Class 7: Writing and Deploying Spark Applications

New York University

Summer 2017

Agenda

1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

- **In addition to the Spark Shell, we can also write Scala Spark programs and compile them**
  - **As we have seen, the shell (or REPL) allows interactive exploration and manipulation of data**
- **Compiled Spark applications can be run as independent programs**
  - E.g., for ETL processing

Agenda

1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

- **Every Spark program needs a SparkContext**
  - The interactive shell creates one for you
- **In your own Spark application you must create the SparkContext**
  - Named `sc` by convention
  - Call `sc.stop` when program terminates

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._

object WordCount {
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: WordCount <file>")
      System.exit(1)
    }

    val sc = new SparkContext()

    val counts = sc.textFile(args(0)).
       flatMap(line => line.split("\\W")).
       map(word => (word,1)).reduceByKey(_ + _)
    counts.take(5).foreach(println)

    sc.stop()
  }
}
```

Agenda

1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

- **Scala or Java Spark applications must be compiled and assembled into JAR files**
  - JAR file will be passed to worker nodes
- **Apache Maven is a popular tool for building applications**
  - For specific setting recommendations, see **`http://spark.apache.org/docs/latest/building-with-maven.html`**
- **Build details will differ depending on**
  - Version of Hadoop
  - Deployment platform (Spark Standalone, YARN, Mesos)
- **You can use an IDE for code development**
  - IntelliJ or Eclipse are two popular IDEs
  - Can run Spark locally in a debugger

Agenda

1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

- **The easiest way to run a Spark Application is using the `spark-submit` script**

