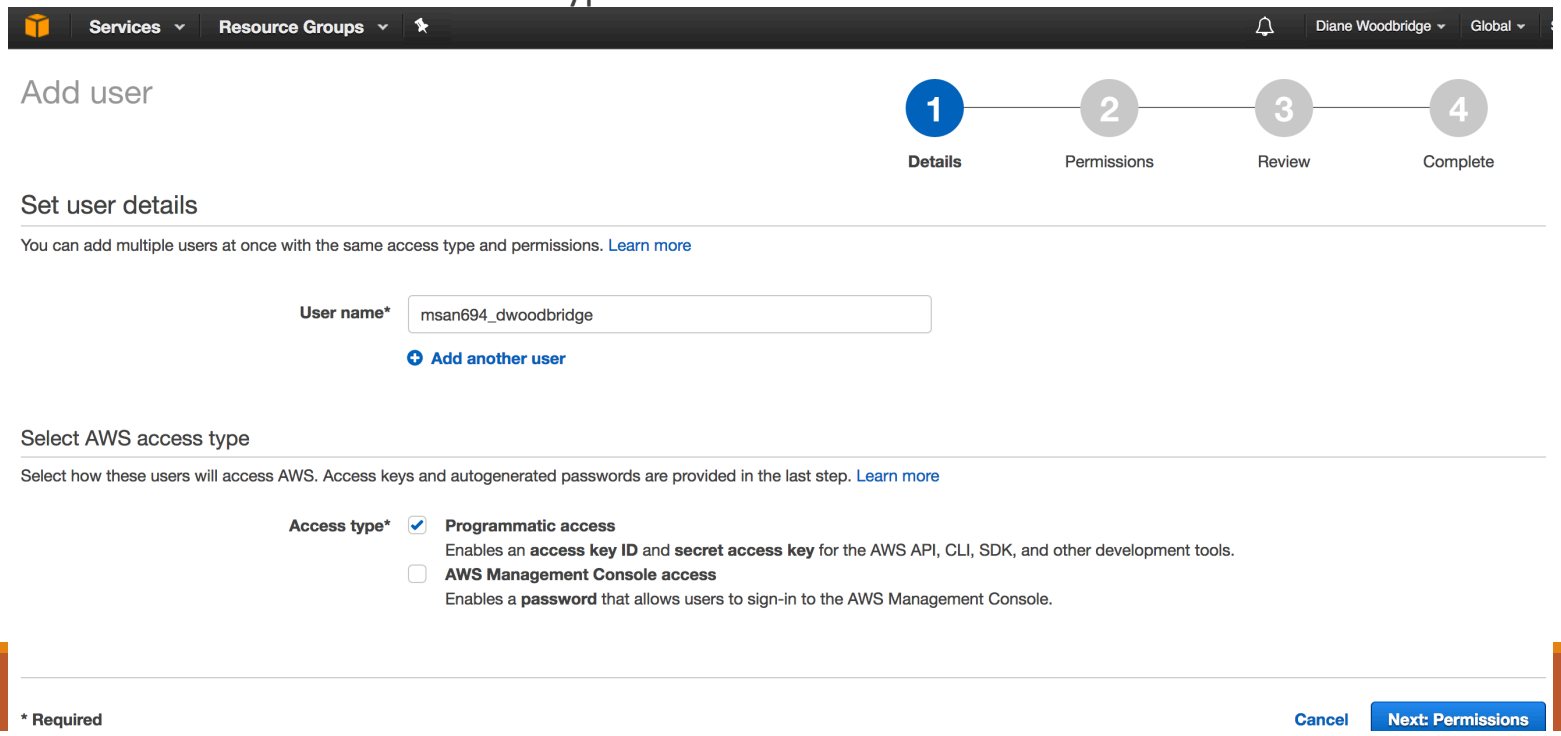
- Choose Add User.



Running on YARN

Using Amazon EMR.

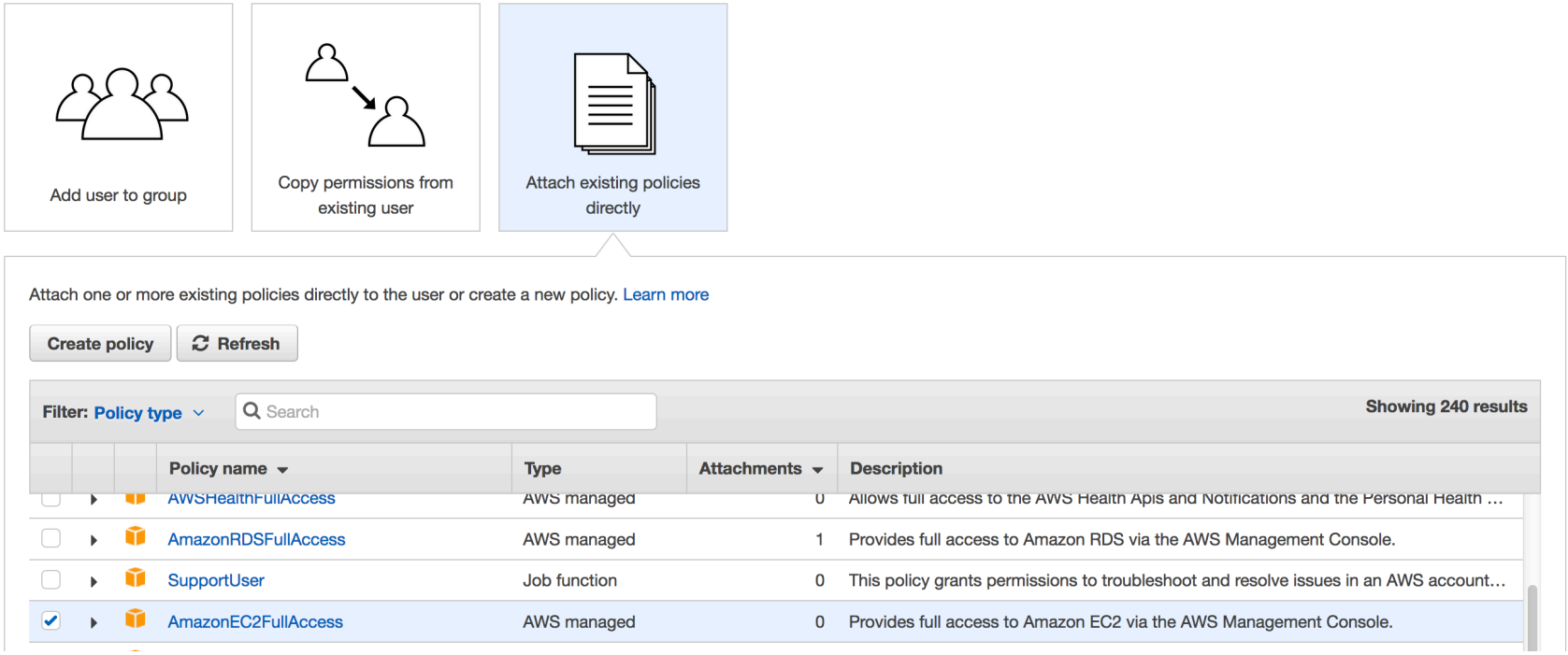
- Choose a user name and access type.



Running on YARN

Using Amazon EMR.

