## Scaling Out

In the last lab, we "scaled up" in that we launched a more powerful server to analyze more data. We also "scaled out" in that we took advantage of parallel processing. Today we are going to take that a step further, scaling out not just to multiple engines (cores) but to multiple nodes (instances). While this is possible to do with IPython parallel, for today's lab we will be using the far more robust and well-known technology: Hadoop.

In order to go forward, we must first take a step backwards. Unlike IPython parallel, which can be leveraged interactively through a Jupyter notebook, Hadoop only works in the form of batch processes. So for our first foray into Hadoop, we will need to wrap up our processes into a map step and a reduce step. The map step will roughly correspond to what we parallelized in Sparkify 4, and the reduce step to how we combined the results at the end. Hadoop is a bit more constrained, however, in how these messages are passed around. Everything must be in the form of key-value pairs. This need not be overly complicated, however. Take a look at how word count is done for inspiration.

First, we will need a Hadoop cluster. The easiest way to get this is through Amazon [Elastic MapReduce](https://console.aws.amazon.com/elasticmapreduce) (EMR). Go through the steps to create a cluster. For now, you can leave the configuration as default:

* Launch mode: Cluster
* Vender: Amazon
* Applications: Core Hadoop
* Instance type: m3.xlarge
* Number of instances: 3 Choose your EC2 key pair and click "Create cluster".

While you are waiting for your cluster to start, you can get started developing your map and reduce scripts. Start by downloading one of the files from your S3 bucket onto your own computer. Next, write your mapper script. One of the things you'll notice if you look at [various](http://www.michael-noll.com/tutorials/writing-an-hadoop-mapreduce-program-in-python/) [examples](http://www.glennklockwood.com/data-intensive/hadoop/streaming.html) of writing Hadoop Streaming jobs in Python is that the scripts always start with the following three lines:

#!/usr/bin/env python

import sys

for line in sys.stdin:

The first line tells the shell that this is a Python script (and not in some other language like Perl). The second line tells Python to import the sys library. (Nothing special about this.) The third line reveals something about how Hadoop Streaming works. Essentially, it streams (hence the name) data into your mapper job through the [standard input](https://en.wikipedia.org/wiki/Standard_streams#Standard_input_.28stdin.29). Similarly, it expects results to come through [stdout](https://en.wikipedia.org/wiki/Standard_streams#Standard_output_.28stdout.29) (or print which is basically a shorthand for sys.stdout.write with a '\n' automatically appended to the end).

Make your scripts executable (i.e. [chmod a+x \*.py](https://en.wikipedia.org/wiki/Chmod)) and test them using the following pattern:

cat <sparkify-stream-data> | ./mapper.py | sort | ./reducer.py

The sort takes the place of Hadoop's Shuffle/Sort (only without the shuffle part since we are still on a single node).

By now, your EMR cluster should be up and running. ssh into the master node and repeat the above process (download a single file from your S3 bucket and test your mapper and reducer scripts using cat and sort). This will help you make sure that your scripts will in fact run on the EC2 machines. If you run into dependency problems, rather than install the required packages, see if you can refactor your code so it no longer has that dependency. (i.e. do you really need Pandas for this?)

Once you've tested it, it's time for the moment of truth. Upload your mapper and reducer scripts to S3 (but not the bucket where your kinesis files live) and follow the instructions to [Submit a Streaming Step](http://docs.aws.amazon.com/emr/latest/ReleaseGuide/CLI_CreateStreaming.html). One thing you will need to watch out for: though from S3's perspective, everything that comes after the bucket name is the key name or the prefix of a set of key names; from Hadoop's perspective (or, more accurately, HDFS's perspective), it must be treated as a normal directory structure. So if you want to reference all the files from every hour of every day of November 2018, you will need to put, as you input s3://[your-bucket-name]/2018/11/\*/\*/\* (in other words, s3://[your-bucket-name]/2018/\* won't work. It will complain that 11 is not a not a file). You will also need to provide the name of a new "directory" in which to place the output. That can be a subdirectory of an existing bucket, but it cannot already exist.

A couple things to be aware of:

1. Your EMR cluster and your bucket need to be in the same AWS region. This will not be a problem if you've been doing everything in us-east-1 (a.k.a. N. Virginia).
2. MapReduce generates one part\_ file for each reducer. When you specify an output location, you are specifying a folder not a file. The folder will contain the files that contain your results.

#### On completing this assignment, you should have submitted:

* An executable mapper (.py) that uses stdin and print (or stdout)
* An executable reducer (.py) that uses stdin and print (or stdout)