Creating a Stand-Alone Spark Environment in Windows Subsystem for Linux

Abraham Vargas

Abstract

Running a local *Spark* cluster has multiple advantages. First, it facilitates unit testing. This can decrease development time tenfold or more for larger applications. By running a local *Spark* cluster on a laptop or PC, smaller testing datasets can be created and multiple *Spark* configuration options can be experimented with without having to run tests on huge datasets through industrial clusters. Second, having a local setup allows for learning *Spark*, its interfaces and syntax, and optimizations. One can learn *Spark* without having to purchase or sign up for online services (e.g., *Amazon AWS*, *Microsoft Azure*, *Databricks*). Other advantages of running a local *Spark* cluster include tweaking options for optimization of production clusters and learning related technologies (e.g., *Hadoop*, *Hive*). Setting up a local *Spark* environment is best performed in *Linux*-based systems, as most Big Data technologies are developed for use in such systems. *Windows Subsystem for Linux* (*WSL*) provides a way to run a *Linux* system concurrently with *Microsoft Windows.* This avoids having to install a full *Linux* distribution, learning the new system, and setting up a multi-boot PC. This guide will utilize the *Ubuntu 18.04 Linux* distribution within *WSL* to teach the process of setting up a local *Spark* cluster.

Contents

1 Target Environment

The target environment in this guide is based on a real-life production setup of a Databricks cluster running on top of an Amazon Web Services (*AWS*) platform. The *Databricks* computing cluster this production system runs the following:

- *Databricks Runtime 7.3*
- *Ubuntu 18.04 LTS*
- *Spark 3.0.1*
- *Python 3.7.6*

As the scope of this guide is limited to configuring a local *Spark* environment in *WSL*, only *Spark 3.0.1* from the items above will be covered in detail.

2 Prerequisites

Before installing *Spark* and its related components *Hadoop* and *Hive*, several software, configuration, and other requirements must be met.

2.1 WSL 2

WSL allows a *Linux* distribution (distro) to run concurrently with *Windows 10*. *WSL 2* now includes a *Linux* kernel and is several times faster than the original *WSL*. This guide assumes that *WSL 2* is installed and is running the *Ubuntu 18.04* distro. To install *WSL 2*, follow the Windows Subsystem for Linux Installation Guide for Windows 10. Make sure to use *Ubuntu 18.04 LTS* as the distro.

2.2 Java

Spark, *Hadoop*, and *Hive* all require a *Java Runtime Environment* (*JRE*). *OpenJDK* is the standard *Java* platform for most *Linux* distributions. Though *OpenJDK 11* is the latest version, *Spark* and its related components require *OpenJDK 8*.

2.2.1 Install *JRE*

To install *OpenJDK 8*, run the following command in a terminal:

```
sudo apt install openjdk-8-jre-headless openjdk-8-jdk-headless
```

2.2.2 Set *JAVA_HOME*

Spark and other components will need to know the path of the *JRE*. This is accomplished by setting the *JAVA_HOME* system variable.

1. Open the *.bashrc* file with any editor (e.g., *vim*, *nano*). For example:

   ```
   vim ~/.bashrc
   ```

2. Add the following line to the *.bashrc* file:

   ```
   export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
   ```

3. Reload shell environment configuration:

   ```
   source ~/.bashrc
   ```

4. Verify that path was correctly set:

   ```
   echo $JAVA_HOME
   ```

   The output should match that in step 2.

2.3 PostgreSQL

Hive requires a database for its *metastore*. In terms of open-source options, *Hive* can use *MySQL*, *PostgreSQL*, and *Derby*. This guide will use *PostgreSQL*.

2.3.1 Install *PostgreSQL*

PostgreSQL 12 is the default version packaged for *Ubuntu 18.04*. Install *PostgreSQL 12* with the command:

```
sudo apt install postgresql-12
```

2.3.2 Start database server

The *PostgreSQL* will need to be started every time *Hive* is started. Initialize the server using the command:

```
sudo service postgresql start
```

2.3.3 Configure *Hive metastore*

The *psql* application is used to interact with *PostgreSQL* in a terminal.