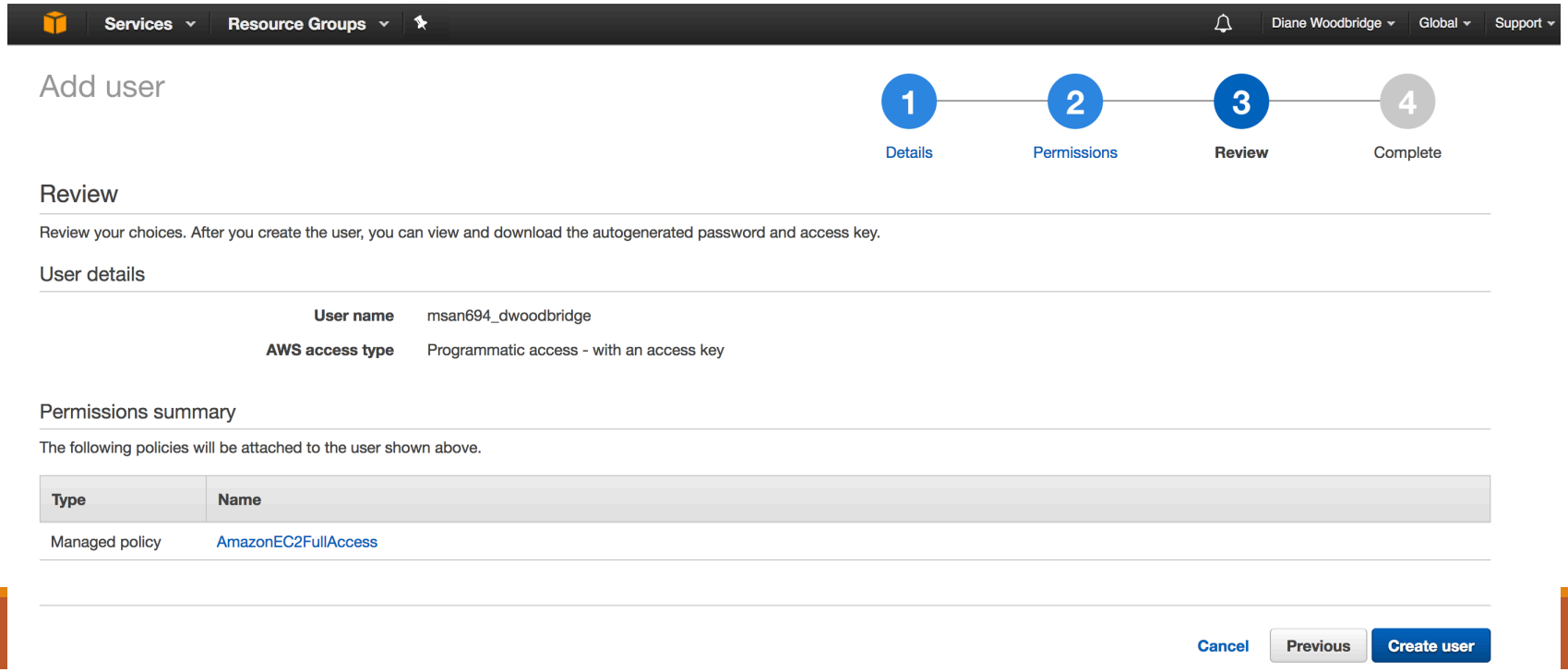
- Set permissions for the user. : Attach existing policies directly. ➔ Amazon EC2FullAccess.



Running on YARN

Using Amazon EMR.

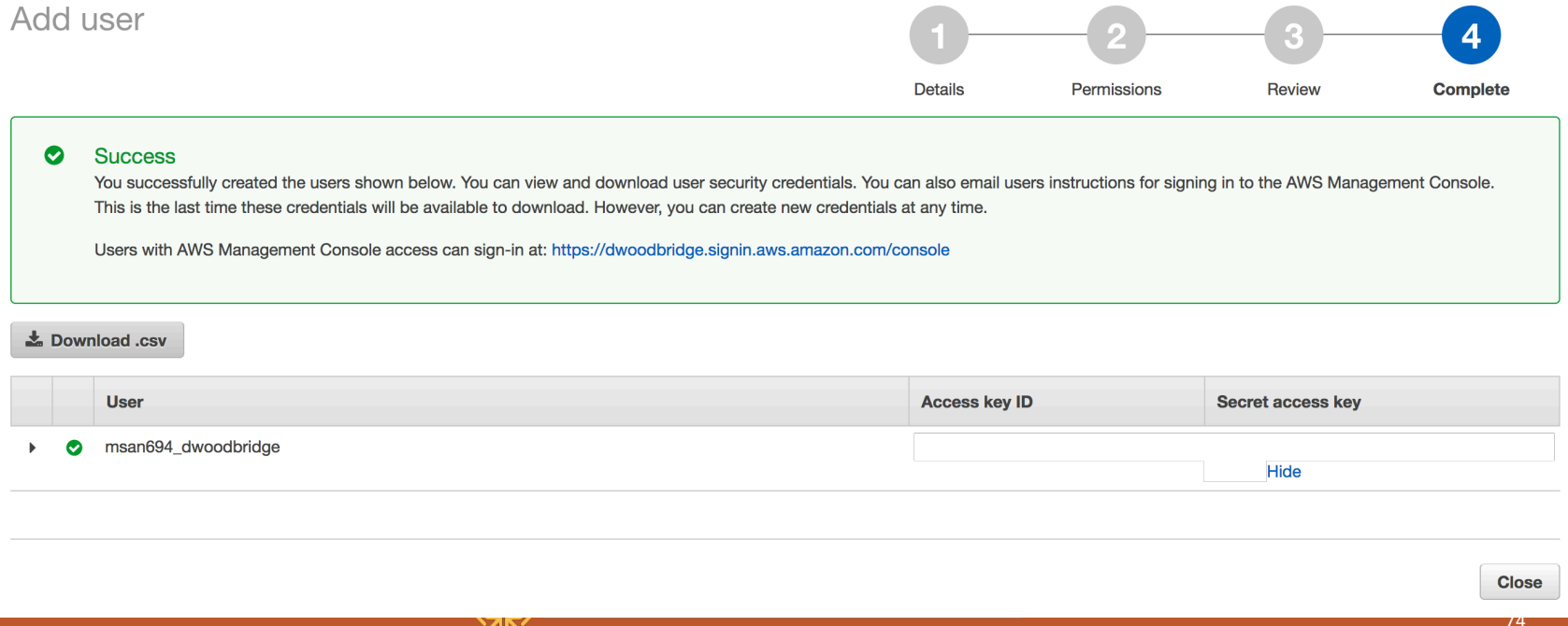
- Create User.



Running on YARN

Using Amazon EMR.

- Copy the Access Key ID and Secret Access Key (Or download .csv).



Running on YARN

Using Amazon EMR.

- On your .profile or .bash_profile store AWS_ACESS_KEY_IDA and AWS_SECRET_ACCESS_KEY.

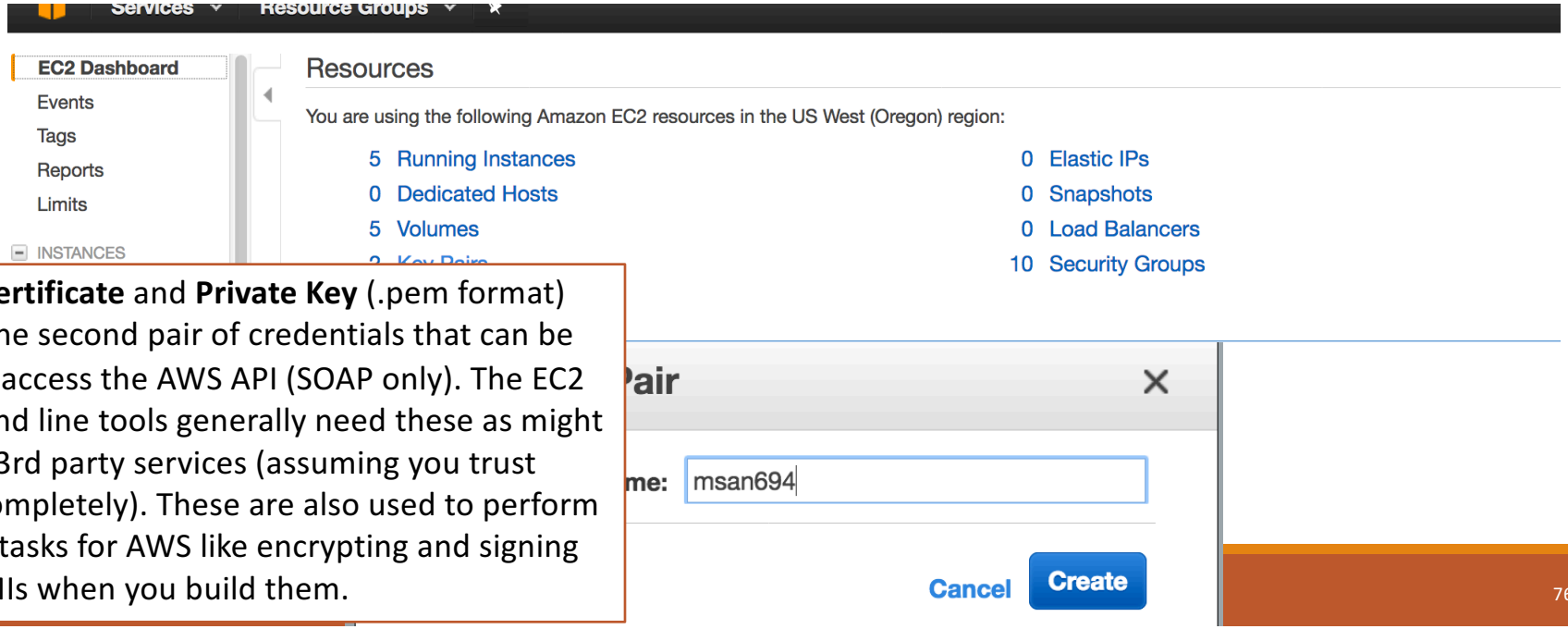
```
export AWS_ACCESS_KEY_ID=AKIAI
export AWS_SECRET_ACCESS_KEY=t
```

AWS **Access Key ID** and **Secret Access Key**.
This can be used to access and control basic AWS services through the API including EC2, S3, SimpleDB, CloudFront, SQS, EMR, RDS, etc.

