Week 5 Lab

This week's work outside of class is optional, but highly recommended. We will walk through installing Jupyter as well as make some sandbox modifications to support running Jupyter as well as Zeppelin and the Spark Web UI and I've also provided some additional fun Spark exercises for those of you who are especially interested in the software part of our class.

Docker Port Forwarding Modifications

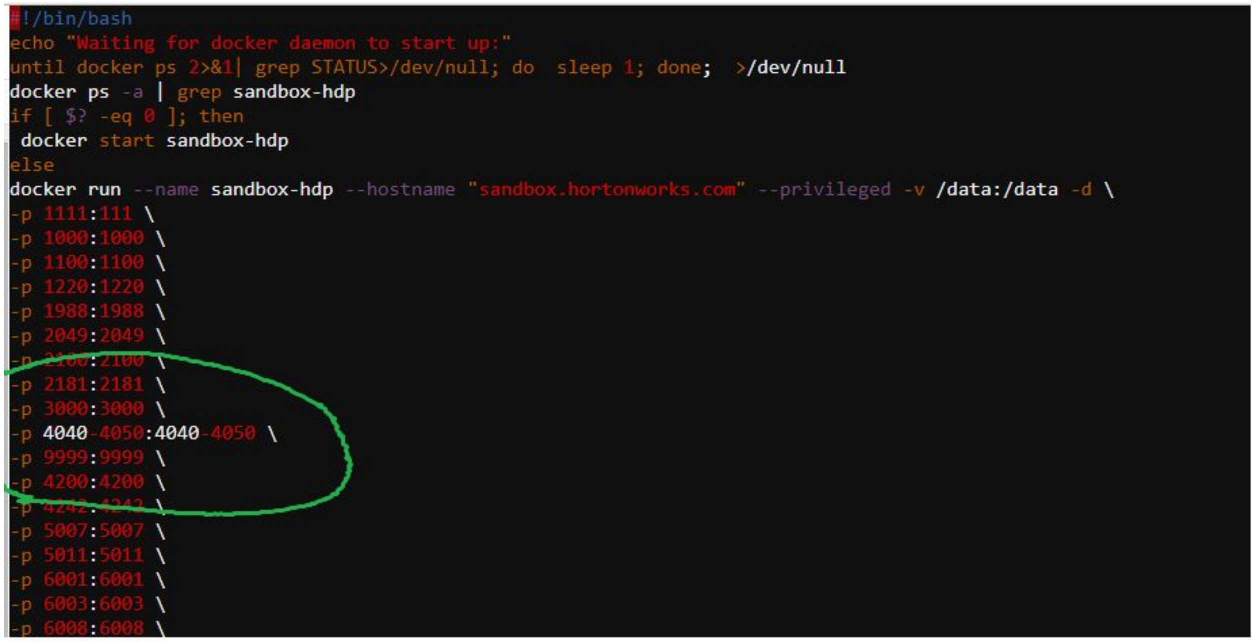
In this step we are going to go through process we talked about in class this week about adding additional port forwards to your sandbox-hdp Docker container. Remember that we can only set up these proxies when the container is created, so we need to commit an image of our current setup and recreate a new container with the new mappings.

SSH into your VM:

- Stop sandbox-hdp Docker container
    - `sudo docker stop sandbox-hdp`
- Confirm not running using Docker 'ps"
    - `sudo docker ps`
- Crate image of the current container state
    - `sudo docker commit sandbox-hdp sandbox-hdp`
        - **Note**: Normally would want to use a new name, but we are short on space and script is hardcoded to use sandbox-hdp
        - This will take a while since it's effectively making a copy of your container
- Edit start-sandbox.sh to add additional port forwardings
    - When you edit the file you will see list of lines with the format "-p PORT:PORT \"
    - Add the following to that list (order doesn't matter)

        `-p 9999:9999 \` (for jupyter)