1. Log into *PostgreSQL* via *psql* with default user *postgres*:

   ```
   sudo -u postgres psql
   ```

2. Add new user *hive* (for simplicity, the new user will also be given *hive* as the password):

```
CREATE USER hive WITH PASSWORD 'hive';
```

3. Create new database for *Hive metastore*:

   ```
   CREATE DATABASE hive_metastore;
   ```

4. Give ownership of *hive_metastore* database to user *hive*:

   ```
   ALTER DATABASE hive_metastore OWNER TO hive;
   ```

5. Exit *psql* with the command: `\q`

2.3.4 Install *Java Database Connectivity* Drivers

The *Java Database Connectivity* (*JDBC*) drivers allow *Java* to interact with databases. As *Hive* runs in *Java*, it requires the *PostgreSQL JDBC* drivers. Install the drivers in *Ubuntu* via the following command:

```
sudo apt install libpostgresql-jdbc-java
```

2.4 SSH

Secure Shell (*SSH*) encrypts communications over insecure networks. To accomplish encryption, *SSH* utilizes public and private keys.

2.4.1 Generate key

Generate an *ssh key* with the following command:

```
ssh-keygen -t rsa
```

Leave the password blank when prompted, as *Hadoop* will utilize a password-less *SSH* login.

2.4.2 Allow *localhost* login

Hadoop requires the ability to log into the *localhost* (i.e., your local PC) via password-less *SSH*. To accomplish this, run the following command:

```
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

2.4.3 Restart *SSH* service

Restart *SSH* and apply new settings:

```
sudo service ssh restart
```

Note: This command must be re-executed any time the following error message is encountered:

```
ssh: connect to host localhost port 22: Connection refused
```

2.4.4 *Could not load host key* error

If attempting to restart *SSH* service returns *Could not load host key* errors, run the following command:

```
ssh-keygen -A
```

Rerunning the command in 2.4.3 should now complete without errors.

2.4.5 Verify password-less *SSH* login

Ensure that *localhost* can be accessed via *SSH* without a password:

```
ssh localhost
```

3 Hadoop

Apache Hadoop is a framework that is used to store and process *big data* in distributed systems. In this setup guide, *Hadoop* will be used as a data storage system on a single PC.

3.1 Download and extract *Hadoop 3.3.0*

This guide will use the latest stable version, *Hadoop 3.3.0*.

3.1.1 Download *Hadoop*

Run the *wget* command in a terminal to download:

```
wget https://apache.osuosl.org/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
```

3.1.2 Extract files

Run the following command to extract the *tar.gz* file:

```
tar -xvzf hadoop-3.3.0.tar.gz
```

A new directory, `hadoop-3.3.0`, should now appear.

3.1.3 Copy folder to */opt*

In *Ubuntu*, `/opt` is the directory used to store add-on applications. *Hadoop*, *Hive*, and *Spark* will all be stored in this directory. Run the following command to move the extracted *Hive* folder:

```
sudo mv hadoop-3.3.0 /opt
```

3.2 Set *HADOOP_HOME*

Setting the *HADOOP_HOME* variable tells the system where *HADOOP* is located.

3.2.1 Edit *.bashrc*

Open the *~/.bashrc* file and add the following line:

```
export HADOOP_HOME=/opt/hadoop-3.3.0
```

3.2.2 Reload configuration

Load the new *bash* configuration with the command:

```
source ~/.bashrc
```

3.2.3 Verify path

Verify that *HADOOP_HOME* was correctly set:

```
echo $HADOOP_HOME
```

The output should match the path in 3.2.1.

3.3 Add *Hadoop* to system path

In order to run *Hadoop* commands without having to specify a full path, e.g.,

```
/opt/hadoop-3.3.0/bin/hadoop fs -mkdir /tmp
```

vs.

```
hadoop fs -mkdir /tmp
```

add the following line to ~/.bashrc:

```
PATH=$PATH:$HADOOP_HOME/bin
```

Make sure to reload the *Bash* configuration as in section 3.2.2.

3.4 Edit configuration files

In this section, *Hadoop* will be configured to run in single-node mode (i.e., on a single PC).

3.4.1 Edit *hadoop-env.sh*

Add the *JAVA_HOME* variable to the *Hadoop* environment, as in section 2.2.2.

1. Open the file `/opt/hadoop-3.3.0/etc/hadoop/hadoop-env.sh` with any editor

2. Un-comment line *54* and add the correct path:

```
# The java implementation to use. By default, this environment
# variable is REQUIRED on ALL platforms except OS X!
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