Running on YARN

Using Amazon EMR.

- On EC2 Console (https://console.aws.amazon.com/ec2/), Choose Key Pairs ➔ Create Key Pairs , downloand the .pem and store somewhere on your machine*.

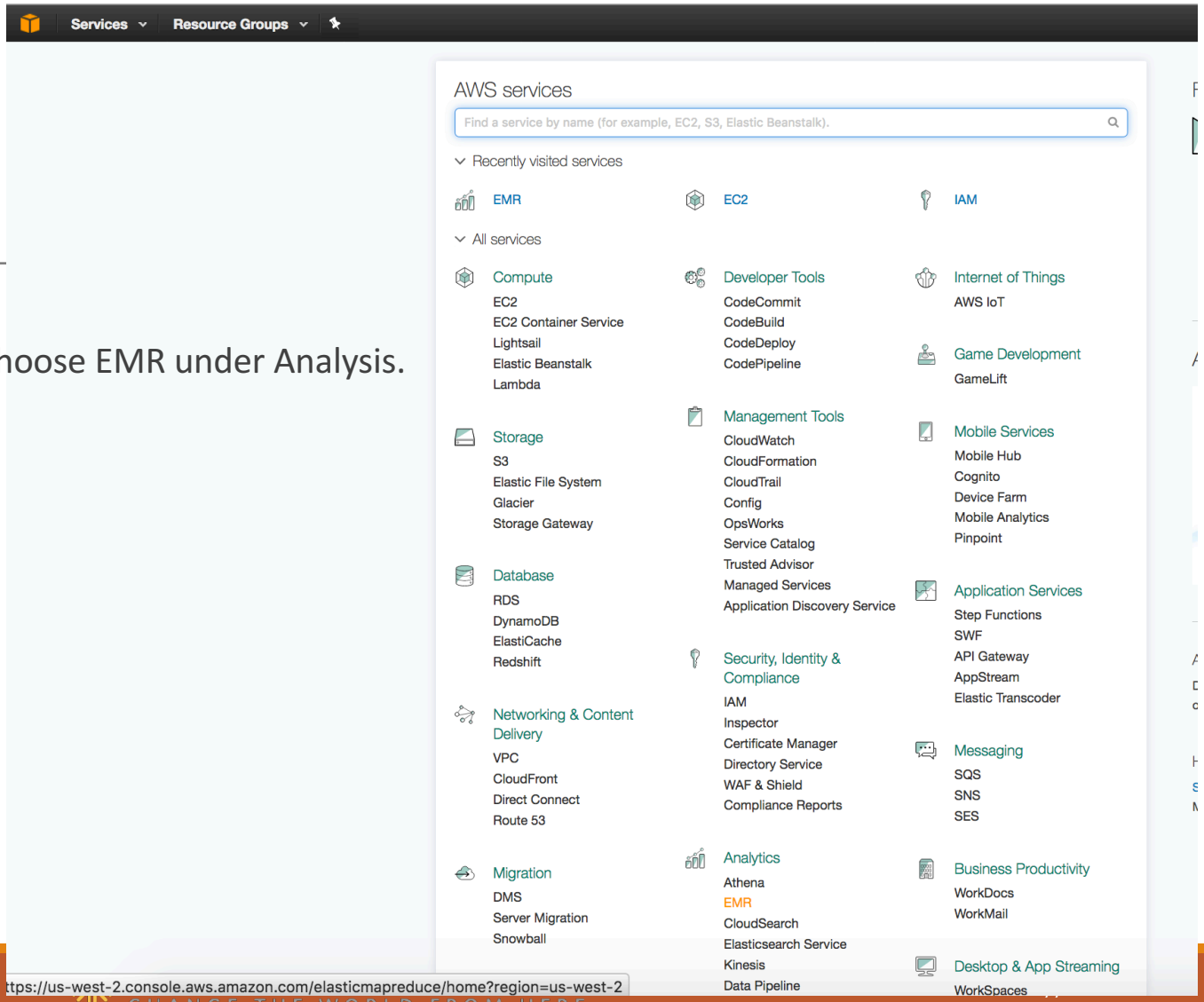


X.509 **Certificate** and **Private Key** (.pem format) This is the second pair of credentials that can be used to access the AWS API (SOAP only). The EC2 command line tools generally need these as might certain 3rd party services (assuming you trust them completely). These are also used to perform various tasks for AWS like encrypting and signing new AMIs when you build them.

Running on YARN

Using Amazon EMR.

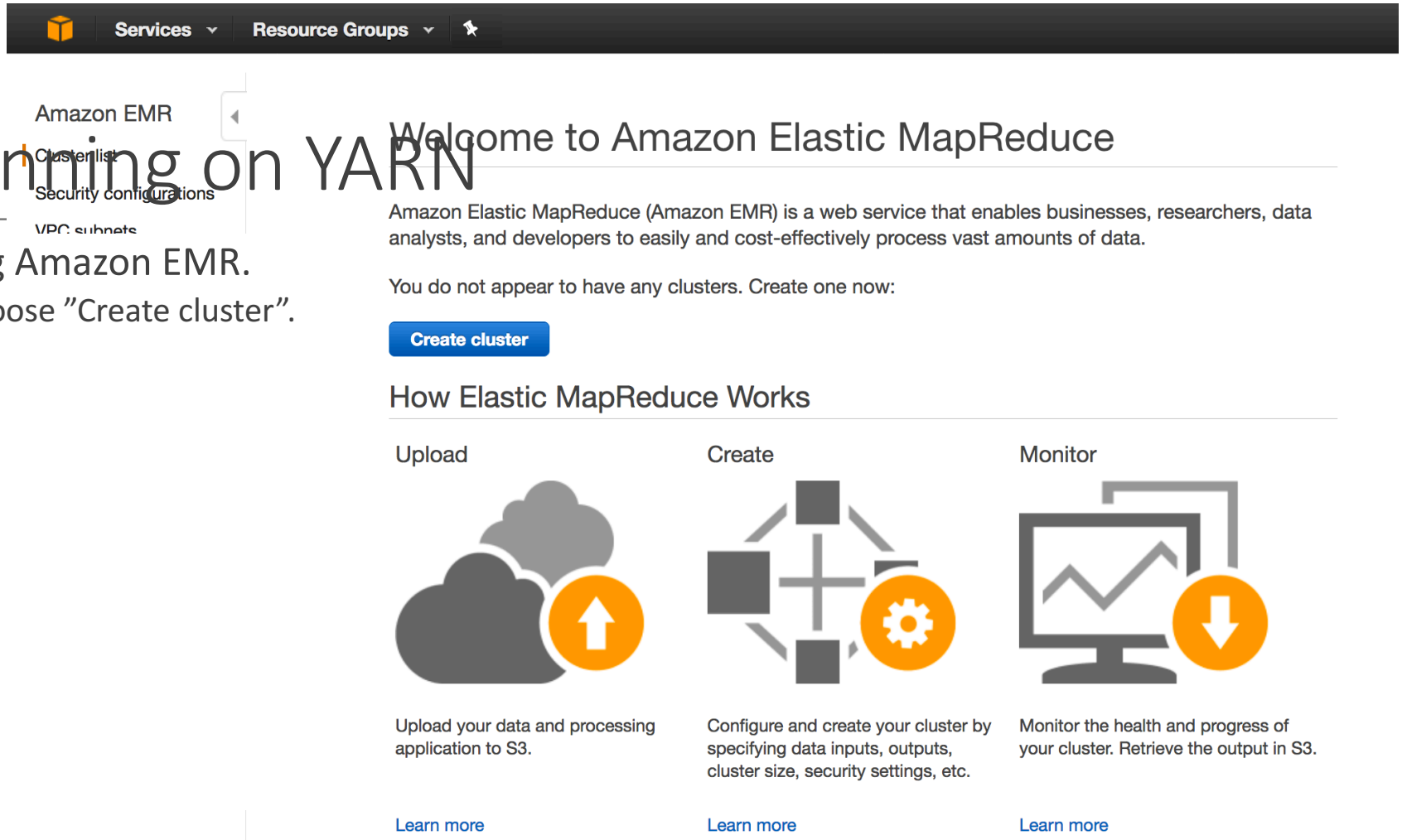
- On your AWS console, choose EMR under Analysis.