        `-p 4040-4050:4040-4050 \` (for spark web ui)



```
#!/bin/bash
echo "Waiting for docker daemon to start up:"
until docker ps 2>&1| grep STATUS>/dev/null; do  sleep 1; done;  >/dev/null
docker ps -a | grep sandbox-hdp
if [ $? -eq 0 ]; then
 docker start sandbox-hdp
else
docker run --name sandbox-hdp --hostname "sandbox.hortonworks.com" --privileged -v /data:/data -d \
-p 1111:111 \
-p 1000:1000 \
-p 1100:1100 \
-p 1220:1220 \
-p 1988:1988 \
-p 2049:2049 \
-p 2100:2100 \
-p 2181:2181 \
-p 3000:3000 \
-p 4040-4050:4040-4050 \
-p 9999:9999 \
-p 4200:4200 \
-p 4242:4242 \
-p 5007:5007 \
-p 5011:5011 \
-p 6001:6001 \
-p 6003:6003 \
-p 6008:6008 \
```

- Remove old container
  `sudo docker rm sandbox-hdp`
- Create new container and start sandbox
  `sudo start-sandbox-hdp.sh`

Log into Azure Web Portal:

- Add Port Forwarding Rules on the VM Networking page (same as we did to allow access to Ambari Web UI during initial setup
  - Jupyter
    - 9999
  - Support up to 10 simulataneous Spark applications
    - 4040-4050
  - Zeppelin (if interested, if not leave port firewalled
    -
- Assuming sandbox is running can verify port mapping by going to ip:4040 and should see a Spark UI

Install Jupyter

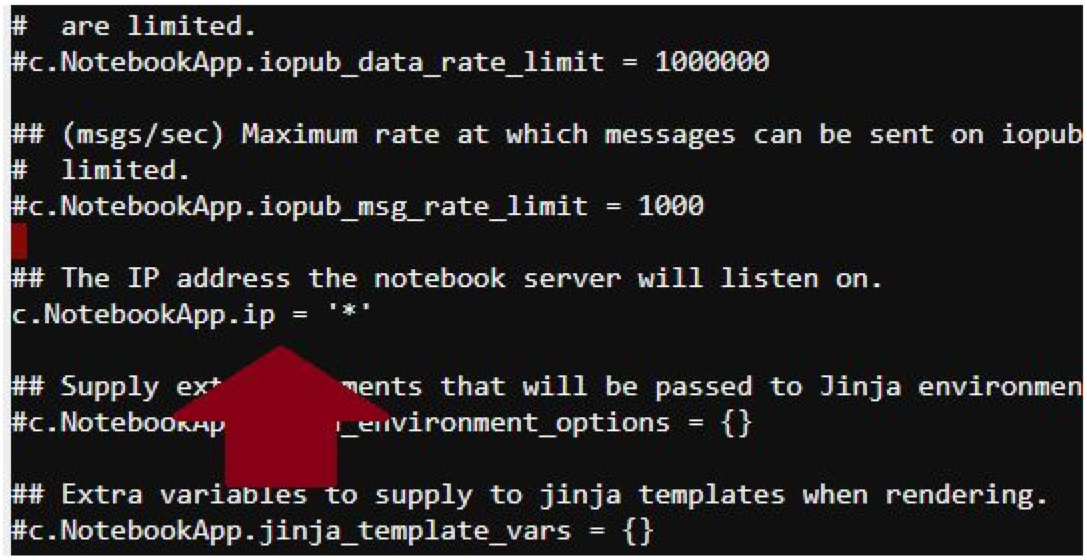
Normally we do NOT install Jupyter as root, but for this sandbox it is the most straightforward way. We are going to use a Python virtual environment manager environment called Anaconda that besides including Jupyter also installs many useful data anlysis libraries as well in one package.