3.4.2 Edit *core-site.xml*

Edit the file `/opt/hadoop-3.3.0/etc/hadoop/core-site.xml` to look like the example below:

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://localhost:9000</value>
  </property>
  <property>
    <name>hadoop.proxyuser.abe.hosts</name>
    <value>*</value>
  </property>
  <property>
    <name>hadoop.proxyuser.abe.groups</name>
    <value>*</value>
  </property>
</configuration>
```

Replace *abe* above with your own *WSL Linux* username.

3.4.3 Edit *hdfs-site.xml*

Edit the file `/opt/hadoop-3.3.0/etc/hadoop/hdfs-site.xml` to look like the example below:

```
<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

3.4.4 Edit *mapred-site.xml*

Edit the file `/opt/hadoop-3.3.0/etc/hadoop/mapred-site.xml` to look like the example below:

```
<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.application.classpath</name>
    <value>
      $HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:
      $HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*
    </value>
  </property>
</configuration>
```

3.4.5 Edit *yarn-site.xml*

Edit the file `/opt/hadoop-3.3.0/etc/hadoop/yarn-site.xml` to look like the example below:

```
<configuration>
  <property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
  </property>
  <property>
    <name>yarn.nodemanager.env-whitelist</name>
    <value>
      JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,
      HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,
      HADOOP_YARN_HOME,HADOOP_MAPRED_HOME
    </value>
  </property>
</configuration>
```

3.5 Format *HDFS*

Format the *Hadoop Distributed File System* (*HDFS*) with the command:

```
hdfs namenode -format
```

The *hdfs* command should warn that the log file does not exist. This will be followed by several lines of *INFO* messages.

3.6 Start *NameNode* and *DataNode* daemons

The *NameNode* and *DataNode* daemons generate system stats that can be viewed through a web interface. To start both daemons, run the command:

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

The above command might return a *connection refused* error. If this occurs, restart the *SSH* service (see section 2.4.3). The web interface should now be accessible via http://localhost:9870/.

3.7 Start *YARN* daemon

YARN acts as a resource manager in a *Hadoop* system. To start the *YARN* daemon, run the command:

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

The resource manager web interface should now be accessible via http://localhost:8088/.

4 Hive

Apache Hive is a data warehousing platform that utilizes *SQL* to interact with data residing in distributed systems. *Hive* requires a *metastore* service, which in this case is provided by *PostgreSQL* (section 2.3).

4.1 Download and extract *Hive*

This guide will use the current latest stable version, *Apache Hive 3.1.2*.

1. Download *Hive* via the *wget* terminal command:

```
wget http://apache.mirrors.pair.com/hive/hive-3.1.2/apache-hive-3.1.2-bin.tar.gz
```

2. Unpack *Hive* file using the *tar* command:

```
tar -xvzf apache-hive-3.1.2-bin.tar.gz
```

3. Move new folder to /opt directory:

```
sudo mv apache-hive-3.1.2-bin /opt
```

4.2 Set *HIVE_HOME*

The *HIVE_HOME* variable must be set in order to let the system know *Hive's* location.

1. Open ~/.bashrc and add the following line:

```
export HIVE_HOME=/opt/apache-hive-3.1.2-bin
```

2. Also add the following line to include *Hive* in the system path:

```
PATH=$PATH:$HIVE_HOME/bin
```

3. Reload *Bash* terminal configuration:

```
source ~/.bashrc
```

4. Ensure *HIVE_HOME* path has been correctly set:

```
echo $HIVE_HOME
```

   Output should display the path configured in step 1 above.

4.3 Initialize all *Hadoop* services

As *Hive* manages data in a *Hadoop* system, the latter must be running. Run the following command in a terminal to start all *Hadoop* services:

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

The above command should warn that the configuration is not recommended for production. If *connection refused* errors are returned, restart *SSH* service as in section 2.4.3.

4.4 Configure *HDFS*

The *HDFS* directories that *Hive* will use must first be created and configured. Run the following commands to create the required *Hive* folders and assign appropriate permissions:

```
hadoop fs -mkdir /tmp
hadoop fs -mkdir -p /user/hive/warehouse
hadoop fs -chmod g+w /tmp
hadoop fs -chmod g+w /user/hive/warehouse
```

4.5 Edit *hive-site.xml*

The *hive-site.xml* file contains required configuration settings for the *Hive metastore*. In this setup, the *metastore* resides within a *PostgreSQL* database (see section 2.3).

