Summary & Highlights

In this module you learned that:

* To connect to the Apache Spark user interface web server, start your application and connect to the application UI using the following URL:  **http://<driver-node>:4040**
* The Spark application UI centralizes critical information, including status information into the **Jobs**, **Stages**, **Storage**, **Environment** and **Executors** tabbed regions. You can quickly identify failures, then drill down to the lowest levels of the application to discover their root causes. If the application runs SQL queries, select the **SQL** tab and the **Description** hyperlink to display the query’s details.​
* The Spark application workflow includes jobs created by the Spark Context in the driver program, jobs in progress running as tasks in the executors, and completed jobs transferring results back to the driver or writing to disk​.
* Common reasons for application failure on a cluster include user code, system and application configurations, missing dependencies, improper resource allocation, and network communications. Application log files, located in the Spark installation directory, will often show the complete details of a failure​.
* User code specific errors include syntax, serialization, data validation. Related errors can happen outside the code If a task fails due to an error, Spark can attempt to rerun tasks for a set number of retries.​ If all attempts to run a task fail, Spark reports an error to the driver and terminates the application. The cause of an application failure can usually be found in the driver event log. ​
* Spark enables configurable memory for executor and driver processes. ​​Executor memory and Storage memory share a region that can be tuned​.
* Setting data persistence by caching data is one technique used to improve application performance​​.
* The following code example illustrates configuration of executor memory on submit for a Spark Standalone cluster: **$ ./bin/spark-submit \​ --class org.apache.spark.examples.SparkPi \​ --master spark://<spark-master-URL>:7077 \​ --executor-memory 10G \​ /path/to/examples.jar \​1000**
* The following code example illustrates setting Spark Standalone worker memory and core parameters: **# Start standalone worker with MAX 10Gb memory, 8 cores​ $ ./sbin/start-worker.sh \​   spark://<spark-master-URL> \​ –-memory 10G –-cores 8**
* Spark assigns processor cores to driver and executor processes during application processing. Executors process tasks in parallel according to the number of cores available or as assigned by the application. ​
* You can apply the argument **‘--executor-cores 8 \’** to set executor cores on submit *per executor*. This example specifies eight cores.
* You can specify the executor cores for a Spark standalone cluster *for the application* using the argument *‘***‘--total-executor-cores 50’** followed by the number of cores for the application. This example specifies 50 cores.
* When starting a worker manually in a Spark standalone cluster, you can specify the number of cores the application uses by using the argument **‘--cores‘** followed by the number of cores. ​Spark’s default behavior is to use all available cores. ​