**Fast Track Apache Spark**

Use these lessons learned to get a quick start on productivity

My Strata NYC 2017 talk is based on research done under the limited funding conditions of academia. This meant that I did not have an infrastructure team, therefore I had to set up my Spark environment myself. In the process I learned a lot of lessons. I want to help you avoid making the mistakes I did so you can start making immediate impact with Apache Spark. Here are the six lessons I learned.

1. You don’t need a database or data warehouse. It is common for Spark setups to use Apache Hadoop’s file system and Hive for querying, but you can use text files and other accepted file formats directly from local directories if in the short term you don’t want to go through the hassle of setting up a database or warehouse.

2. You don’t need a cluster of machines. You can hit the ground running using your local machine or a single server. This is also helpful in that you do not have to consider what cluster manager to install - YARN or Mesos. You can simply use the stand alone cluster manager that comes with Spark. Just make sure that if you use one machine it should have multiple cores and enough memory to cache your data.

3. Don’t know scala? Start learning Spark in the language you do know- whether it be java, python, or R. In versions 2.0+ a lot of additional support was added for R, namely in the form of SparkR and sparklyr.

4. Use a notebook. Don’t bother trying to configure an IDE or using the shell to write applications. Of course, using the shell is great if you are submitting an application or doing some basic coding. However, the pain of configuring an IDE with Spark is not worth the investment. Any programming I did in Spark was either in the shell or a Jupyter notebook. The latter being used the majority of the time.

5. Use DataFrames instead of RDDs (Resilient Distributed Datasets) for ease of use. There has been extensive work done between versions 1.6 and 2.0 to make functionality traditionally done using RDDs easier to do with DataFrames. DataFrames are also efficient across languages. So if you’re more comfortable with java, python, or R there is no performance loss suffered by not switching to scala.

6. Avoid partial actions. A partial action, is an action that does not allow the DAG (Directed Acyclic Graph) for a Spark job to be fully evaluated. Thus transformations along the way do not complete. For example, one issue whose solution eluded me for some time was why computation was slow on data I cached. Eventually I realized by looking at the web UI that only some of the data was cached. However, I still did not know why all the data was not being cached. In fact, I knew that since cache() was a transformation, I had to call an action before it would be evaluated. The code I was running was data=df.cache().show(). However, because show() is defaulted to 20, and my data had more than 20 rows, show() was a partial action. I should have been running data=df.cache().count(), as count() will always act as a full action. So be wary of partial actions. Thanks to [Holden Karau](https://www.amazon.com/Holden-Karau/e/B00G4JS6IO/ref=ntt_dp_epwbk_0) for pointing this one out to me.

These tips highlight Spark’s ability to deliver serious gains in productivity despite limited user computing capability. There is definitely an ideal set up for each organization’s particular needs. As such, one or all of the following: expanding to a cluster setup, building a data warehouse, and utilizing an infrastructure team might be necessary at some point. My aim here is simply to make it easier to create a proof of concept that justifies stakeholder investment and/or provide some pointers if you decide that a bare bones set up is best for you.