```
$ spark-submit --class WordCount MyJarFile.jar fileURL
```

- **Spark can run**
  - Locally
    - No distributed processing
  - Locally with multiple worker threads
  - On a cluster
- **Local mode is useful for development and testing**
- **Production use is almost always on a cluster**

Spark Platform Options

- **Apache Mesos**
  - First platform supported by Spark
- **Spark Standalone**
  - Included with Spark
  - Easy to install and run
  - Limited configurability and scalability
  - Useful for testing, development, or small systems

Spark Platform Options *(continued)*

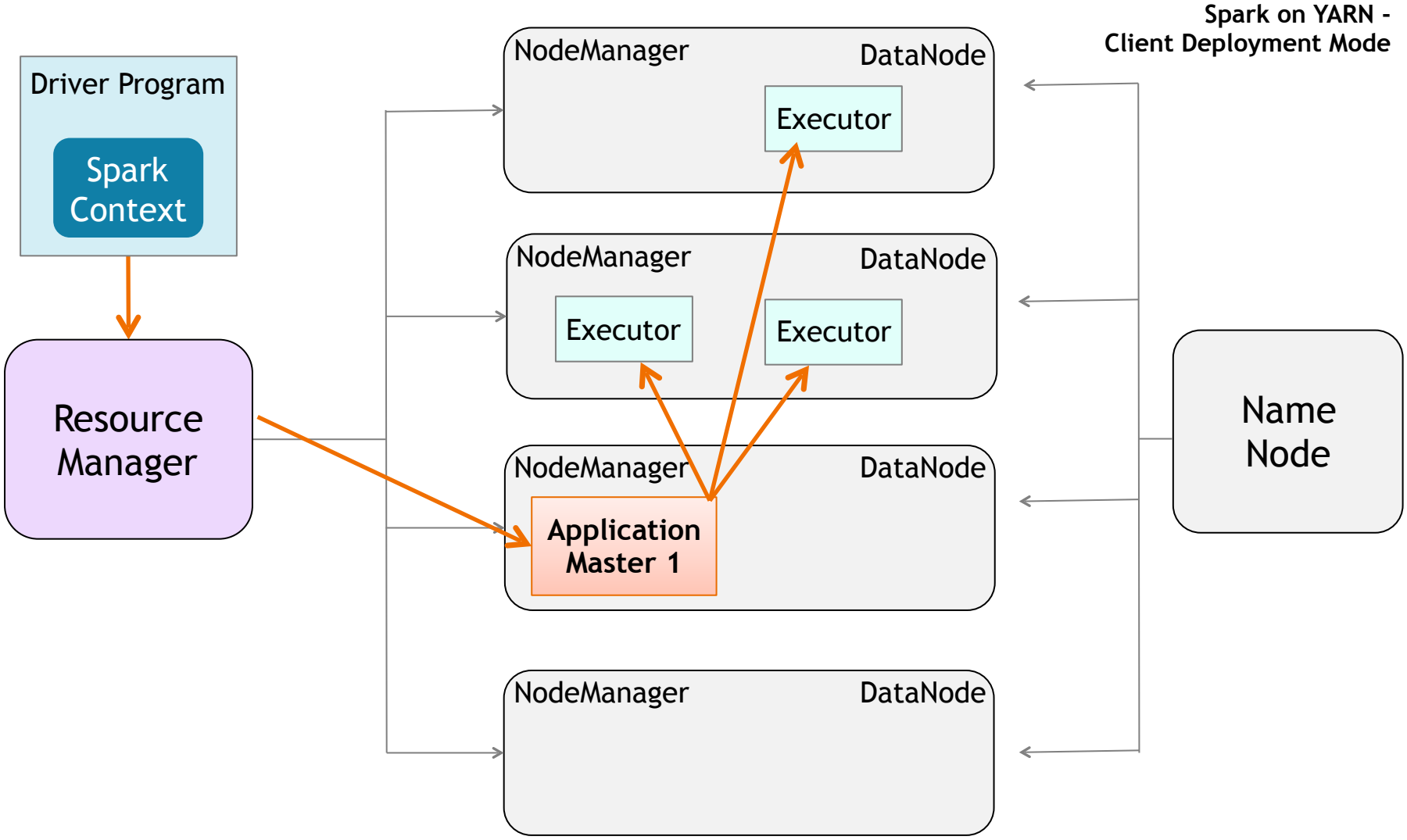
- **YARN (Yet Another Resource Negotiator)**
  - Included in CDH as well as other Hadoop distributions
  - Allows sharing cluster resources with other applications (e.g. MapReduce)