4.5.1 Create *hive-site.xml*

Copy the file *hive-default.xml.template* and rename it as *hive-site.xml*:

```
cd /opt/apache-hive-3.1.2-bin/conf
cp hive-default.xml.template hive-site.xml
```

4.5.2 Edit configuration settings

Edit the following values in the *hive-site.xml* file at the lines shown below. Note that <*value*/> must be replace with </*value*> in *line 462*.

```
461    <name>hive.metastore.uris</name>
462    <value>thrift://127.0.0.1:9083</value>

                              ⋮

568    <name>javax.jdo.option.ConnectionPassword</name>
569    <value>hive</value>

                              ⋮

583    <name>javax.jdo.option.ConnectionURL</name>
584    <value>jdbc:postgresql://127.0.0.1/hive_metastore</value>

                              ⋮

1101   <name>javax.jdo.option.ConnectionDriverName</name>
1102   <value>org.postgresql.Driver</value>

                              ⋮

1126   <name>javax.jdo.option.ConnectionUserName</name>
1127   <value>hive</value>
```

4.5.3 Remove illegal characters

Line 3215 in *hive-site.xml* contains illegal characters that will cause a *Java RuntimeException*. Remove the characters highlighted below from line 3215:

```
<property>
  <name>hive.txn.xlock.iow</name>
  <value>true</value>
  <description>...locks for &#8; ...OVERWRITE.</description>
</property>
```

4.6 Create *PostgreSQL* database structure

The *Hive Schema Tool* will create a database structure for the *Hive metastore*. Before this can happen, a few configurations must be adjusted.

4.6.1 Update *guava* version

The versions of *guava* between *Hadoop 3.3.0* and *Hive 3.1.2* are not compatible. Having different versions will result in a *Java NoSuchMethodError* in step 4.6.2.

1. Delete *guava 19.0* from *Hive*:

   ```
   cd /opt/apache-hive-3.1.2-bin
   rm lib/guava-19.0.jar
   ```

2. Replace with *guava 27.0* included with *Hadoop*:

   ```
   cp /opt/hadoop-3.3.0/share/hadoop/hdfs/lib/guava-27.0-jre.jar lib/
   ```

4.6.2 Run *Schema Tool*

The command below will create the database structure for the *Hive metastore*. Once successfully completed, the output should conclude with the message *schemaTool completed*.

```
schematool -dbType postgres -initSchema
```

4.7 Start *Hive* services

In order to use *Hive*, its *metastore* and server must be started.

4.7.1 Prevent *URISyntaxException*

To prevent a *Java URISyntaxException* from being returned when attempting to run *Hive* services, add the following lines directly after *line 21* in *hive-site.xml*:

```
<property>
  <name>system:java.io.tmpdir</name>
  <value>/tmp/hive/java</value>
</property>
<property>
  <name>system:user.name</name>
  <value>${user.name}</value>
</property>
```

4.7.2 Start *Hive metastore* service

Run the following command in a terminal to start *Hive metastore* services:

```
hive --service metastore
```

4.7.3 Start *Hive* server

Open another terminal and run the following command to Initialize the *Hive* server:

```
hive --service hiveserver2
```

4.7.4 Access web interface

Hive also provides a web interface with information on running sessions, queries, etc. After about *one minute* of running both *metastore* and *Hive* server, access the web interface via `http://localhost:10002/`

5 Spark

With *Hadoop* and *Hive* configured, we can finally set up and use *Spark*.

5.1 Download and extract *Spark*

This guide will use *Spark 3.0.1*. As *Hadoop 3.3.0* is already installed, we'll download *Spark Pre-built for Apache Hadoop 3.2 and later*.

1. Use the *wget* command to download *Spark 3.0.1*:

   ```
   wget https://archive.apache.org/dist/spark/spark-3.0.1/spark-3.0.1-bin-hadoop3.2.tgz
   ```

2. Extract the downloaded file:

   ```
   tar -xvzf spark-3.0.1-bin-hadoop3.2.tgz
   ```

3. Move *Spark* folder into */opt* directory:

   ```
   sudo mv spark-3.0.1-bin-hadoop3.2 /opt
   ```

5.2 Set *SPARK_HOME* variable

Edit ∼/.bashrc and add the following line:

```
export SPARK_HOME=/opt/spark-3.0.1-bin-hadoop3.2
```