Running on YARN

Using Amazon EMR.

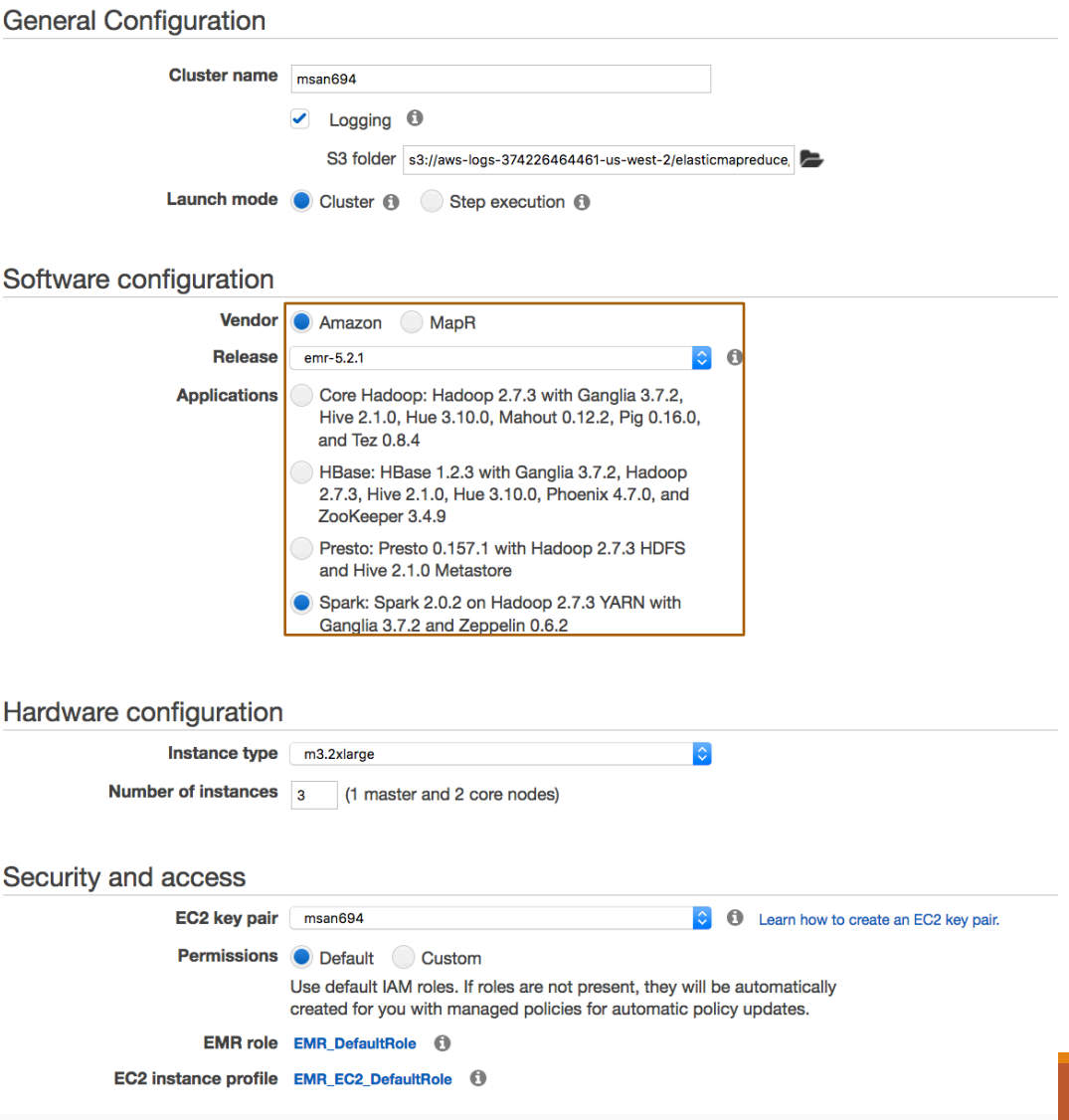
- Choose ”Create cluster”.



Running on YARN

Using Amazon EMR.

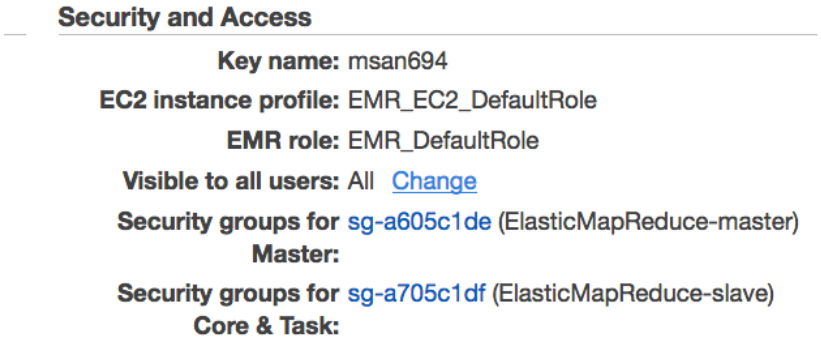
- Insert all the information including Cluster name, Applications (Spark), Hardware configuration,etc.



Running on YARN

Using Amazon EMR.

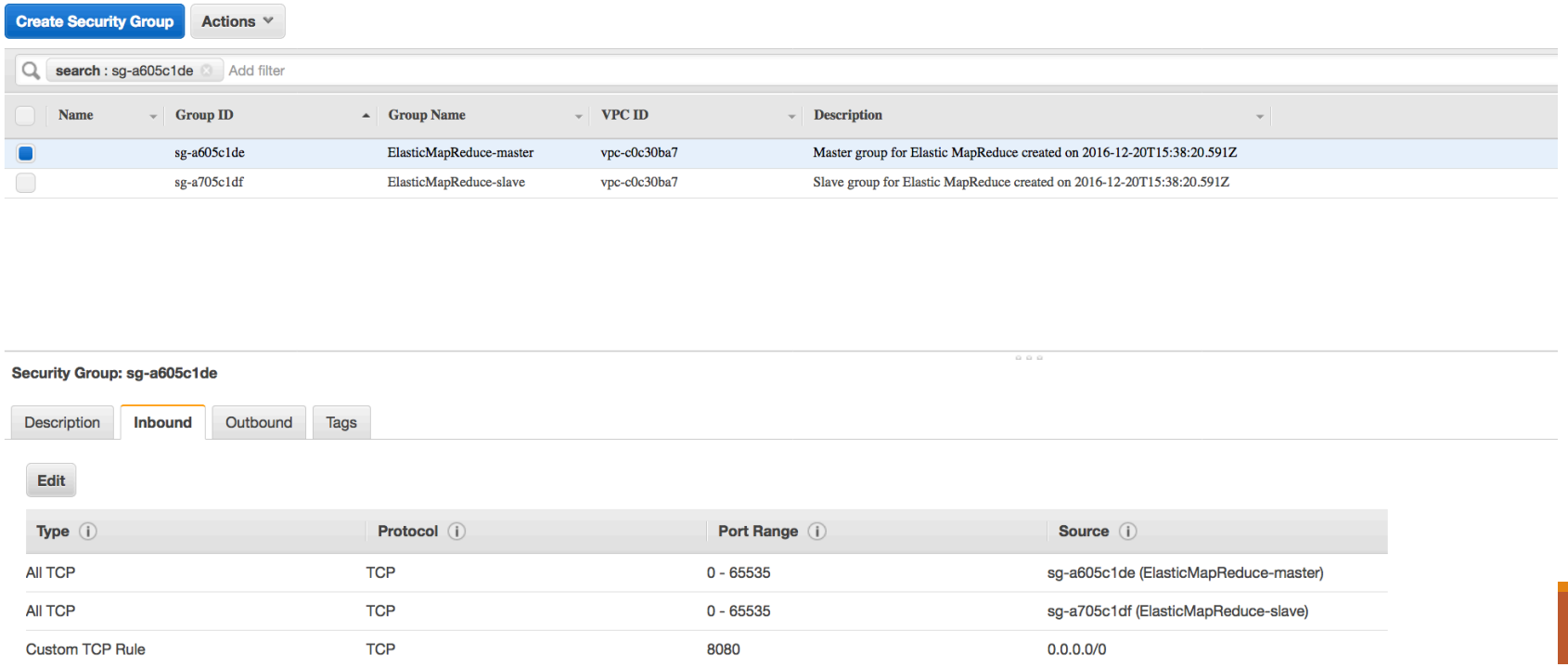
- Once it is created, choose Security groups for Master under Security Access.