- **We will focus on Spark on YARN in the slides that follow**
  - Spark on YARN is the most common platform in production deployments

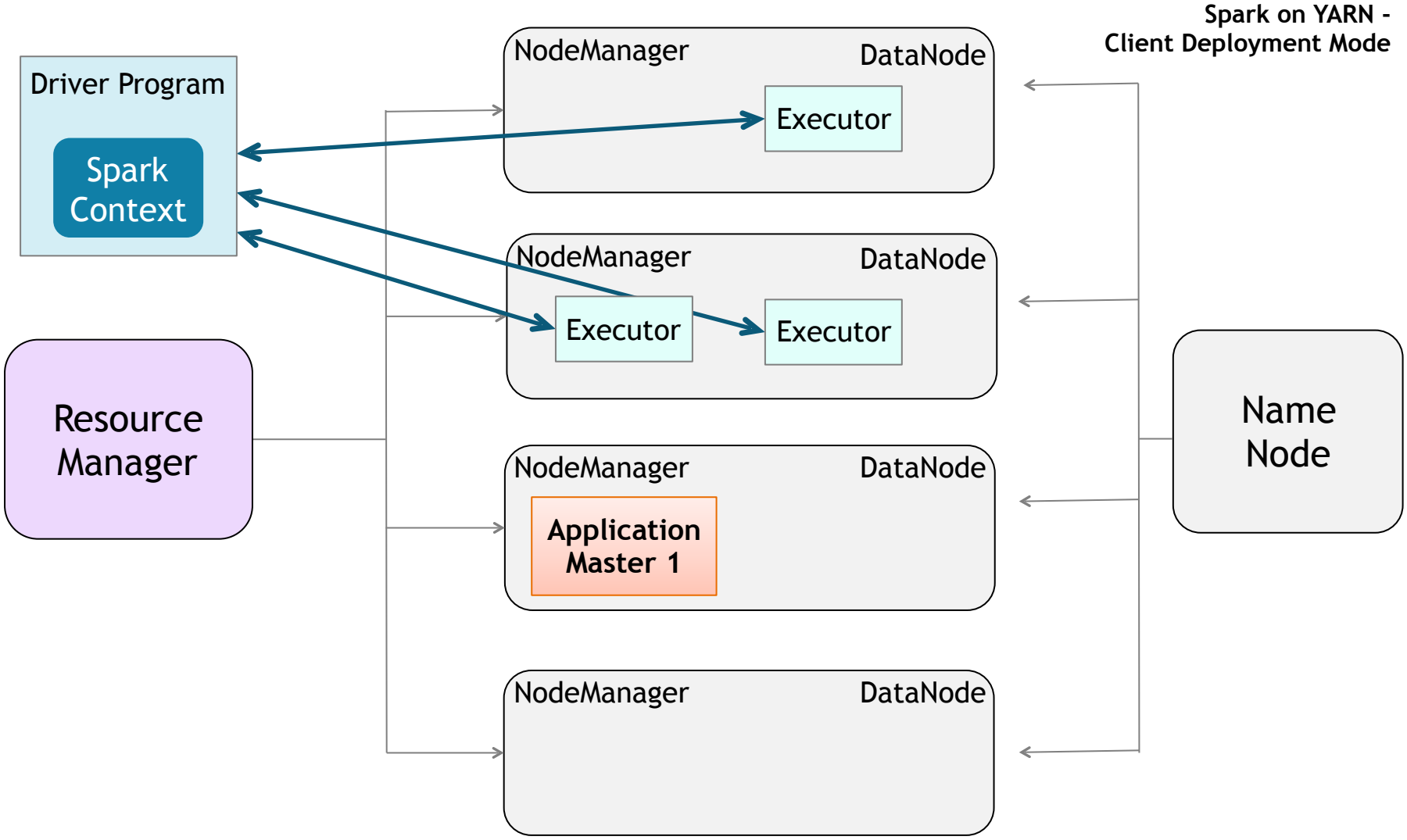
Spark Deployment Modes

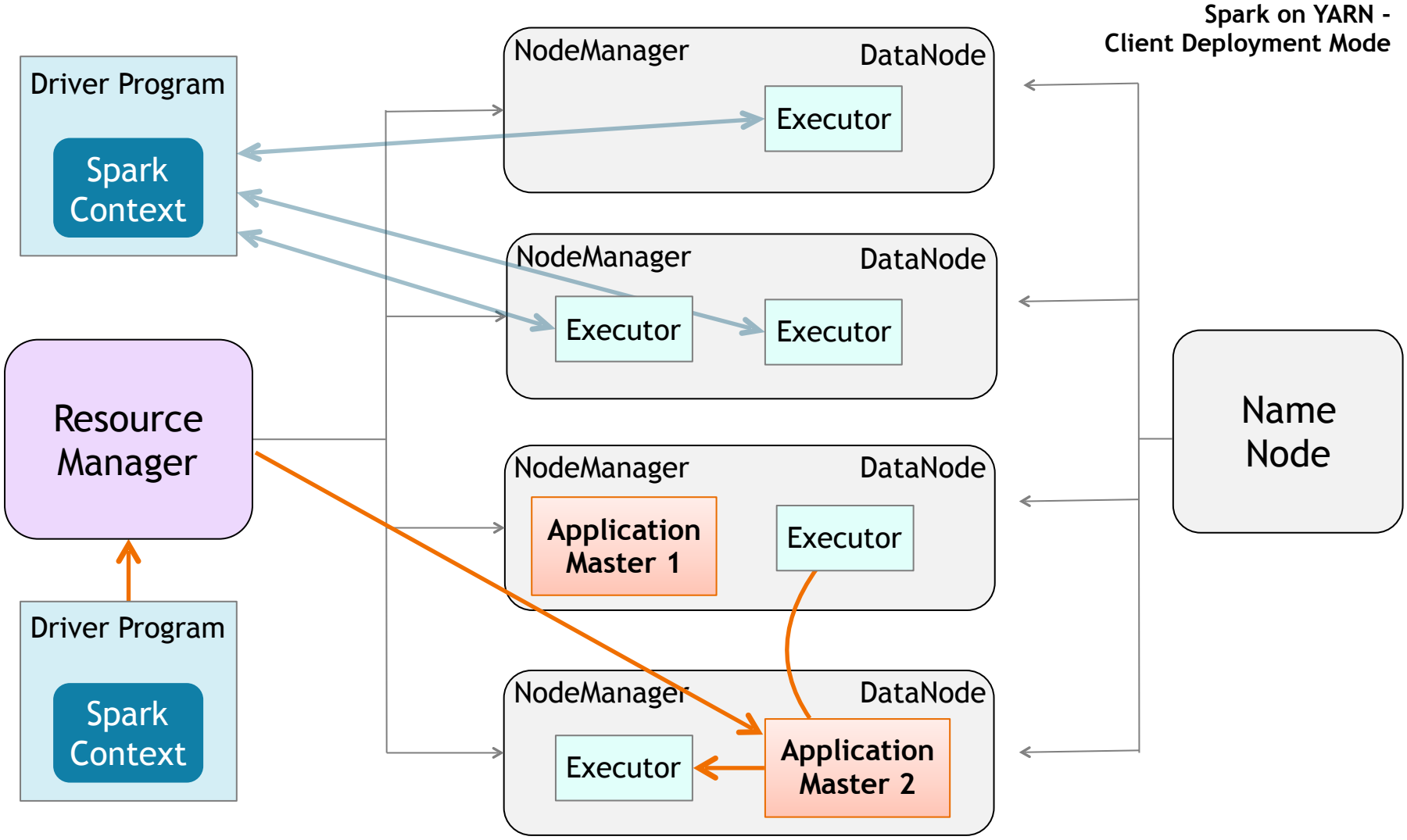
- Spark Client Deployment Mode
- Spark Cluster Deployment Mode

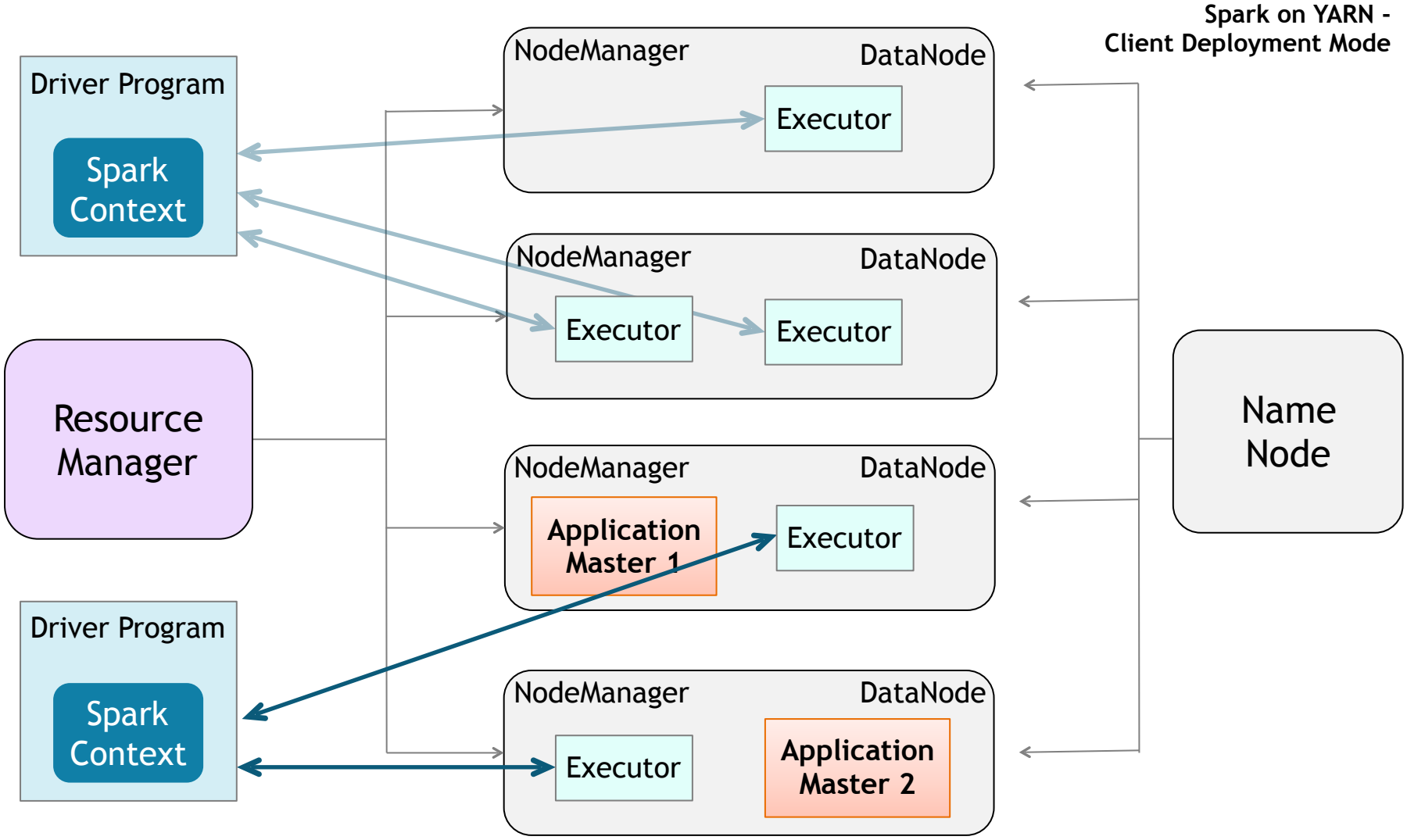
Spark Deployment Modes

- **Let's look at Spark client deployment mode first**
  - In client mode, the driver runs on the host where the job was submitted
  - The ApplicationMaster requests executor containers from the ResourceManager
- **Client mode supports interactive use of Spark, cluster mode does not**
  - Spark applications that require user input need the Spark driver to run inside the client process that initiated the Spark application
    - For example, applications like spark-shell and pyspark



