HADOOP CLUSTER SETUP GUIDE:

Passwordless SSH Sessions:

Before we start our installation, we have to ensure that passwordless SSH Login is possible to any of the Linux machines of CS120. In order to do so, log in to your account and run the following commands:

For Further understanding, look up:
https://hadoopabcd.wordpress.com/2015/05/16/linux-password-less-ssh-logins/

```
ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
cat $HOME/.ssh/id_rsa.pub >> $HOME/.ssh/authorized_keys
```

Setting up Config Files:

Step 1. Create a new folder for your configuration files:

```
mkdir ~/hadoopConf
```

Step 2. Create **core-site.xml**, **hdfs-site.xml**, **mapred-site.xml**, and **yarn-site.xml**. Click on the link below to download the sample core-site.xml, hdfs-site.xml, mapred-site.xml, and yarn-site.xml:

```
wget http://www.cs.colostate.edu/~cs535/configuration/PA1/conf.tar
tar xvf conf.tar
```

Copy configuration files into your configuration directory:

```
mv * ~/hadoopConf
```

For the files core-site.xml, mapred-site.xml and hdfs-site.xml, anything with "HOST" or "PORT" in the config files should be replaced with actual hostnames and port numbers of your choice. The list of hostnames in CSB120 is available at:

http://www.cs.colostate.edu/~info/machines

Make sure that you should select at least 10 nodes (CS120 machines only) for your slaves file that is detailed below in Step 4.

You need to select a set of available ports from the non-privileged port range. Do not select any ports between 56,000 and 57,000. These are dedicated for the shared HDFS cluster. To reduce possible port conflicts, you will be each assigned a port range.

Step 3. Set the environment variables in your .bashrc file

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR= <path to your config directory mentioned in Step 2>
export YARN_CONF_DIR=${HADOOP_CONF_DIR}
export HADOOP_LOG_DIR= /tmp/${USER}/hadoop-logs
export YARN_LOG_DIR= /tmp/${USER}/yarn-logs
export JAVA_HOME=/usr/local/jdk1.8.0_51
export HADOOP_OPTS="-Dhadoop.tmp.dir=/s/${HOSTNAME}/a/tmp/hadoop-${USER}"
```

Step 4. Create **masters** and **slaves** file

Masters file should contain the name of the machine you choose as your secondary namenode. Slaves contains the list of all your worker nodes. Programming assignment 1 requires at least 10 such nodes.

Running HDFS

a. **First Time Configuration**

Step 1. Log into your namenode

You will have specified your namenode in core-site.xml.

```
<property>
  <name>fs.default.name</name>
  <value>hdfs://host:port</value>
</property>
```

Step 2. Format your namenode.

```
$HADOOP_HOME/bin/hdfs namenode -format
```

b. **Starting HDFS**

Run the start script:

```
$HADOOP_HOME/sbin/start-dfs.sh
```

Step 3. Check the web portal to make sure that HDFS is running, open on your browser the url **http://<namenode>:<port given>** for the property **dfs.namenode.http-address** in hdfs-site.xml

c. **Stopping HDFS**

Step 1. Log into your namenode
Step 2. Run the stop script:

```
$HADOOP_HOME/sbin/stop-dfs.sh
```

Running Yarn

a**. Starting Yarn**

Step 1. Log into your resource manager
(Resource Manager is the node you set as HOST in the yarn-site.xml)

Step 2. Run the start script

```
$HADOOP_HOME/sbin/start-yarn.sh
```

Step 3. Check the web portal to make sure yarn is running
http://<resourcemanager>:<port given for the property yarn.resourcemanager.webapp.address>

b. **Stoping Yarn**

Step 1. Log into your resource manager
Step 2. Run the stop script

```
$HADOOP_HOME/sbin/stop-yarn.sh
```

Running a Job Locally

a. **Setup**

Step 1. Open mapred-site.xml

Step 2. Change the value of property **mapreduce.framework.name** to

local. To be safe, restart HDFS and ensure yarn is stopped

b. **Running the Job From any node:**