Running on YARN

Using Amazon EMR.

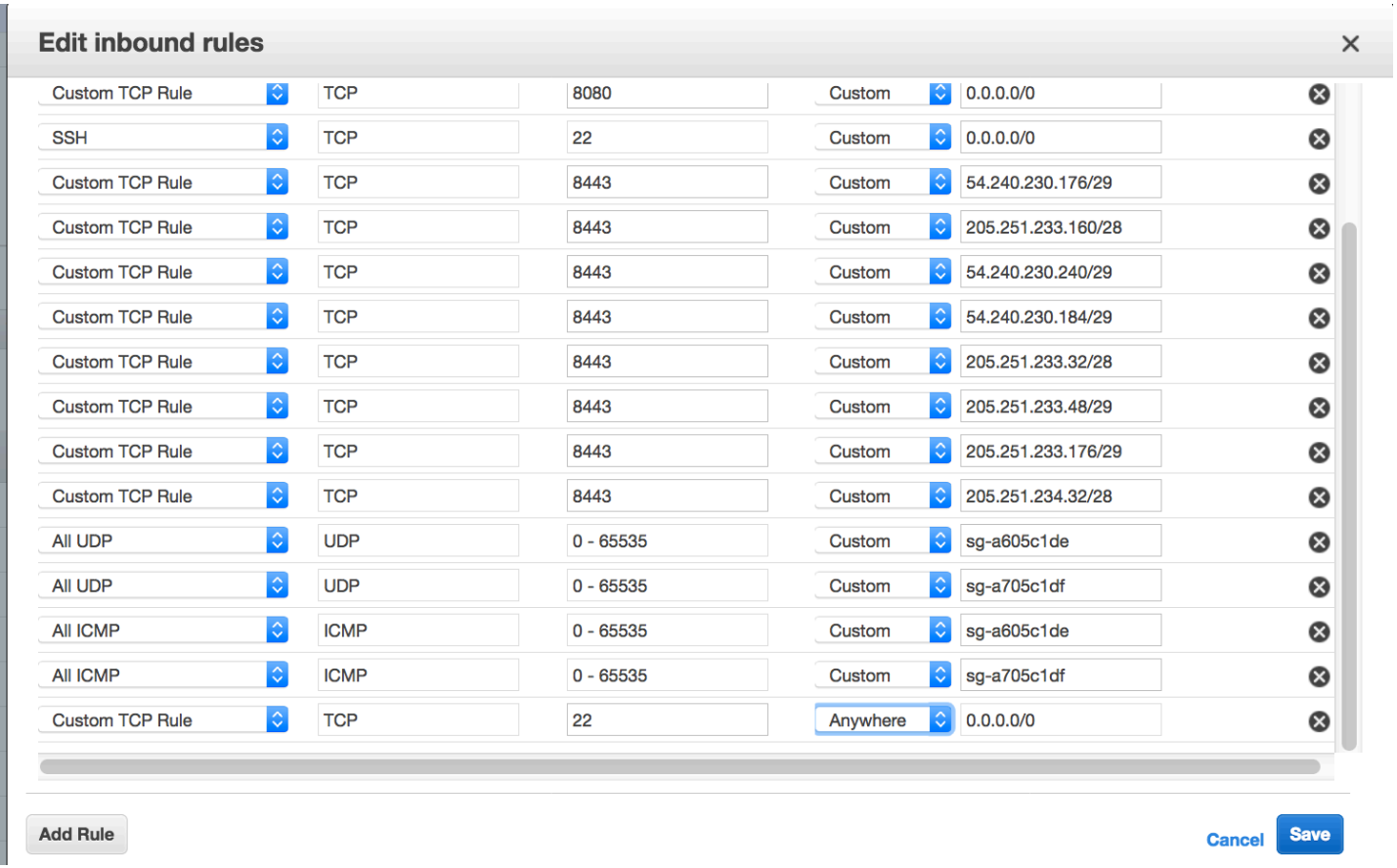
- On the master, add an inbound rule if you don't see SSH (Protocol : TCP, Port : 22, Source : 0.0.0.0/0)



Running on YARN

Using Amazon EMR.

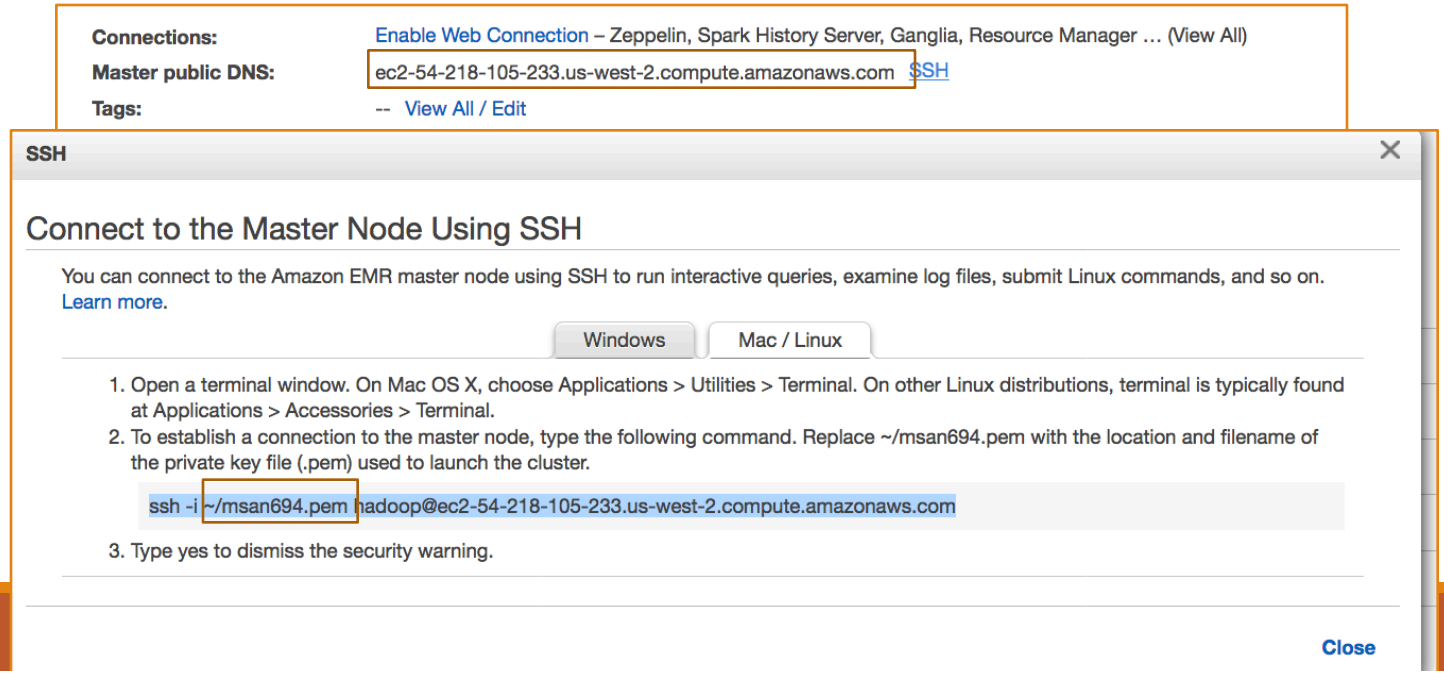
- Custom TCP Rule to allow port 22 from Anywhere. (Enabled)



Running on YARN

Using Amazon EMR.

- On your EMR Cluster, click SSH on your Master public DNS and copy the command. (Make sure .pem location is same as yours.)



Running on YARN

Using Amazon EMR.