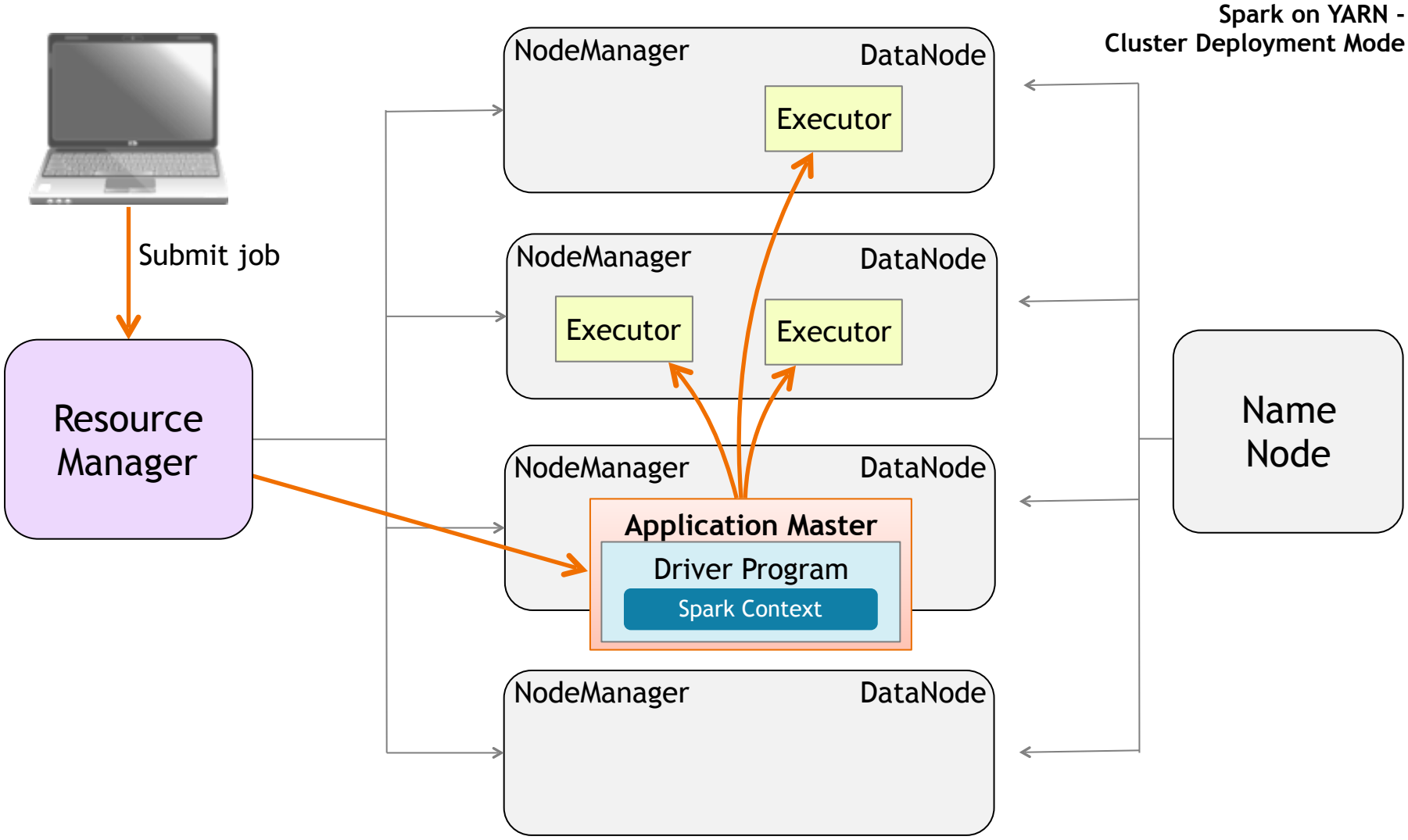




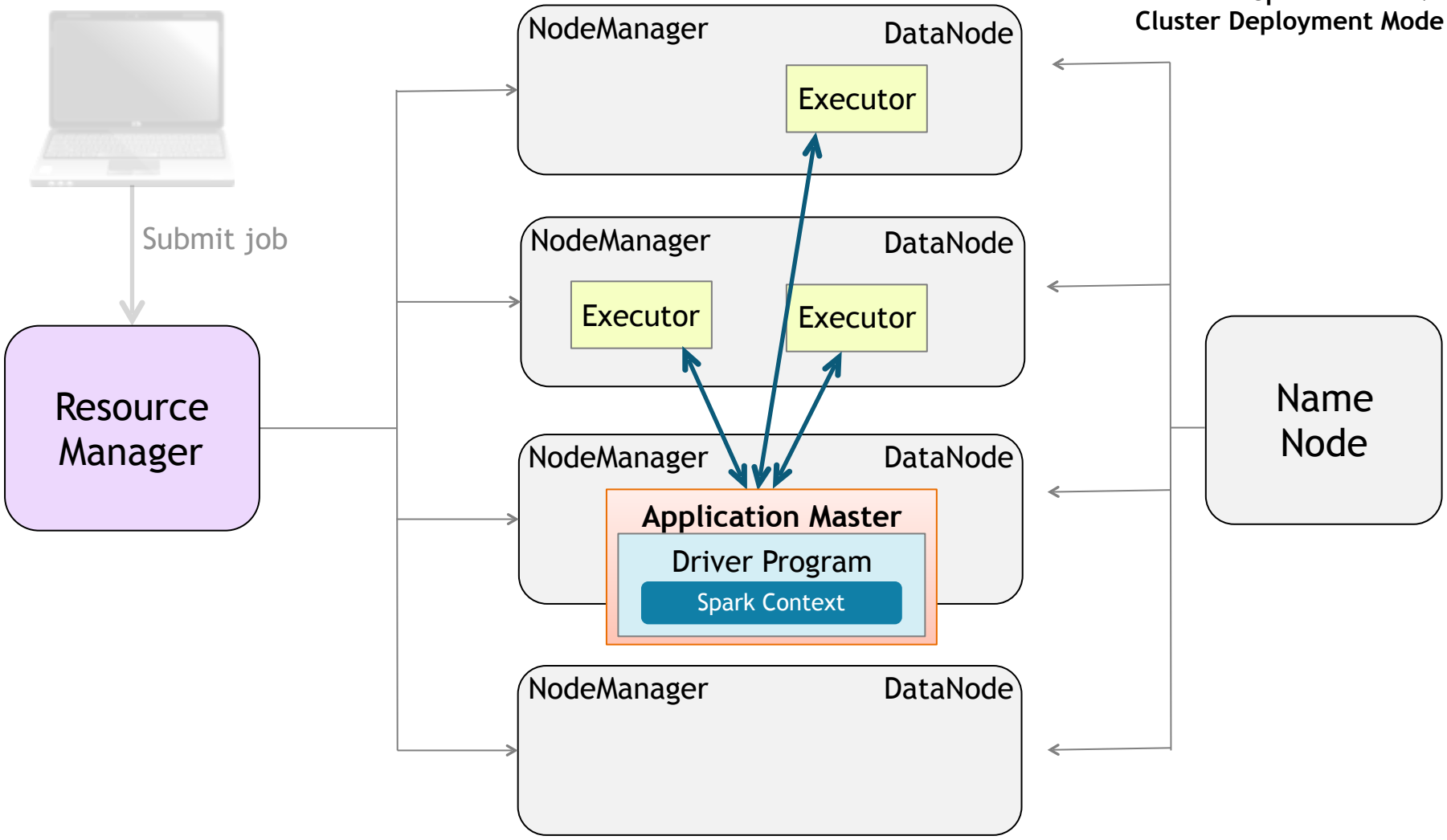


Spark Deployment Modes *(continued)*

- **Let's look at Spark cluster deployment mode next**
  - In cluster mode, the driver runs in the ApplicationMaster on a cluster host chosen by YARN
  - The ApplicationMaster process is responsible for driving the application and requesting resources from the ResourceManager
  - The client that launches the application doesn't need to continue running for the entire lifetime of the application



Spark on YARN - Cluster Deployment Mode



- **Use `spark-submit --master` to specify cluster options**
  - Local options
    - **`local[*]`** - run locally with as many threads as cores (default)
    - **`local[n]`** - run locally with n threads
    - **`local`** - run locally with a single thread