```
$HADOOP_HOME/bin/hadoop jar your.jar yourClass args
```

Running a Job in Yarn

a. **Setup**

Step 1. Open mapred-site.xml

Step 2. Change the value of property **mapreduce.framework.name** to **yarn**

To be safe, restart HDFS and yarn

b. **Running the Job From any node:**

```
$HADOOP_HOME/bin/hadoop jar your.jar yourClass args
```

Download and configure Apache Spark

1. Download Spark release **1.6.1, Pre-built for Hadoop 2.6** using following command

```
wget http://d3kbcqa49mib13.cloudfront.net/spark-1.6.1-bin-hadoop2.6.tgz
```

   This will download Gzip archive with .tgz extension named **spark-1.6.1-bin-hadoop2.6.tgz**.

2. **Unzip** downloaded archive to your **home** directory.
   - Change directory (**cd**) to folder containing downloaded archive.
   - Run following command to **unzip** archive and **copy** to your **home** directory.

```
tar -xvzf spark-1.6.1-bin-hadoop2.6.tgz -C ~
```

3. Set **SPARK_HOME** environment variable to point above unzipped folder using following method.
   - Set in *.bashrc* file
     - Open *.bashrc* file, which will be inside your **home** directory.

```
gedit ~/.bashrc
```

     - Add following line, if it doesn't exist.

```
export SPARK_HOME=${HOME}/spark-1.6.1-bin-hadoop2.6
```

     - Reflect changes made in *.bashrc* file
       - Close *.bashrc* file and reflect changes using following command.

```
source ~/.bashrc
```

       OR

       - Close all terminals and reopen them.

     - Check **SPARK_HOME** environment variable value using following command. It should be **unzipped**(Spark) folder location.

```
echo $SPARK_HOME
```

Placeholders are denoted with <> in this document. You need to replace those (including <>) with appropriate values.

4. Changes to configure Spark need to be done in *conf* folder, which is inside unzipped folder. Templates for every configuration files are already given in this folder. We just need to make changes in required templates and save them with correct names.

    - **slaves**
        - This file stores the machines list/worker nodes (each machine name on new line) your Spark environment use to execute code.
        - Save ***slaves.template*** as ***slaves.***
        - Remove *localhost* from the list and copy all the slave machines used in Hadoop (**${HADOOP_CONF_DIR}/slaves**) and paste in ***slaves.***
            - This will maximize the benefit of data locality as same nodes are working as datanodes(HDFS) and workernodes(Spark).
        - Save the ***slaves*** file.

    - **spark-env.sh**
        - This file stores primary configuration options for Spark environment.
        - Spark provides several options for deployment, but we are configuring for **Spark Standalone Deploy Mode**. For more information, refer http://spark.apache.org/docs/latest/.
        - Save ***spark-env.sh.template*** as ***spark-env.sh***.
        - Update following options (search for **standalone**) with appropriate values in ***spark-env.sh*** and don't forget to **export** them. You can refer sample below this table.

Environment variable	Description
SPARK_MASTER_IP	Bind the master to a specific **hostname** or **IP address**.
SPARK_MASTER_PORT	Start the master on a different port (default: 7077). You need to select a set of available ports from the **non-privileged port range**. Do not select any ports between **56,000 and 57,000**. These are dedicated for the shared HDFS cluster. To reduce possible port conflicts, you will be each **assigned a port range**.
SPARK_MASTER_WEBUI_PORT	Port for the **master web UI** (default: 8080). Follow port selection instructions mentioned in **SPARK_MASTER_PORT.**

SPARK_WORKER_INSTANCES	To set the number of **worker processes per node** (e.g. 2, 4)
SPARK_WORKER_CORES	Total **number of cores** to allow Spark applications to use **per worker instance** (e.g. 1, 2) (default: all available cores).
SPARK_WORKER_MEMORY	Total **amount of memory** to allow Spark applications to use **per worker instance**, e.g. 1000m, 2g (default: total memory minus 1 GB).

- Sample environment variables are given below. Put appropriate values in placeholders.