5.3 Add *Spark* to system path

Adding *Spark* to the system path will allow you to run *Spark* commands without using full paths. Add the following line to ∼/.bashrc:

```
PATH=$PATH:$SPARK_HOME/bin
```

5.4 Set local IP

Ubuntu adds a *hostname* entry to the */etc/hosts* file, which can cause conflicts with *localhost* when *Spark* attempts to resolve IP addresses. To avoid IP-related warnings and errors, add the following line to ∼/.bashrc:

```
export SPARK_LOCAL_IP=localhost
```

5.5 Reload configuration

Reload the new *Bash* shell configuration:

```
source ~/.bashrc
```

5.6 Enable *Hive* support

In order to use *Hive* with *Spark*, we need to copy configuration files from both *Hive* and *Hadoop*. Run the following commands to copy the necessary files to *Spark*:

```
cp $HADOOP_HOME/etc/hadoop/core-site.xml $SPARK_HOME/conf/
cp $HADOOP_HOME/etc/hadoop/hdfs-site.xml $SPARK_HOME/conf/
cp $HIVE_HOME/conf/hive-site.xml $SPARK_HOME/conf/
```

5.7 Enable *Spark* logs web interface

Spark provides a web interface from which event logs can be viewed.

5.7.1 Edit configuration file

In order to activate the logging capability of *Spark*, the *spark-defaults.conf* file must be created and edited.

1. Copy default *Spark* configuration file:

   ```
   cd $SPARK_HOME/conf
   cp spark-defaults.conf.template spark-defaults.conf
   ```

2. Add the following lines to *spark-defaults.conf*:

   ```
   spark.eventLog.enabled           true
   spark.eventLog.dir               hdfs://localhost:9000/spark-logs
   spark.history.fs.logDirectory    hdfs://localhost:9000/spark-logs
   ```

5.7.2 Create log directory in *HDFS*

The logs generated by *Spark* will be saved to a *HDFS* folder. Create the folder and assign appropriate permissions using the commands below:

```
hadoop fs -mkdir /spark-logs
hadoop fs -chmod g+w /spark-logs
```

5.7.3 Start *Spark* history server

Run the following command to start *Spark* history log server:

```
$SPARK_HOME/sbin/start-history-server.sh
```

5.7.4 Access web interface

The web interface can now be accessed via http://localhost:18080/. The page will display the message *No completed applications found!* the first time it's accessed. This message can be safely ignored and will not appear after Spark jobs are run.

5.8 *Spark* Shells

Spark provides multiple shells that utilize different programming and scripting languages.

5.8.1 Scala

The default shell uses *Scala* as a *Spark* interface. *Scala* is included with *Spark* and only requires a *JRE*. Entering the command below will start the default *Scala* shell:

```
spark-shell
```

5.8.2 Spark SQL

Spark includes *Spark SQL*, which utilizes *HiveSQL* syntax. To start a *Spark SQL* shell, run the command:

```
spark-sql
```

5.8.3 Python

PySpark is *Spark's* interface for the *Python* scripting language. Before using *PySpark*, *Python* must first be installed and configured. These tasks are beyond the scope of this documentation. The command below will initialize a *Python* shell:

```
pyspark
```

5.8.4 R

Spark also has an *R* interface, *SparkR*. Before utilizing the *SparkR* shell, *R* must first be installed and configured, which is beyond the scope of this documentation. Run the following command to start a *SparkR* shell:

```
sparkR
```

6 Putting It All Together

Now that *Spark* and all other necessary components are installed and configured, they must be activated every time when starting a new *Windows* session. Activating all components can be accomplished manually each time, or by the use of a script for automation.

6.1 Option 1: Manual activation

To manually start all components required for using a *Spark* shell, the following steps must be performed in order:

1. Start *PostgreSQL* server (section 2.3.2)

   ```
   sudo service postgresql start
   ```

2. Restart *SSH* service (section 2.4.3)

   ```
   sudo service ssh restart
   ```

3. Start all *Hadoop* services (section 4.3)

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

4. Start *Hive metastore* and server (section 4.7). The commands below must be run in **new** and **separate** terminals:

   ```
   hive --service metastore
   hive --service hiveserver2
   ```

5. Start *Spark* history server (section 5.7.3):

   ```
   $SPARK_HOME/sbin/start-history-server.sh
   ```

6.2 Option 2: Script automation

As an alternative to performing the steps above every time before being able to use *Spark*, you can create a simple *Bash* shell script.