```
$ spark-submit --master local[3] —class WordCount MyJarFile.jar fileURL
```

- **Use `spark-submit --master` to specify cluster options**
  - Cluster options
    - **`yarn-client`**
    - **`yarn-cluster`**

```
$ spark-submit --master yarn-cluster --class WordCount MyJarFile.jar fileURL
```

- **The Spark Shell can also be run on a cluster**
- **Pyspark and spark-shell both have a `--master` option**
  - **`yarn`** (client mode only)

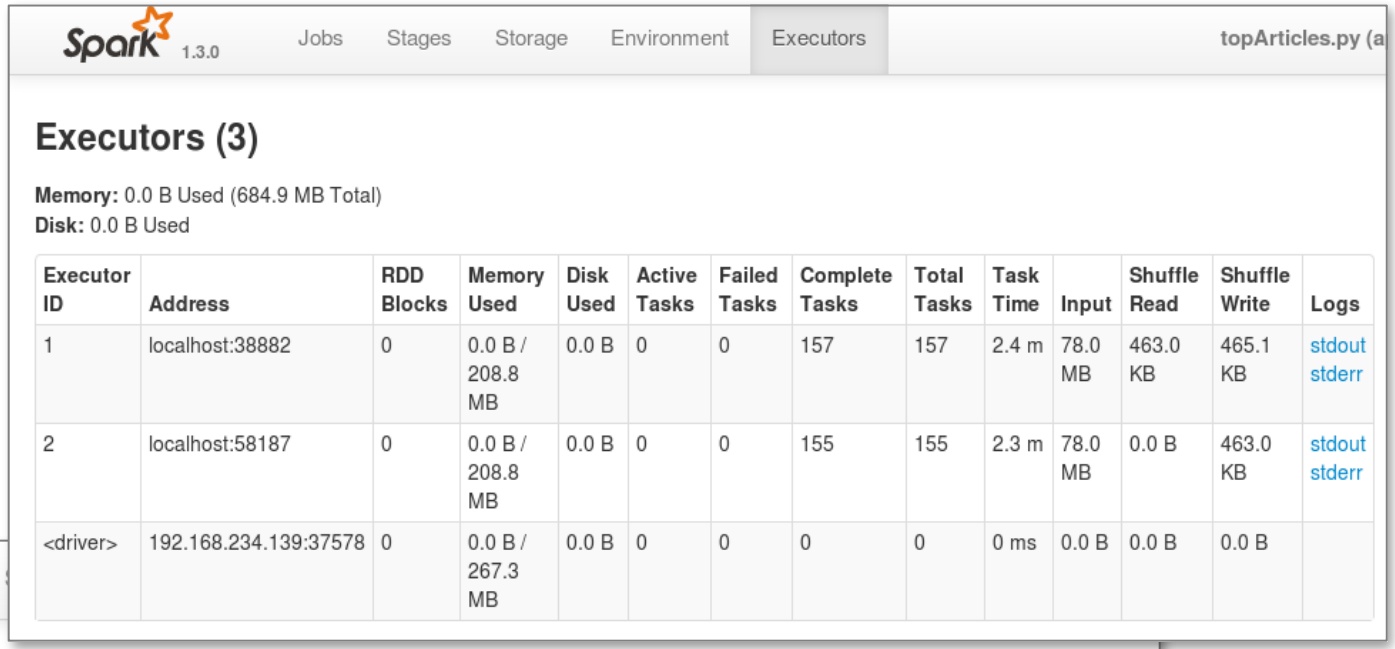
```
$ spark-shell --master yarn
```

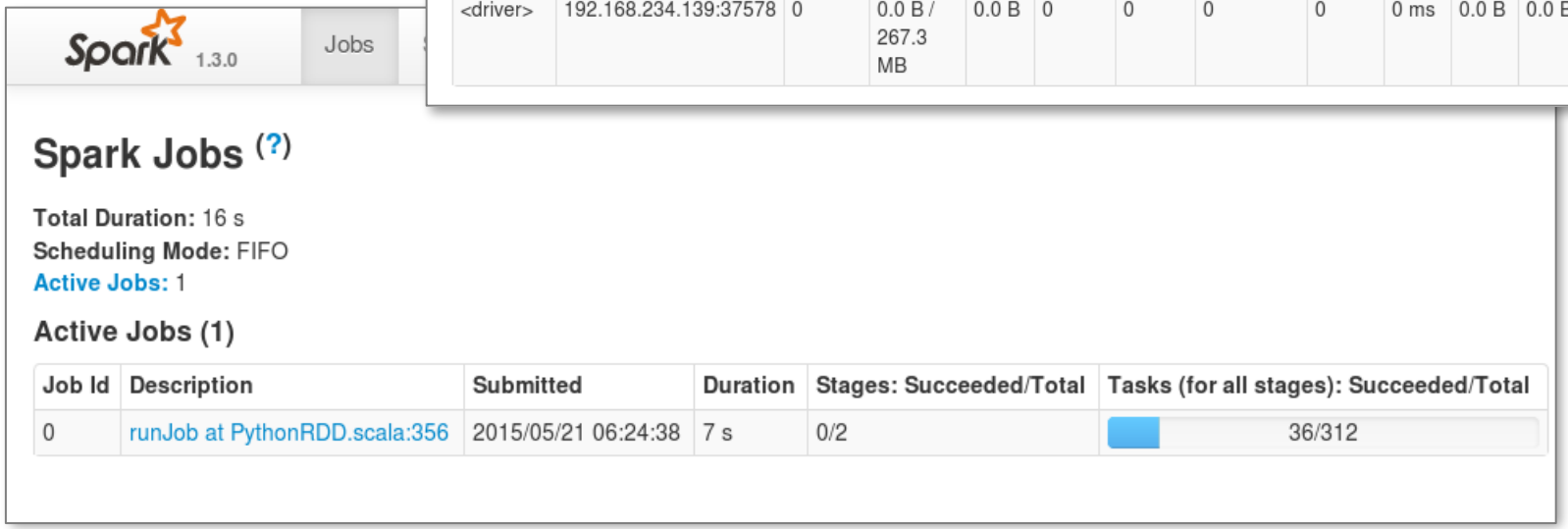
- **Some other `spark-submit` options for clusters**
  - **`--jars`** – additional JAR files (Scala and Java only)
  - **`--py-files`** – additional Python files (Python only)
  - **`--driver-java-options`** – parameters to pass to the driver JVM
  - **`--executor-memory`** – memory per executor (e.g. 1000M, 2G) (Default: 1G)
  - **`--packages`** -- Maven coordinates of an external library to include
- **Plus several YARN-specific options**
  - **`--num-executors`**
  - **`--queue`**
- **Show all available options**
  - **`--help`**

Agenda

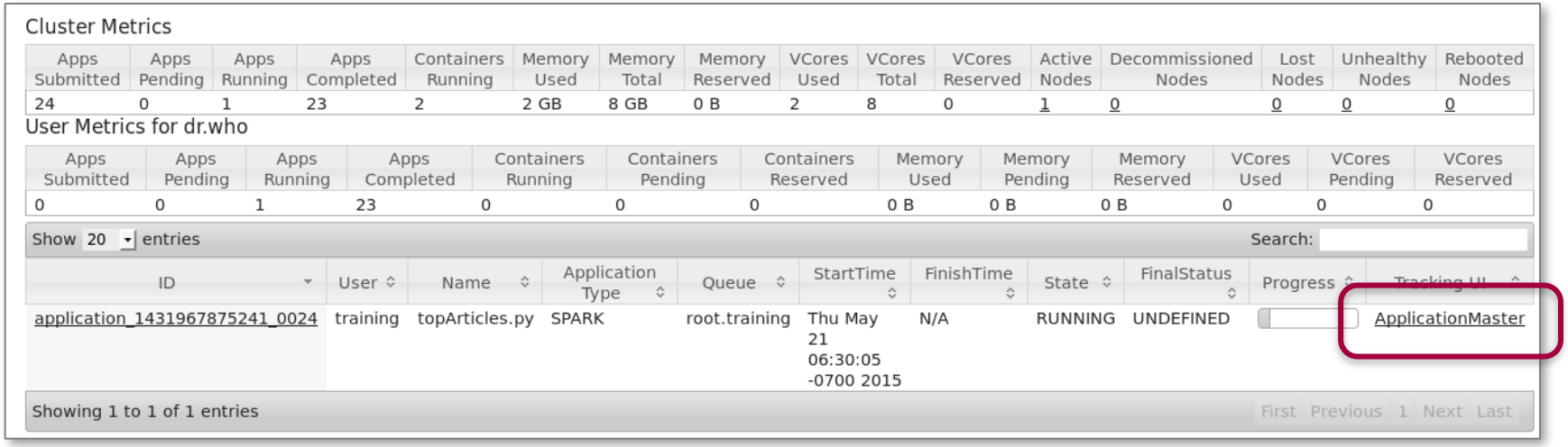
1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

The Spark UI lets you monitor running jobs, and view statistics and configuration





- **The Web UI is run by the Spark drivers**
  - When running locally: **`http://localhost:4040`**
  - When running on a cluster, access via the cluster UI, e.g. YARN UI



Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebooted Nodes
24	0	1	23	2	2 GB	8 GB	0 B	2	8	0	1	0	0	0	0

User Metrics for dr.who

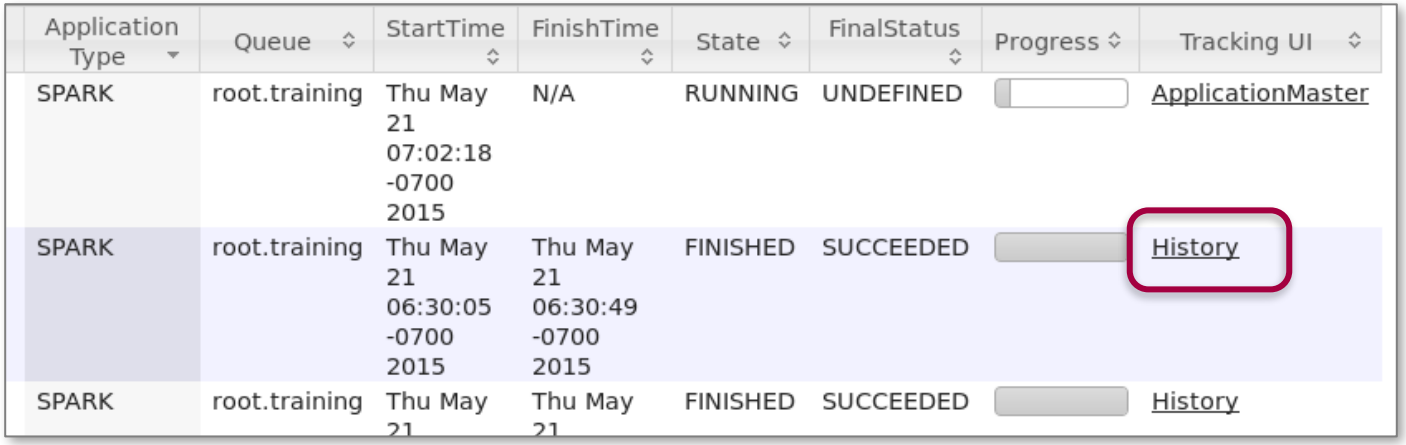
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserved
0	0	1	23	0	0	0	0 B	0 B	0 B	0	0	0

Show 20 entries Search:

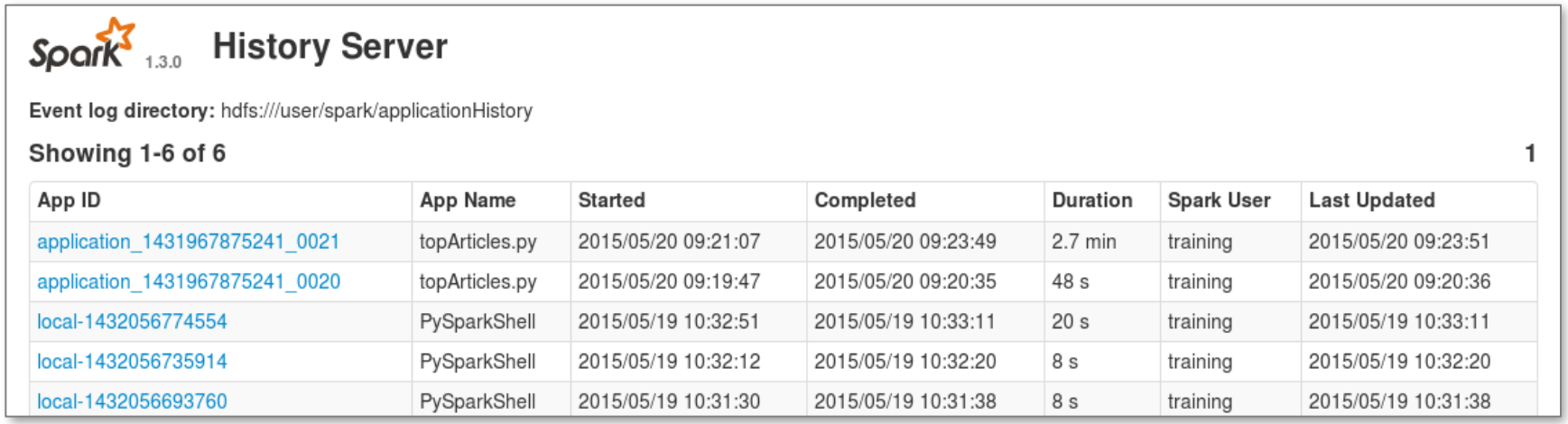
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1431967875241_0024	training	topArticles.py	SPARK	root.training	Thu May 21 06:30:05 -0700 2015	N/A	RUNNING	UNDEFINED		ApplicationMaster

Showing 1 to 1 of 1 entries First Previous 1 Next Last