- Run the command and say “yes” when it asks ”Are you sure you want to continue connecting?”

```
ML-ITS-603436:spark-2.0.0-bin-hadoop2.7 dwoodbridge$ ssh -i msan694.pem  hadoop@ec2-54-218-105-233.us-west-2.compute.amazonaws.com
The authenticity of host 'ec2-54-218-105-233.us-west-2.compute.amazonaws.com (54.218.105.233)' can't be established.
ECDSA key fingerprint is SHA256:lvuUviA9QfdsQoyEngU4aEQ8eqyNOa21uv1g2BWK16I.
Are you sure you want to continue connecting (yes/no)? yes
```

Running on YARN

Using Amazon EMR.

```
ML-ITS-603436:spark-2.0.0-bin-hadoop2.7 dwoodbridge$ ssh -i msan694.pem  hadoop@ec2-54-218-105-233.us-west-2.compute.amazonaws.com
The authenticity of host 'ec2-54-218-105-233.us-west-2.compute.amazonaws.com (54.218.105.233)' can't be established.
ECDSA key fingerprint is SHA256:lvuUviA9QfdsQoyEngU4aEQ8eqyNOa21uv1g2BWK16I.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-54-218-105-233.us-west-2.compute.amazonaws.com,54.218.105.233' (ECDSA) to the list of known hosts.
Last login: Sun Jan  1 02:12:37 2017

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2016.09-release-notes/

EEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR:::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-11-131 ~]$
```

Running on YARN

Using Amazon EMR.

- Try Pyspark.

```
[hadoop@ip-172-31-11-131 ~]$ pyspark
Python 2.7.12 (default, Sep  1 2016, 22:14:00)
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel).
17/01/01 02:21:37 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 2.0.2
      /_/

Using Python version 2.7.12 (default, Sep  1 2016 22:14:00)
SparkSession available as 'spark'.
>>>
```

Running on YARN

Using Amazon EMR.

- Set Jupyter Notebook.
  - First install anaconda.

    `wget http://repo.continuum.io/archive/Anaconda3-4.1.1-Linux-x86_64.sh`

    `sh Anaconda3-4.1.1-Linux-x86_64.sh`
    - Add “export PATH=/home/hadoop/anaconda3/bin:$PATH” in your .bashrc.
  - Make sure to install on place with enough disk space.

```
[hadoop@ip-172-31-15-224 ~]$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
devtmpfs        15382916      28  15382888   1% /dev
tmpfs           15413684       0  15413684   0% /dev/shm
/dev/xvda1      10190136 6901256   3188612  69% /
/dev/xvdb1       5232640   36772   5195868   1% /emr
/dev/xvdb2      73352692  309812  73042880   1% /mnt
/dev/xvdc       78594056   34052  78560004   1% /mnt1
[hadoop@ip-172-31-15-224 ~]$ cd /mnt
```

https://medium.com/@josemarcialportilla/getting-spark-python-and-jupyter-notebook-running-on-amazon-ec2-dec599e1c297#.csw3m7v6b

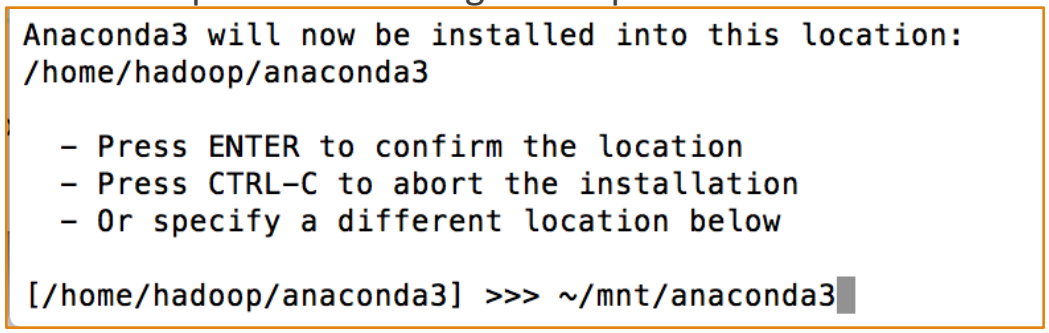
Running on YARN

Using Amazon EMR.

- Set Jupyter Notebook.
  - First install anaconda.

    `wget http://repo.continuum.io/archive/Anaconda3-4.1.1-Linux-x86_64.sh`

    `sh Anaconda3-4.1.1-Linux-x86_64.sh`
    - Add “export PATH=/home/hadoop/mnt/anaconda3/bin:$PATH” in your .bashrc.
  - Make sure to install on place with enough disk space.



https://medium.com/@josemarcialportilla/getting-spark-python-and-jupyter-notebook-running-on-amazon-ec2-dec599e1c297#.csw3m7v6b

Running on YARN

Using Amazon EMR.

- Set Jupyter Notebook.
  - First install anaconda.

    `wget http://repo.continuum.io/archive/Anaconda3-4.1.1-Linux-x86_64.sh`

    `sh Anaconda3-4.1.1-Linux-x86_64.sh`
    - Add “export PATH=/home/hadoop/anaconda3/bin:$PATH” in your .bashrc.
  - Change the python version to 2.7.

```
[hadoop@ip-172-31-41-248 bin]$ which python
~/anaconda3/bin/python
```

```
[hadoop@ip-172-31-15-224 mnt]$ conda install python=2.7
Fetching package metadata .......
Solving package specifications: .......
```

https://medium.com/@josemarcialportilla/getting-spark-python-and-jupyter-notebook-running-on-amazon-ec2-dec599e1c297#.csw3m7v6b

Running on YARN

Using Amazon EMR.

- Set Jupyter Notebook.
  - Configure Jupyter
    - Jupyter comes with Anaconda, but we will need to configure it in order to use it through EC2 and connect with SSH.

      `jupyter notebook --generate-config`

```
[hadoop@ip-172-31-15-224 ~]$ source ~/.bashrc
[hadoop@ip-172-31-15-224 ~]$ jupyter notebook --generate-config
Writing default config to: /home/hadoop/.jupyter/jupyter_notebook_config.py
```

Running on YARN

Using Amazon EMR.

- Set Jupyter Notebook.
  - Create certifications.
    - mkdir certs
    - cd certs
    - sudo openssl req -x509 -nodes -days 365 -newkey rsa:1024 -keyout msan694.pem -out msan694.pem

```
Generating a 1024 bit RSA private key
.........................++++++
.....................................................................................++++++
writing new private key to 'msan694.pem'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:US
State or Province Name (full name) []:California
```

Running on YARN

Using Amazon EMR.

- Edit configuration file.
  - Change the configuration we generated.
    - cd ~/.jupyter
    - vi jupyter_notebook_config.py

```
# Configuration file for jupyter-notebook.

c= get_config()
c.NotebookApp.certfile = u'/home/hadoop/certs/msan694.pem'
c.NotebookApp.ip = '*' # Run on all IP addresses of your instance
c.NotebookApp.open_browser = False   # Don't open browser by default
c.NotebookApp.port = 8888 # Use port 8888 for notebook
```

 # Notebook config this is where you saved your pem cert

Running on YARN

Using Amazon EMR.

- Add PYSPARK_DRIVER_PYTHON and PYSPARK_DRIVER_PYTHON_OPTS.

```
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```

Running on YARN

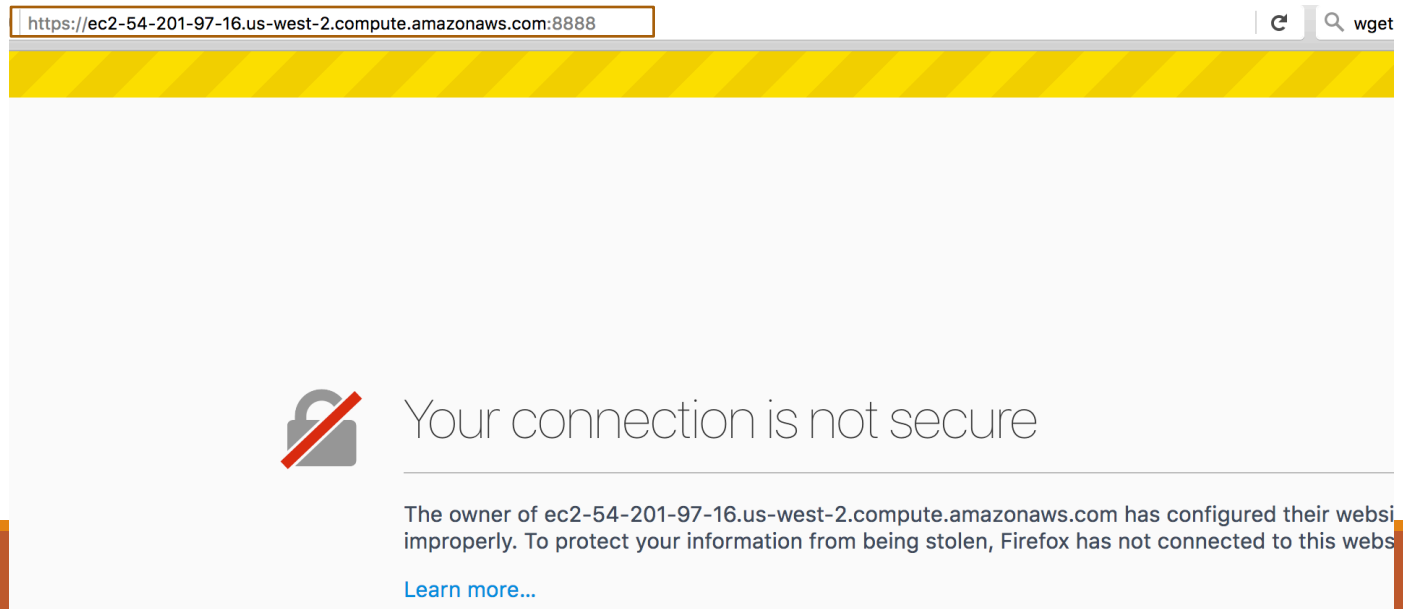
Using Amazon EMR.

- Check jupyter notebook and enable notebook for pyspark.
  - Make sure port 8888 is available.

Running on YARN

Using Amazon EMR.

- Check jupyter notebook and enable notebook for pyspark.
  - type jupyter notebook and access it by Master public DNS:8888 on your computer.



Running on YARN

Using Amazon EMR.

- Edit configuration file.
  - Change the configuration we generated.
    - Add PYSPARK_DRIVER_PYTHON and PYSPARK_DRIVER_PYTHON_OPTS on ~/.bashrc.