1. Open an editor and enter the following lines:

   ```
   #! /bin/bash

   sudo service postgresql start
   sudo service ssh restart
   $HADOOP_HOME/sbin/start-all.sh
   nohup hive --service metastore > .hivemetastore &
   nohup hive --service hiveserver2 > .hiveserver &
   $SPARK_HOME/sbin/start-history-server.sh
   ```

Notice that the script above uses nohup. This is to avoid having to open multiple terminals for the *Hive* services to run in.

2. Save the script as *start-all-spark* (or any other name you'd like to give it)

3. Make the file executable with the command:

   ```
   chmod +x start-all-spark
   ```

4. Run the script to start all services required for using *Spark*:

   ```
   ./start-all-spark
   ```

7 Optional Configuration

The previous sections contain all necessary steps in setting up a local *Spark* environment. This section details additional configuration options that, though not required, facilitate developing applications in a local *Spark* environment.

7.1 PySpark

7.1.1 Hive warnings

A large number of *warning* messages are printed every time a *PySpark* session is started, similar to the example below:

```
WARN HiveConf: HiveConf of name hive.metastore.client.capability.check does not exist
```

These warnings result from *PySpark* initializing *Hive* with unsupported features. Each feature can be individually disabled by removing it from the file `/opt/apache-hive-3.1.2-bin/conf/hive-site.xml`. As a simpler alternative, the original `hive-site.xml` file can be replaced with a pre-configured version from this repository.

7.1.2 PYTHONPATH variable

In order to import *PySpark* modules when developing *Python* packages and applications, the *PYTHONPATH* environment variable must first be set.

1. Edit the `~/.bashrc` file by adding the lines:

   ```
   export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH
   export PYTHONPATH=$SPARK_HOME/python/lib/py4j-0.10.9-src.zip:$PYTHONPATH
   ```

2. Load the new settings

   ```
   source ~/.bashrc
   ```

7.2 Spark SQL

By default, starting a *Spark SQL* session will trigger dozens of *INFO* messages. This results in crowded and difficult to read *SQL* output. Use the following steps to configure the logging level so that only warning messages or higher are displayed.

1. Copy the default *log4j* template

   ```
   cd $SPARK_HOME/conf
   cp log4j.properties.template log4j.properties
   ```

2. Edit line 19 of the new `log4j.properties` file, as shown below

   ```
   log4j.rootCategory=WARN, console
   ```

References

[1] Hewlett Packard Enterprise Development LP, *Configuring a Remote PostgreSQL Database for the Hive Metastore*, 2020, https://docs.datafabric.hpe.com/61/Hive/Config-RemotePostgreSQLForHiveMetastore.html

[2] The Apache Software Foundation, *Hadoop: Setting up a Single Node Cluster*, 2020, https://hadoop.apache.org/docs/r3.3.0/hadoop-project-dist/hadoop-common/SingleCluster.html

[3] Debian Installer Team, *Ubuntu Installation Guide*, 2020, https://help.ubuntu.com/lts/installation-guide/amd64/index.html

[4] Pradeep Kumar, Javi Roman, *Hive Issue 22915*, 2020, https://issues.apache.org/jira/browse/HIVE-22915

[5] Raymond Tang, *Apache Hive 3.1.1 Installation on Windows 10 using Windows Subsystem for Linux*, 2018, https://kontext.tech/column/hadoop/309/apache-hive-311-installation-on-windows-10-using-windows-subsystem-for-linux

[6] Vu Duc Tiep, *[Hive Installation] java.net.URISyntaxException: Relative path in absolute URI*, 2017, http://driftingengineer.blogspot.com/2017/09/hive-installation-javaneturisyntaxexcep.html

[7] Raymond Tang, *Apache Spark 2.4.3 Installation on Windows 10 using Windows Subsystem for Linux*, 2018, https://kontext.tech/column/spark/311/apache-spark-243-installation-on-windows-10-using-windows-subsystem-for-linux

[8] The Apache Software Foundation, *Spark Configuration: Environment Variables*, 2021, https://spark.apache.org/docs/latest/configuration.html#environment-variables

[9] The Apache Software Foundation, *HADOOP2: Connection Refused*, 2019, https://cwiki.apache.org/confluence/display/HADOOP2/ConnectionRefused