SSH into sandbox (from VM)

- Download anaconda installation script
  - `wget https://repo.continuum.io/archive/Anaconda3-5.0.1-Linux-x86_64.sh`
- Run installation script
  - `bash Anaconda3-5.0.1-Linux-x86_64.sh`
- Follow prompts
  - Accept license
  - Choose to install in "`/opt/anaconda3`" when prompted for path
- A bunch of packages will be downloaded and installed
- At the end will be prompted to add to PATH
  - Make sure to select "no" (default) to add to prompt
- Anaconda + Jupyter is now installed
- Generate a config file and update to allow access vi VM
  - `/opt/anaconda3/bin/jupyter notebook --generate-config --allow-root`

- Edit `/root/.jupyter/jupyter_notebook_config.py` & the following changes
    - Remove the comment at the beginning of the line and make the updates in the text

```
c.NotebookApp.ip = '*'
c.NotebookApp.open_browser = False
c.NotebookApp.port = 9999
```



```
#  are limited.
#c.NotebookApp.iopub_data_rate_limit = 1000000

## (msgs/sec) Maximum rate at which messages can be sent on iopub
#  limited.
#c.NotebookApp.iopub_msg_rate_limit = 1000

## The IP address the notebook server will listen on.
c.NotebookApp.ip = '*'

## Supply ext      ments that will be passed to Jinja environmen
#c.NotebookAp      _environment_options = {}

## Extra variables to supply to jinja templates when rendering.
#c.NotebookApp.jinja_template_vars = {}
```

If you are only going to use Python are done installing things

To start Jupyter using just plain Python (no Spark)

- `/opt/anaconda3/bin/jupyter notebook --allow-root`
- Copy "token" string, paste into browser and replace localhost with your IP and should see Jupyter home screen
    - Should look something like this
    - `http://localhost:9999/?token=810134bc844491a627de49ced8805a96c85edacb88151137`
    -