```
# Source global definitions
if [ -f /etc/bashrc ]; then
  . /etc/bashrc
fi

# set the default region for the AWS CLI
export AWS_DEFAULT_REGION=$(curl --retry 5 --silent -
export JAVA_HOME=/etc/alternatives/jre
# added by Anaconda3 4.1.1 installer
export PATH="/home/hadoop/anaconda3/bin:$PATH"

export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS="notebook"
```

Running on YARN

Running Spark on Amazon EMR.

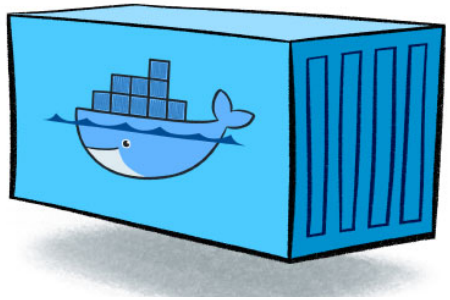
- Apache Spark on Hadoop YARN is natively supported in Amazon EMR, and you can quickly and easily create managed Apache Spark clusters from the AWS Management Console, AWS CLI, or the Amazon EMR API.
- You can leverage additional Amazon EMR features, including fast Amazon S3 connectivity using the Amazon EMR File System (EMRFS), integration with the Amazon EC2 Spot market, and Auto Scaling to add or remove instances from your cluster.
- Reference
  - http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-spark.html
  - http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-spark-shell.html

https://aws.amazon.com/emr/details/spark/

Running on Mesos

Mesos

- The spark project was originally started in order to demonstrate the usefulness of Mesos.
- Example – Apple Siri, eBay, Netflix, Twitter, Uber, etc.
- Support applications written in Java, C, C++, and Python.
- Run Docker containers
  - Docker container -Docker containers wrap a piece of software in a complete filesystem that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment.
  - With Docker support, you can run Mesos on any application.



Running on Mesos

Mesos architecture

Mesos resource scheduling

Using Amazon CloudFormation

Running on Mesos

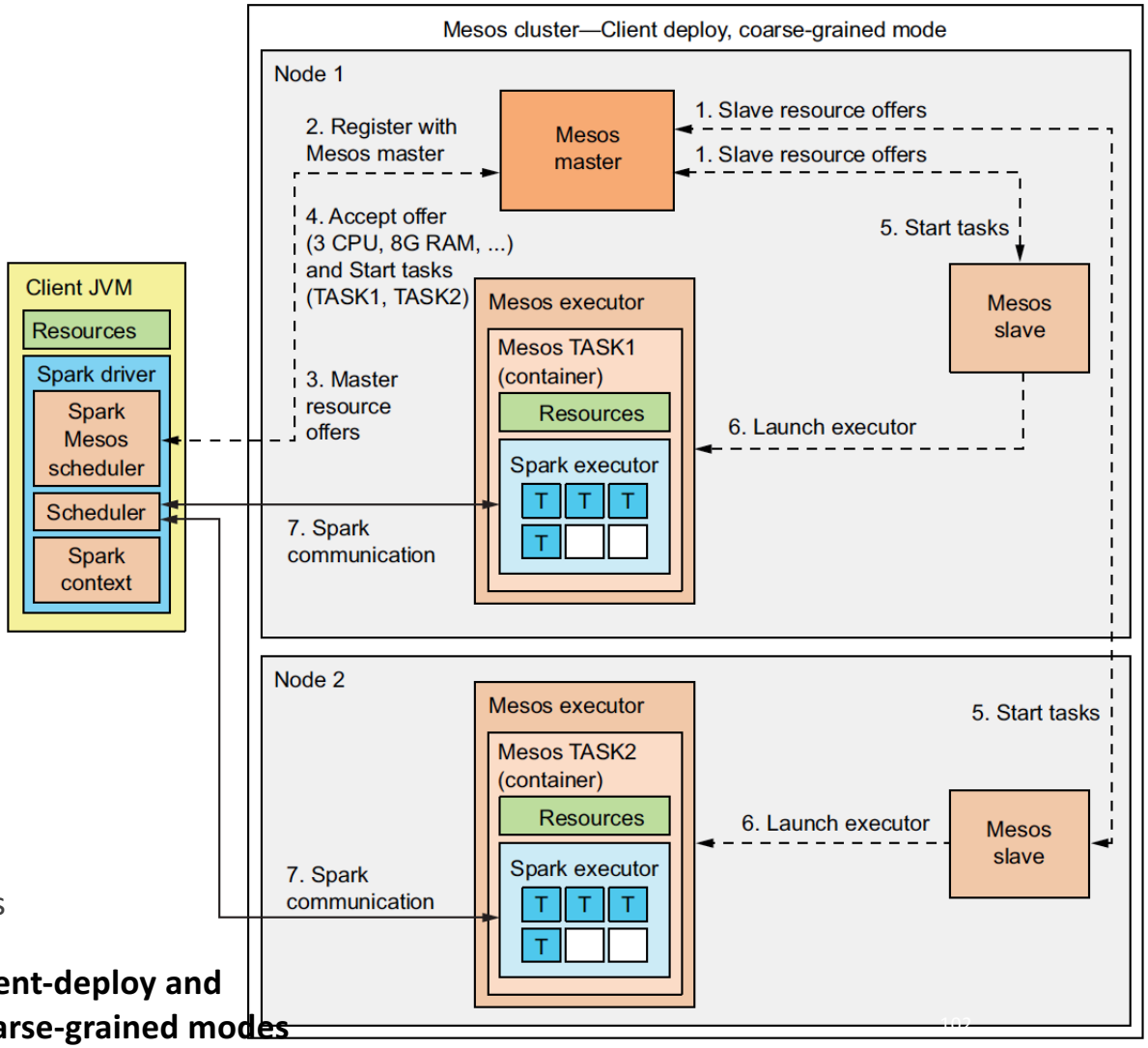
Mesos architecture

- Components
  - Masters
    - Schedule slave resources among applications.
    - Capable of scheduling disk space, network ports and even custom resource.
  - Slaves
    - Launch the application's executors, which execute tasks.
  - Applications (frameworks) – has a scheduler and an executor.
    - **Instead of applications demanding resources from the master, a master offers resources to applications, which they can accept or refuse.**
    - Scheduler : Accept or reject resources offered by the Mesos master and starts Mesos executors on slaves.
    - Executor: Run tasks as requested by the frameworks' schedulers.

Running on Mesos

Mesos architecture

1. Mesos slaves offer their resources to the master.
2. Spark's Mesos-specific scheduler, running in a driver registers with the Mesos master.
3. The Mesos master in turn offers the available resources to the Spark Mesos scheduler.
4. Spark's Mesos scheduler accepts some of the resources and sends a list of resources, along with a list of tasks it wants to run using the resources, to the Mesos master.
5. The master asks the slaves to start the tasks with the requested resources.