- **Viewing Spark Job History**
  - Spark UI is only available while the application is running
  - Use Spark History Server to view metrics for a completed application
    - Optional Spark component
- **Accessing the History Server**
  - For local jobs, access by URL
    - E.g. `localhost:18080`
  - For YARN Jobs, click History link in YARN UI



- **Spark History Server**



Spark 1.3.0 History Server

Event log directory: hdfs:///user/spark/applicationHistory

Showing 1-6 of 6 1

App ID	App Name	Started	Completed	Duration	Spark User	Last Updated
application_1431967875241_0021	topArticles.py	2015/05/20 09:21:07	2015/05/20 09:23:49	2.7 min	training	2015/05/20 09:23:51
application_1431967875241_0020	topArticles.py	2015/05/20 09:19:47	2015/05/20 09:20:35	48 s	training	2015/05/20 09:20:36
local-1432056774554	PySparkShell	2015/05/19 10:32:51	2015/05/19 10:33:11	20 s	training	2015/05/19 10:33:11
local-1432056735914	PySparkShell	2015/05/19 10:32:12	2015/05/19 10:32:20	8 s	training	2015/05/19 10:32:20
local-1432056693760	PySparkShell	2015/05/19 10:31:30	2015/05/19 10:31:38	8 s	training	2015/05/19 10:31:38

Agenda

1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

- **Spark provides numerous properties for configuring your application**
- **Some example properties**
  - **`spark.master`**
  - **`spark.app.name`**
  - **`spark.local.dir`** – where to store local files such as shuffle output (default **`/tmp`**)
  - **`spark.ui.port`** – port to run the Spark Application UI (default **`4040`**)
  - **`spark.executor.memory`** – how much memory to allocate to each Executor (default **`512m`**)
  - And many more...
    - See Spark Configuration page for more details

- **Spark Applications can be configured**
  - Declaratively *or*
  - Programmatically

- **`spark-submit` inlined parameter**
  - e.g., **`spark-submit --driver-memory 500M`**
- **Properties file**
  - Tab- or space-separated list of properties and values
  - Load with **`spark-submit --properties-file`** ***`filename`***
  - Example:

```
spark.master      spark://masternode:7077
spark.local.dir   /tmp
spark.ui.port     4444
```

- **Site defaults properties file**
  - **`$SPARK_HOME/conf/spark-defaults.conf`**
  - Template file provided

- Spark configuration settings are part of the SparkContext
- Configure using a SparkConf object
- Some example functions
  - `setAppName(`*`name`*`)`
  - `setMaster(`*`master`*`)`
  - `set(`*`property-name, value`*`)`
- `set` functions return a SparkConf object to support chaining

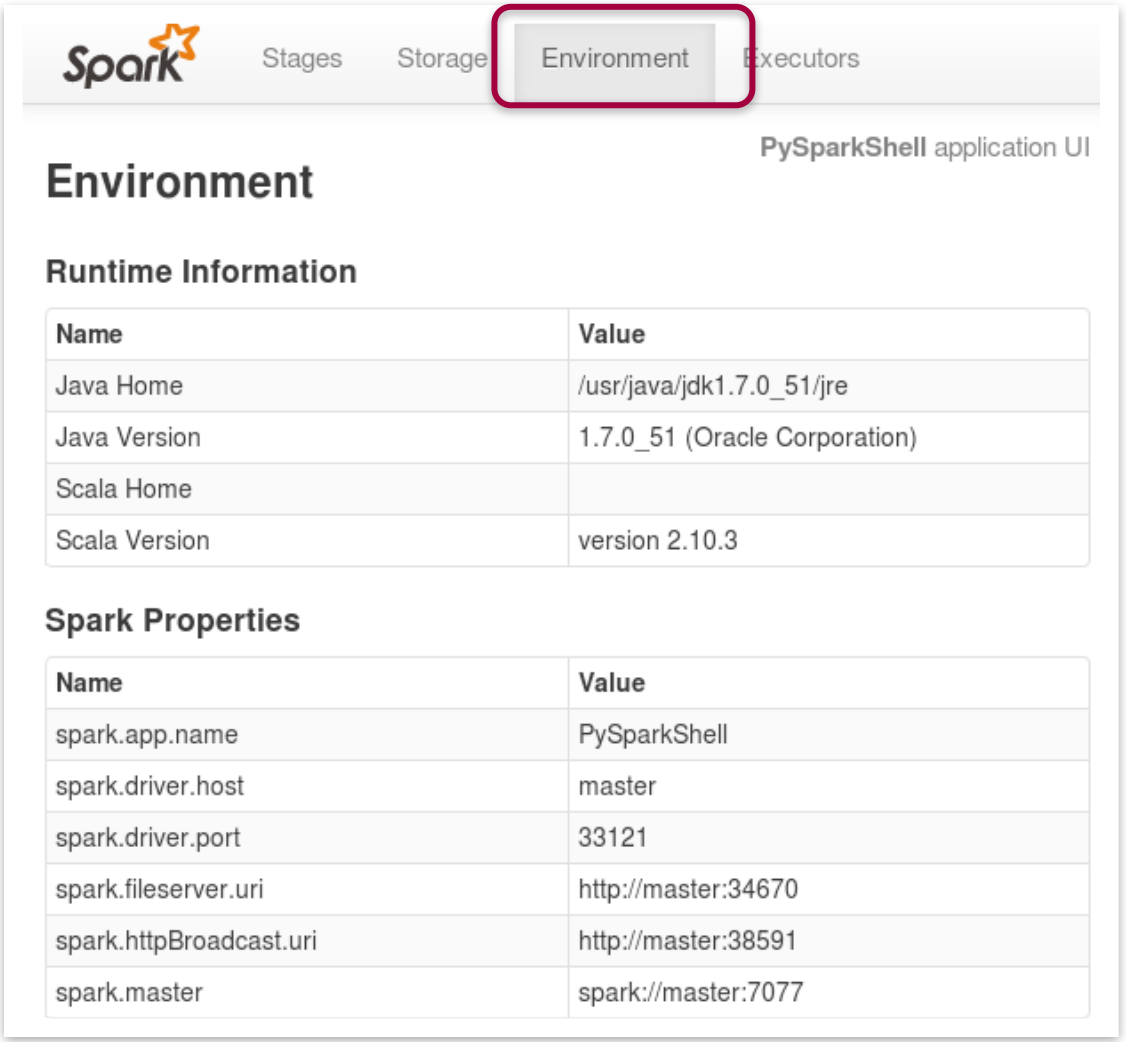
```
import org.apache.spark.SparkContext
import org.apache.spark.SparkContext._
import org.apache.spark.SparkConf