```
export SPARK_MASTER_IP=<hostname in CS120>
export SPARK_MASTER_PORT=<port number>
export SPARK_MASTER_WEBUI_PORT=<port number>
export SPARK_WORKER_CORES=1
export SPARK_WORKER_MEMORY=2g
export SPARK_WORKER_INSTANCES=1
```

- Save the ***spark-env.sh*** file.

- **spark-defaults.conf**
  - This file is used to set default properties included when running **spark-submit**. This is useful for setting default environmental settings.
  - Each line in this file consists of a key and a value separated by **whitespace**.
  - Save ***spark-defaults.conf.template*** as ***spark-defaults.conf***.
  - Update following options with appropriate values in ***spark-defaults.conf***. You can refer sample below this table.

Property	Description
spark.master	The cluster manager to connect to. Value of this property will be "spark://<**SPARK_MASTER_IP**>:<**SPARK_MASTER_PORT**>"
spark.eventLog.enabled	Whether to log Spark events, useful for reconstructing the Web UI after the application has finished. (default: false)
spark.eventLog.dir	Base directory in which Spark events are logged, if **spark.eventLog.enabled** is **true**. Within this base directory, Spark creates a sub-directory for each application, and logs the events specific to the application in this directory. Users may want to set

	this to a unified location like an HDFS directory so history files can be read by the history server. (default: file:///tmp/spark-events) (e.g. hdfs://dover:42133/spark_log). If you are using HDFS directory, it **should exist** before launching Spark application. The URI for HDFS is referred from **fs.default.name** property in **core-site.xml**.
spark.serializer	Class to use for serializing objects that will be sent over the network or need to be cached in serialized form. The default of Java serialization works with any Serializable Java object but is quite slow, so we recommend using **org.apache.spark.serializer.KryoSerializer**
spark.logConf	Logs the effective SparkConf as INFO when a SparkContext is started. (default: false)
spark.executor.memory	Amount of memory to use per executor process. (default: 1g)

- Sample key-value pairs are given below. Put appropriate values in placeholders.

```
spark.master     spark://<SPARK_MASTER_IP>:<SPARK_MASTER_PORT>
spark.eventLog.enabled   true
spark.eventLog.dir               hdfs://<fs.default.name>/spark_log
spark.serializer    org.apache.spark.serializer.KryoSerializer
spark.logConf   true
spark.executor.memory        2g
```

- Save the ***spark-defaults.conf*** file.

5. Launching Spark Cluster
   - To be safe, start Hadoop cluster
   - Start the cluster
     - Login to Spark Masternode, specified in **SPARK_MASTER_IP.**
     - Run following script to launch Master as well as slaves.

```
$SPARK_HOME/sbin/start-all.sh
```

     - Check master webUI using <**SPARK_MASTER_IP**>:<**SPARK_MASTER_WEBUI_PORT**>

   - Stop the cluster
     - Login to Spark Masternode, specified in **SPARK_MASTER_IP.**
     - Run following script to launch Master as well as slaves.

```
$SPARK_HOME/sbin/stop-all.sh
```

- You can individually start and stop master as well as slave instances. Refer http://spark.apache.org/docs/latest/spark-standalone.html#cluster-launch-scripts for additional information.

6. Launching Spark Applications
   - Spark applications can be launched using ***spark-submit*** script.
   - Change directory to your project folder.
   - Run following command from any node of cluster with appropriate values.

```
$SPARK_HOME/bin/spark-submit --class <yourClass> --deploy-mode
cluster  --supervise  <yourJar>
```

You can refer http://spark.apache.org/docs/latest/submitting-applications.html for more information.