6. Slaves launch executors, which launch Spark executors in task containers.
7. Spark executors connect to the Spark driver and freely communicate with it, executing Spark tasks as usual.

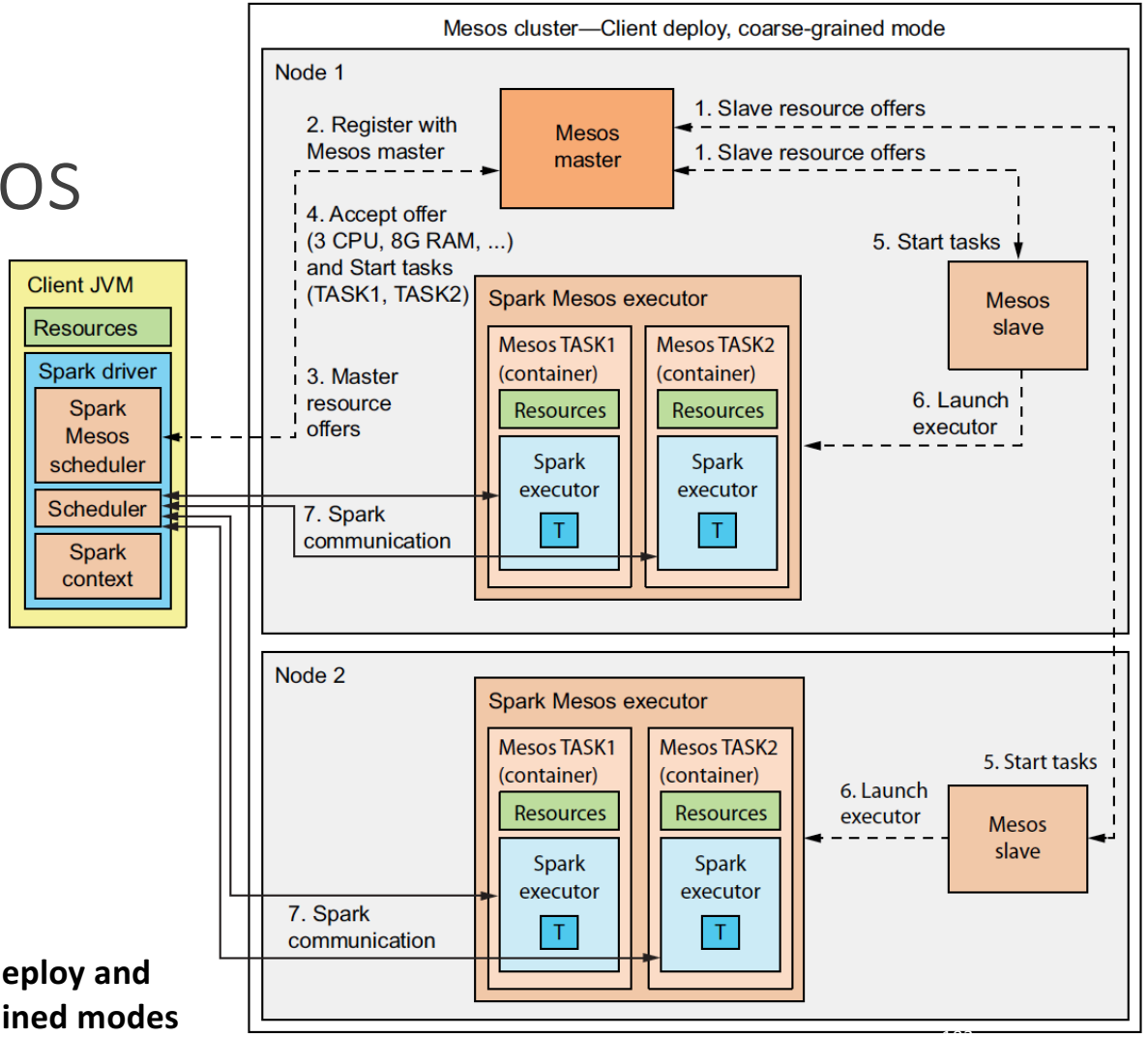


Client-deploy and coarse-grained modes

Running on Mesos

Mesos architecture

- Coarse-grained and fine-grained modes.
  - Coarse-grained : Spark starts one Spark executor per Mesos slave.
  - Fine-grained mode : one Spark executor is started per Spark task.
    - More communication, data serialization and setting up of Spark executor processes will be needed.
    - Could be slower, but it uses cluster resources more flexibly.
    - It's used mainly for batch or streaming jobs that have long-running tasks, because the slowdown due to management of Spark executors is negligible.



Client-deploy and fine-grained modes

Running on Mesos

Mesos architecture

- Also supports ZooKeeper to elect for master high-availability.

Running on Mesos

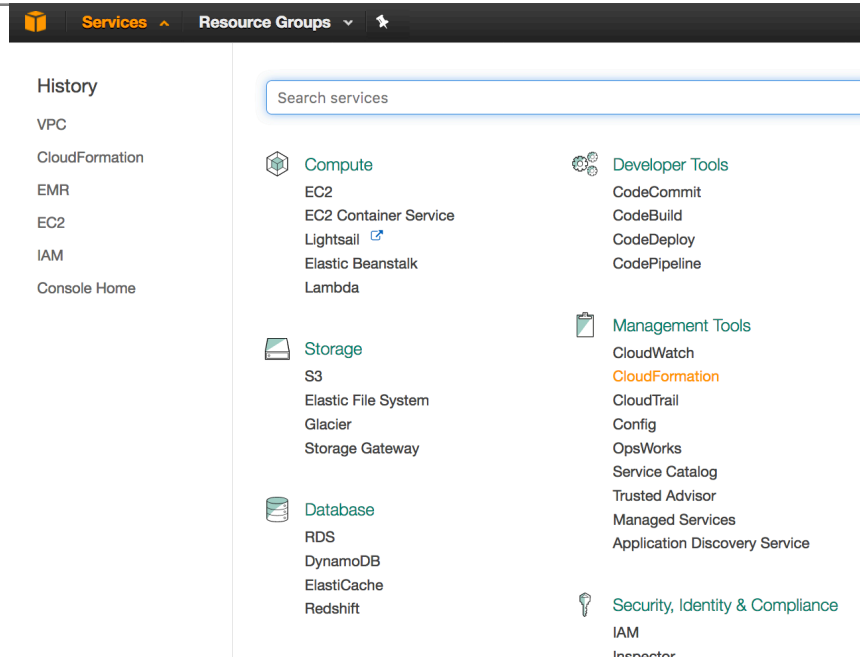
Mesos resource scheduling

- Dominant Resource Fairness (DRF) algorithm (Default)
  - Track each framework's resources and determine the dominant resource for each framework, which is the resource type the framework uses the most.
  - Then calculate the framework's dominant share which is the share of all cluster resource of the dominant resource type used by the framework.
    - Example. If a framework is using 1 CPU and 6 GB of RAM in a cluster of 20 CPU s and 36 GB of RAM, then it's using 1/20 of all CPU s and 1/6 of all RAM ; thus its dominant resource is RAM , and its dominant share is 1/6.
  - DRF allocates the newest resource to the framework with the lowest dominant share.
  - The framework can accept or reject the offer. If it currently has no work to do, or if the offer doesn't contain enough resources for its needs, the framework can reject the offer.

Running on Mesos

Using Amazon CloudFormation

- AWS DCOS *(Datacenter Operating System):* The Mesosphere Datacenter Operating System (DCOS) is a cluster management platform designed to abstract physical and cloud-based resources behind a single API and this includes Apache Mesos.
- Download key pairs (.pem).
- Create a Mesosphere DCOS Cluster by Using AWS CloudFormation to launch a stack.
- Once the stack is created, access the Mesosphere DCOS dashboard and install/configure Spark.



https://aws.amazon.com/blogs/apn/announcing-mesosphere-dcos-on-aws/
https://mesosphere.com/amazon/
https://dcos.io/docs/1.7/usage/tutorials/spark/

Running on a Cluster

Standalone

Hadoop YARN

Mesos Cluster

EMR*

Reference

Distributed Computing with Spark, Reza Zadeh, http://stanford.edu/~rezab/slides/bayacm_spark.pdf.

Spark Online Documentation : http://spark.apache.org/docs/latest/

Karau, Holden, et al. *Learning spark: lightning-fast big data analysis*. O'Reilly Media, Inc., 2015.

Zecevic, Petar, et al. Spark in Action, Manning, 2016.