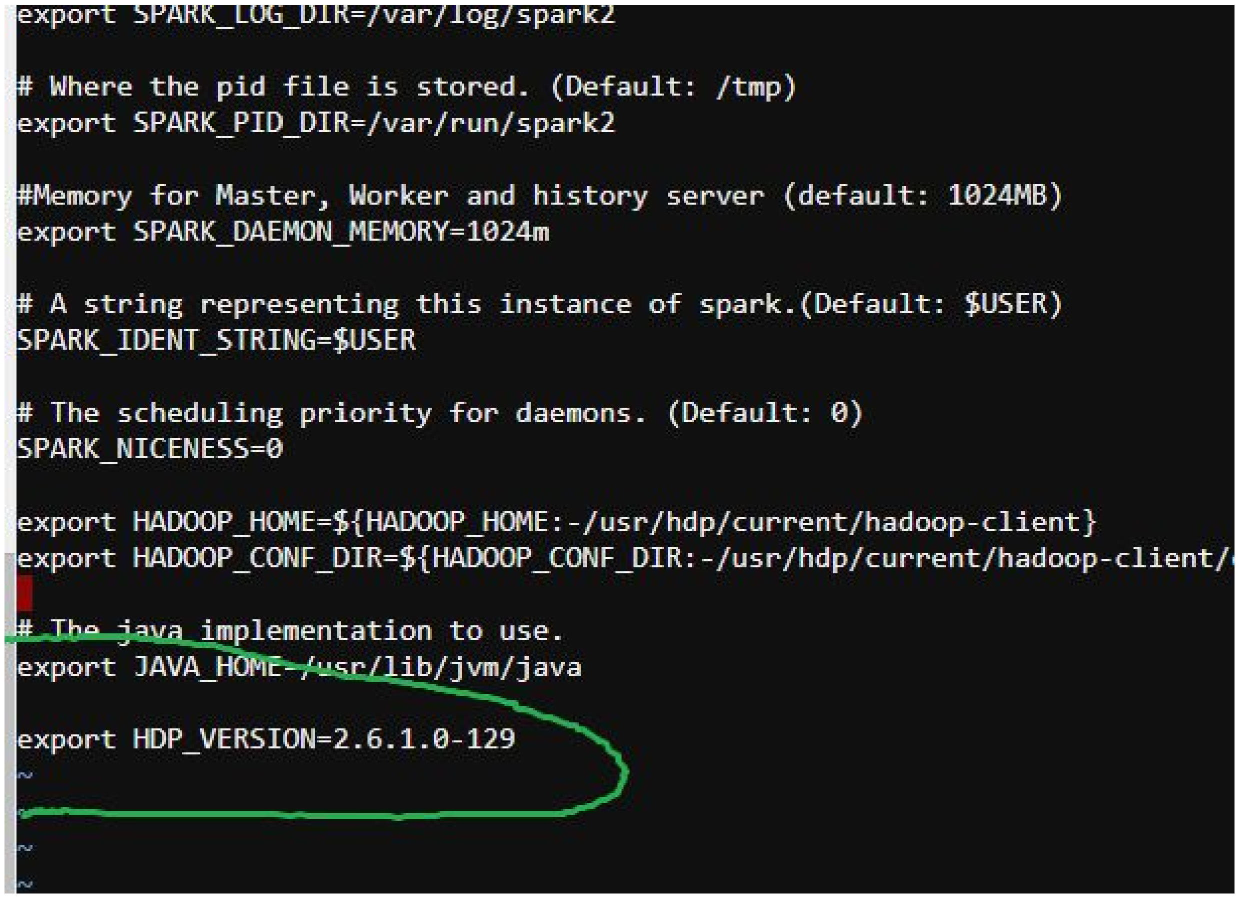
To use pyspark in Jupyter

- Need to tell pyspark to use Jupyter for python environment
- "`PYSPARK_DRIVER_PYTHON=/opt/anaconda3/bin/jupyter PYSPARK_DRIVER_PYTHON_OPTS='notebook --allow-root' pyspark`"
    - This command is one line
    - Alternatively could set these env vars to export when log in to avoid typing every time (if you know what that means)

If using Scala need to install a kernel for Spark:

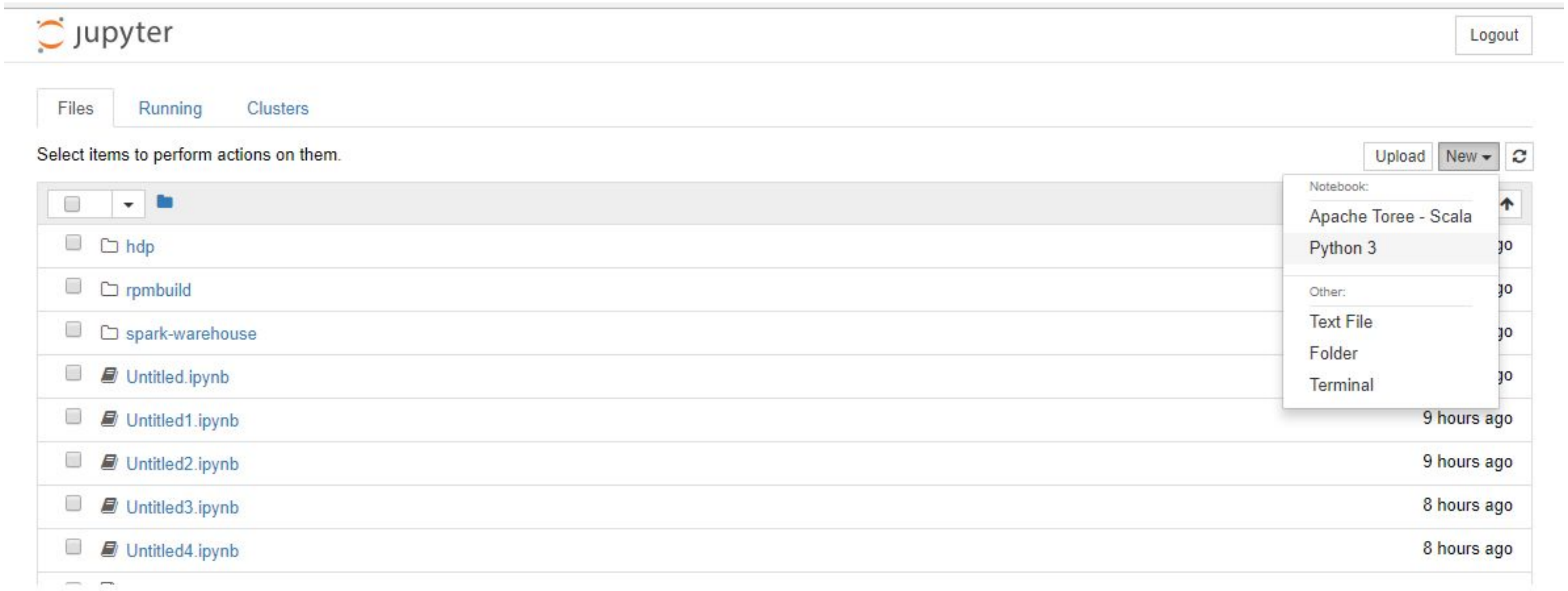
- Install Apache Toree
    - SSH into sandbox (if not already there)
    - `/opt/anaconda3/bin/pip install https://dist.apache.org/repos/dist/dev/incubator/toree/0.2.0/snapshots/dev1/toree-pip/toree-0.2.0.dev1.tar.gz`
        - Again, one long command
    - `/opt/anaconda3/bin/jupyter toree install --spark_home=/usr/hdp/current/spark2-client/`

