Slide 6:
There are three ways to monitor Spark applications. The first way is the Web UI. The default port is 4040. The port in the lab environment is 8088. The information on this UI is available for the duration of the application. If you want to see the information after the fact, set the spark.eventLog.enabled to true before starting the application. The information will then be persisted to storage as well.

Metrics is another way to monitor Spark applications. The metric system is based on the Coda Hale Metrics Library. You can customize it so that it reports to a variety of sinks such as CSV. You can configure the metrics system in the metrics.properties file under the conf directory.

Finally, you can also use external instrumentations to monitor Spark. Gangalia is used to view overall cluster utilization and resource bottlenecks. Various OS profiling tools and JVM utilities can also be used for monitoring Spark.

Slide 7:
The Web UI is found on port 4040, by default, and shows the information for the current application while it is running.

The Web UI contains the following information.
A list of scheduler stages and tasks.
A summary of RDD sizes and memory usage.
Environmental information and information about the running executors.

To view the history of an application after it has ran, you can start up the history server. The history server can be configured on the amount of memory allocated for it, the various JVM options, the public address for the server, and a number of properties.

You will see all of this in the lab exercise.

Slide 8:
Spark programs can be bottlenecked by any resource in the cluster. Due to Spark's nature of the in-memory computations, data serialization and memory tuning are two areas that will improve performance. Data serialization is crucial for network performance and to reduce memory use. It is often the first thing you should look at when tuning Spark applications. Spark provides two serialization libraries. Java serialization provides a lot more flexibility, but it is quiet slow and leads to large serialized objects. This is the default library that Spark uses to serialize objects. Kyro serialization is much quicker than Java, but does not support all Serializable types. It would require you to register these types in advance for best performance. To use Kyro serialization, you can set it using the SparkConf object.

With memory tuning, you have to consider three things. The amount of memory used by the objects (whether or not you want the entire object to fit in memory). The cost of accessing those objects and the overhead garbage collection.

You can determine how much memory your dataset requires by creating a RDD, put it into cache, and look at the SparkContext log on your driver program. Examining that log will show you how much memory your dataset uses.

Few tips to reduce the amount of memory used by each object. Try to avoid Java features that add overhead such as pointer based data structures and wrapper objects. If possible go with arrays or primitive types and try to avoid nested structures.

Serialized storage can also help to reduce memory usage. The downside would be that it will take longer to access the object because you have to deserialized it before you can use it.

You can collect statistics on the garbage collection to see how frequently it occurs and the amount of time spent on it. To do so, add the line to the SPARK_JAVA_OPTS environment variable.

Slide 9:
The level of parallelism should be considered in order to fully utilize your cluster. It is automatically set to the file size of the task, but you can configure this through optional parameters such as in the SparkContext.textFile. You can also set the default level in the spark.default.parallelism config property. Generally, it is recommended to set 2-3 tasks per CPU core in the cluster.

Sometimes when your RDD does not fit in memory, you will get an OutOfMemoryError. In cases like this, often by increasing the level of parallelism will resolve this issue. By increasing the level, each set of task input will be smaller, so it can fit in memory.

Using Spark's capability to broadcast large variables greatly reduces the size of the serialized object. A good example would be if you have some type of static lookup table. Consider turning that into a broadcast variable so it does not need to be passed on to each of the worker nodes.

Spark prints the serialized size of each tasks in the master. Check that out to examine if any tasks are too large. If you see some tasks larger than 20KB, it's worth taking a look to see if you can optimize it further, such as creating broadcast variables.

Slide 10:
Having completed this lesson, you should be able to describe the cluster overview. You should also know where and how to set Spark configuration properties. You also saw how to monitor Spark using the UI, metrics or various external tools. Finally, you should understand some performance tuning considerations.

Slide 11:
Next steps. Complete lab exercise #5 and then, congratulations, you have completed this course.