Documentation

Rahul Gogoi

1 Hadoop installation on two Ubuntu containers

1. Create a network for Hadoop by executing the following instruction on the host machine:

```
podman network create hadoop-net
```

2. Create a file named **Dockerfile** to define the Ubuntu environment. This file will install all necessary dependencies, set up a user, configure passwordless SSH, and download Hadoop.

```
# Dockerfile
FROM ubuntu:22.04

# Avoid interactive prompts during installation
ARG DEBIAN_FRONTEND=noninteractive

# Install dependencies, including nano and vim
RUN apt-get update && apt-get install -y \
    openjdk-11-jdk \
    ssh \
    wget \
    pdsh \
    nano \
    vim && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Download and unpack Hadoop (v3.3.6 is a stable version)
RUN wget https://dlcdn.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz -P /tmp/ && \
    tar -xvf /tmp/hadoop-3.3.6.tar.gz -C /opt/ && \
    rm /tmp/hadoop-3.3.6.tar.gz && \
    ln -s /opt/hadoop-3.3.6 /opt/hadoop

# Add hadoop user and group
RUN groupadd --gid 1000 hadoop && \
    useradd --uid 1000 --gid 1000 -m hadoop

# Set up passwordless SSH for the 'hadoop' user
RUN su - hadoop -c "ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa" && \
    su - hadoop -c "cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys" && \
    chmod 600 /home/hadoop/.ssh/authorized_keys

# Set environment variables
ENV HADOOP_HOME=/opt/hadoop
ENV HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
ENV JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
ENV PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin

# Set default shell for hadoop user to bash
RUN chsh -s /bin/bash hadoop

# Change ownership of hadoop installation to the hadoop user
RUN chown -R hadoop:hadoop /opt/hadoop-3.3.6

# Create the parent directory for Hadoop data and set ownership
RUN mkdir -p /opt/hadoop_data
RUN chown -R hadoop:hadoop /opt/hadoop_data

# Copy config folder to Hadoop's config directory
```

```
COPY config/* $HADOOP_CONF_DIR/

# Resolve Missing privilege separation directory error
RUN mkdir /run/sshd

# Run SSH service
CMD ["/usr/sbin/sshd", "-D"]
```

3. Create the configuration directory by executing the following command:

```
mkdir config
```

Create final config files within the config directory:

- **config/hadoop-env.sh**:

```
# Set JAVA_HOME
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64

# Force pdsh to use ssh
export PDSH_RCMD_TYPE=ssh
```

- **config/core-site.xml**:

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop-master:9000</value>
    </property>
</configuration>
```

- **config/hdfs-site.xml**:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
    <property>
        <name>dfs.namenode.name.dir</name>
        <value>file:///opt/hadoop_data/hdfs/namenode</value>
    </property>
    <property>
        <name>dfs.datanode.data.dir</name>
        <value>file:///opt/hadoop_data/hdfs/datanode</value>
    </property>
</configuration>
```

- **config/yarn-site.xml**:

```
<configuration>
    <property>
        <name>yarn.resourcemanager.hostname</name>
        <value>hadoop-master</value>
    </property>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

- **config/mapred-site.xml**:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
```

```
        <name>yarn.app.mapreduce.am.env</name>
        <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
    </property>
    <property>
        <name>mapreduce.map.env</name>
        <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
    </property>
    <property>
        <name>mapreduce.reduce.env</name>
        <value>HADOOP_MAPRED_HOME=${HADOOP_HOME}</value>
    </property>
</configuration>
```

- **config/workers**:

```
hadoop-worker
```

4. Build the docker image by running the following command in the same directory as your Dockerfile:

```
podman build -t hadoop-ubuntu .
```

5. Launch two containers from the hadoop-ubuntu image: one for the master and one for the worker. We will map the necessary UI ports to your host machine.

```
# Launch the master container
podman run -d --name hadoop-master --hostname hadoop-master --network hadoop-net --cap-add=SYS_NICE -p 9870:9870 -p \
     8088:8088 hadoop-ubuntu

# Launch the worker container
podman run -d --name hadoop-worker --hostname hadoop-worker --network hadoop-net --cap-add=SYS_NICE hadoop-ubuntu
```

6. Format and Start the Cluster:
   These commands are run from your host machine but executed on the master container.

   (a) Format the NameNode:
   This initializes the HDFS filesystem. Only run this once!

   ```
   # Run this on your host machine
   podman exec -u hadoop hadoop-master hdfs namenode -format
   ```

   (b) Start HDFS and YARN daemons:
   The scripts will use the workers file to start daemons on all nodes.

   ```
   # Run these on your host machine
   podman exec -u hadoop hadoop-master start-dfs.sh
   podman exec -u hadoop hadoop-master start-yarn.sh
   ```

7. Verify the Hadoop Cluster:

   (a) Check Running Java Processes:
   Use the **jps** command on both containers.