object WordCount {
  def main(args: Array[String]) {
    if (args.length < 1) {
      System.err.println("Usage: WordCount <file>")
      System.exit(1)
    }

    val sconf = new SparkConf()
      .setAppName("Word Count")
      .set("spark.ui.port","4141")
    val sc = new SparkContext(sconf)

    val counts = sc.textFile(args(0)).
       flatMap(line => line.split("\\W")).
       map(word => (word,1)).
       reduceByKey(_ + _)
    counts.take(5).foreach(println)
  }
}
```

- **You can view the Spark property setting in the Spark Application UI**



Agenda

1. Review
2. Spark Applications vs. Spark Shell
3. Creating the SparkContext
4. Building a Spark Application (Scala and Java)
5. Running a Spark Application
6. The Spark Application Web UI
7. Configuring Spark Properties
8. Logging

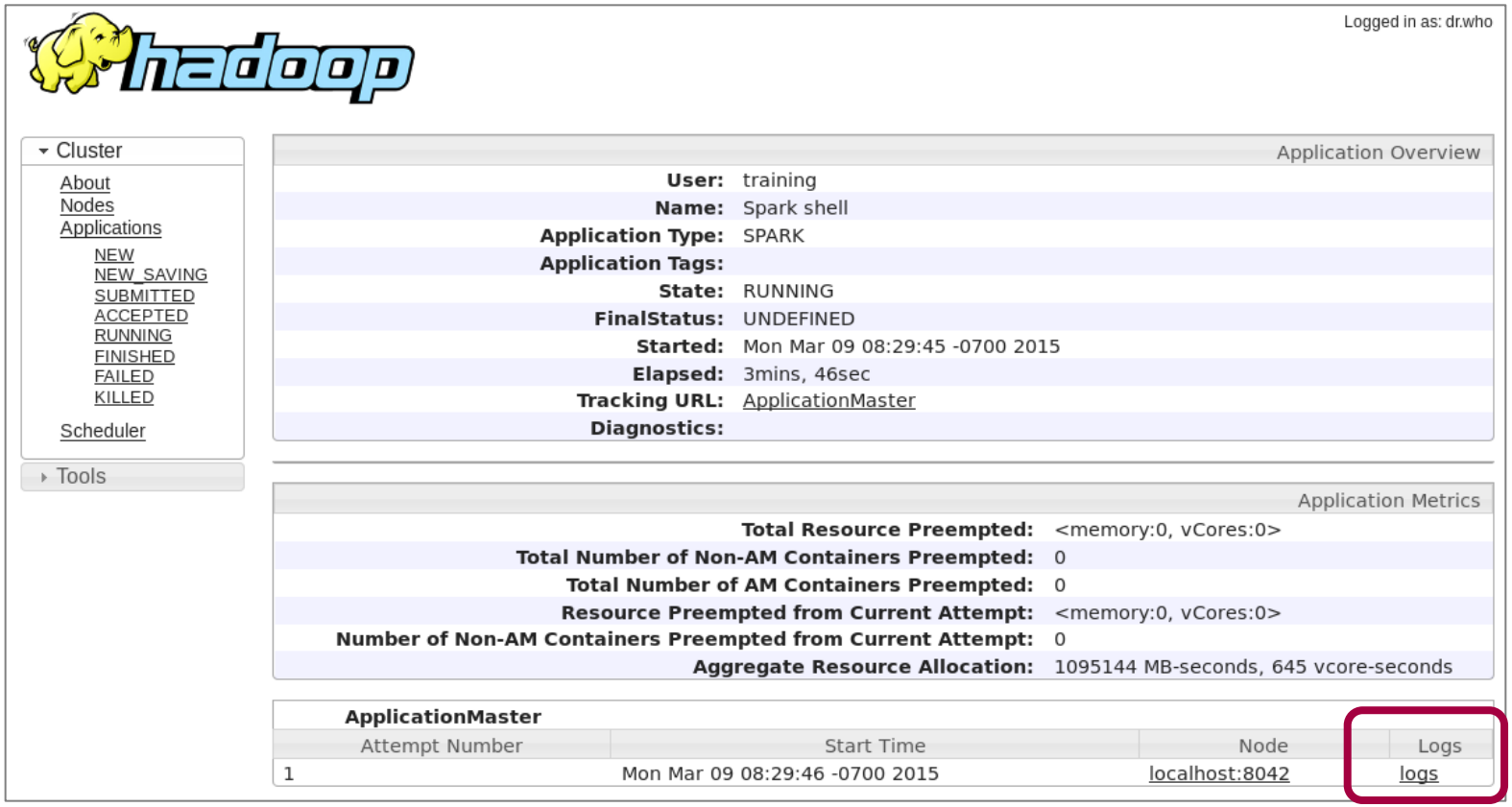
- **Spark uses Apache Log4j for logging**
  - Allows for controlling logging at runtime using a properties file
    - Enable or disable logging, set logging levels, select output destination
  - For more info see `http://logging.apache.org/log4j/1.2/`
- **Log4j provides several logging levels**
  - Fatal
  - Error
  - Warn
  - Info
  - Debug
  - Trace
  - Off

- **Log file locations depend on your cluster management platform**
- **YARN**
  - If log aggregation off, logs are stored locally on each worker node
  - If log aggregation is on, logs are stored in HDFS
    - Default **`/var/log/hadoop-yarn`**
    - Access via **`yarn logs`** command or YARN RM UI

```
$ yarn application -list

Application-Id                  Application-Name Application-Type…
application_1441395433148_0003  Spark shell      SPARK  …
application_1441395433148_0001  myapp.jar        MAPREDUCE  …

$ yarn logs -applicationId <appid>
…
```



- **Logging levels can be set for the cluster, for individual applications, or even for specific components or subsystems**
- **Default for machine: `$SPARK_HOME/conf/log4j.properties`**
  - Start by copying **`log4j.properties.template`**

`log4j.properties.template`

```
# Set everything to be logged to the console
log4j.rootCategory=INFO, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
…
```

- **Spark will use the first `log4j.properties` file it finds in the Java classpath**
- **Spark Shell will read `log4j.properties` from the current directory**
  - Copy **`log4j.properties`** to the working directory and edit

***…my-working-directory*/log4j.properties**

```
# Set everything to be logged to the console
log4j.rootCategory=DEBUG, console
log4j.appender.console=org.apache.log4j.ConsoleAppender
log4j.appender.console.target=System.err
…
```

Homework

See the homework packet for details.