- Update spark-env.sh to include correct HDP version
    - This is a sandbox finicky thing that only needs to be done for this environment
    - Edit /usr/hdp/current/spark2-client/conf/spark-env.sh
    - Add an additional “export” line like the following
        - `export HDP_VERSION=2.6.1.0-129`



```
export SPARK_LOG_DIR=/var/log/spark2

# Where the pid file is stored. (Default: /tmp)
export SPARK_PID_DIR=/var/run/spark2

#Memory for Master, Worker and history server (default: 1024MB)
export SPARK_DAEMON_MEMORY=1024m

# A string representing this instance of spark.(Default: $USER)
SPARK_IDENT_STRING=$USER

# The scheduling priority for daemons. (Default: 0)
SPARK_NICENESS=0

export HADOOP_HOME=${HADOOP_HOME:-/usr/hdp/current/hadoop-client}
export HADOOP_CONF_DIR=${HADOOP_CONF_DIR:-/usr/hdp/current/hadoop-client/

# The java implementation to use.
export JAVA_HOME=/usr/lib/jvm/java

export HDP_VERSION=2.6.1.0-129
~
~
~
~
```

Can start Jupyter with command "`/opt/anaconda3/bin/jupyter notebook`" and then select "Apache Toree - Scala" to start a Spark scala kernel



Select items to perform actions on them.

hdp	
rpmbuild	
spark-warehouse	
Untitled.ipynb	
Untitled1.ipynb	9 hours ago
Untitled2.ipynb	9 hours ago
Untitled3.ipynb	8 hours ago
Untitled4.ipynb	8 hours ago

Assignment 5

This is a course about big data, right? So far we've only looked at data that was a few MB. Let's see if anything changes when we move up.

I've made available a dataset of loan data from LendingTree from 2007-2015. The data is a CSV file with approximately 800K rows and 75 columns, total size a little over 400MB.

Download the sample data from here:

https://s3-us-west-2.amazonaws.com/bigdata210/loan.csv

There is also a data dictionary that explains the columns in excel format here:

https://s3-us-west-2.amazonaws.com/bigdata210/LCDataDictionary.xlsx

Use this data for the following questions:

1) Create a "histogram" of the counts for each type defined in the "home_ownership" column.
2) This a very wide data set, and we frequently will only be querying a subset of the columns
    a) Currently it is in CSV format. Is there a more appropriate file format? If so, then explain why.
    b) Convert the CSV data to the format you specified in part a) by saving the file to that format, and experiment with queries to show the difference in performance.
3) For each category of "home_ownership", what is the count of each "loan_status"?
4) Did any of these loans originate in King county based on the data we have in wa_zipcodes.txt? (the file is misleadingly named... it's not all zipcodes in WA but only King county zip codes"). Note: since the data only has the first 3 of each zip, as long as the first 3 match then count it.