   - On the master: You should see NameNode, ResourceManager, and SecondaryNameNode.

     ```
     podman exec -u hadoop hadoop-master jps
     ```

   - On the worker:
     You should see DataNode and NodeManager.

     ```
     podman exec -u hadoop hadoop-worker jps
     ```

   (b) Access Web UIs:
   Open your web browser and navigate to the following addresses:

   - HDFS NameNode UI: http://localhost:9870
     Go to the "Datanodes" tab to see your hadoop-worker listed.

- YARN ResourceManager UI: http://localhost:8088
  Go to the "Nodes" tab to see your hadoop-worker node.

Your two-node Hadoop cluster is now up and running on Podman! You can start submitting MapReduce jobs to the cluster.
You may login into the **hadoop-master**'s shell by executing:

```
podman exec -it -u hadoop hadoop-master bash
```

2 Load data, mapper and reducer onto the setup

1. Create a directory in the installation folder:

   ```
   mkdir process
   ```

2. Copy the required datafile, mapper.py and reducer.py onto the process folder.

3. Copy the content of the process file to the container:

   ```
   podman cp process/mapper.py hadoop-master:/tmp/mapper.py
   podman cp process/reducer.py hadoop-master:/tmp/reducer.py
   podman cp process/generated_data.csv hadoop-master:/tmp/generated_data.csv
   ```

4. Get to the hadoop master shell:

   ```
   podman exec -it -u hadoop hadoop-master bash
   ```

5. Create the **input** directory in **HDFS**:

   ```
   hdfs dfs -mkdir -p /user/hadoop/input
   ```

6. Upload the data file to input folder:

   ```
   hdfs dfs -put /tmp/generated_data.csv /user/hadoop/input
   ```

7. Run the **Hadoop streaming job**:

   ```
   hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-*.jar -files /tmp/mapper.py,/tmp/reducer.py -mapper '\
       python3 mapper.py' -reducer 'python3 reducer.py' -input /user/hadoop/input/generated_data.csv -output /user/\
       hadoop/output
   ```

   The options are:

   - **hadoop jar ...**: This tells Hadoop to run a job from the specified JAR file.
   - **-files /tmp/mapper.py,/tmp/reducer.py**: This is a crucial step. It tells Hadoop to ship your local script files to all the worker nodes in the cluster so they can be executed.
   - **-mapper 'python3 mapper.py'**: Specifies the command to run for the map task.
   - **-reducer 'python3 reducer.py'**: Specifies the command to run for the reduce task.
   - **-input /user/hadoop/input/...**: The input data path in HDFS.
   - **-output /user/hadoop/output**: The directory in HDFS where the final results will be stored. This directory must not exist before you run the command; Hadoop will create it.

8. After the job is complete, check the output:

   ```
   # 1. List the files in your output directory
   hdfs dfs -ls /user/hadoop/output

   # You will see a file named 'part-00000' and a '_SUCCESS' file.

   # 2. View the contents of the result file
   hdfs dfs -cat /user/hadoop/output/part-00000
   ```

3 Sample

For the generated_data.csv being of the form: To get the products purchased per unique phone number, the codes

PhoneNumber	Product
9127352678	Biscuit
9127352668	Biscuit
9127352678	Cheese
...	...

will be as follows:

- **mapper.py**:

```
#!/usr/bin/env python3
import sys

for line in sys.stdin:                      # The input provided to the mapper is through stdin.
    line = line.strip()                     # This will remove any leading or trailing whitespaces.
    if not line:                            # Incase the line is invalid, skip onto the next line.
        continue
    if line.startswith("PhoneNumber"):      # Skip header if present.
        continue
    phone, product = line.split(",", 1)     # As the file is a csv, the phone and product can be separated by \
     splitting them by the ',' (comma) symbol.
    print(f"{phone}\t{product}")            # Print the phone_number and the product.
# The output of the mapper will then be fed to the reducer using the stdin.
```

- **reducer.py**:

```
#!/usr/bin/env python3
import sys

current_phone = None
products = []

for line in sys.stdin:                                          # The input provided to the reducer is through \
     stdin.
    line = line.strip()                                         # Strip the line again in case any whitespaces \
     got introduced.
    if not line:                                                # Continue if not a valid line.
        continue
    phone, product = line.split("\t", 1)                        # Split the variables using '\t' as the delimiter\
      set by the mapper function.

    if current_phone == phone:                                  # Check phone number by phone number, if the \
     retrieved phone number = currently observed phone number, append the product to the product list.
        products.append(product)
    else:
        if current_phone:                                       # In case the phone number was set to none, set \
     the retrieved phone number to currently observed phone number.
            print(f"{current_phone}\t{','.join(products)}")
        current_phone = phone
        products = [product]

# Final output
if current_phone:
    print(f"{current_phone}\t{','.join(products)}")             # Print the output.
```