Running Hadoop and Python MapReduce in Docker (Windows 10)

1. Introduction

Hadoop is an open-source framework that allows distributed processing of large data sets across clusters of computers using a simple programming model called **MapReduce**.

This document will show how to set up a **single-node Hadoop environment using Docker** on Windows 10, explore its components (HDFS and YARN), and run a **Python MapReduce program** using the `mrjob` library.

2. Core Hadoop Components

Before starting, it is important to understand what each component does.

Component	Description
HDFS (Hadoop Distributed File System)	A distributed file system that stores data across multiple nodes. Even in a single-node setup, it behaves like a distributed system.
YARN (Yet Another Resource Negotiator)	A resource manager that handles scheduling and execution of jobs across cluster nodes.
MapReduce	The programming model Hadoop uses for distributed computation. A MapReduce job consists of a **Mapper** (processing input data into key-value pairs) and a **Reducer** (aggregating values for each key).

3. System Requirements

Resource	Requirement
Operating System	Windows 10 (Pro, Enterprise, or Home with WSL 2)
Memory	Minimum 8 GB (4–6 GB allocated to Docker)
CPU	2 or more cores
Disk Space	At least 20 GB free
Internet Access	Required for downloading Docker images

4. Installing Docker Desktop

1. Visit https://www.docker.com/products/docker-desktop/.

2. Download and install Docker Desktop for Windows.
3. During installation, **enable the WSL 2 backend**.
4. Restart your computer after installation.
5. Open Docker Desktop and ensure it is running (the whale icon should appear in the system tray).

---

5. Configuring Docker Resources

1. Open **Docker Desktop → Settings → Resources**.
2. Allocate:
   - **CPUs:** 2
   - **Memory:** 4–6 GB
   - **Swap:** 1–2 GB
3. Click **Apply & Restart**.

These settings ensure the Hadoop container has enough resources to run smoothly.

---

6. Setting Up the Hadoop Environment

6.1 Create a Working Folder

Open PowerShell or Command Prompt and create a folder for the project:

```
mkdir C:\hadoop-docker
cd C:\hadoop-docker
```

6.2 Create a Docker Compose File

Create a file named `docker-compose.yml` inside this folder and paste the following content:

```
version: '3'
services:
  hadoop:
    image: sequenceiq/hadoop-docker:2.7.1
    container_name: hadoop
    hostname: hadoop-master
    ports:
      - "9870:9870"   # HDFS NameNode UI
      - "8088:8088"   # YARN ResourceManager UI
    command: /etc/bootstrap.sh -bash
    tty: true
```

6.3 Launch the Hadoop Container

Run the following command inside your `C:\hadoop-docker` folder:

```
docker-compose up -d
```

This command downloads the Hadoop Docker image and starts a single-node cluster.

To confirm it is running:

```
docker ps
```

You should see an entry similar to:

```
CONTAINER ID   IMAGE                          COMMAND                  STATUS
abcd1234       sequenceiq/hadoop-docker:2.7.1  "/etc/bootstrap.sh…"   Up ...
```

---

7. Accessing Hadoop

You can access Hadoop from your web browser via the following interfaces:

Service	URL	Description
HDFS NameNode UI	http://localhost:9870	Browse files stored in HDFS.
YARN ResourceManager UI	http://localhost:8088	View job and resource status.

---

8. Accessing the Container Shell

To interact with Hadoop directly, open a terminal inside the container:

```
docker exec -it hadoop /bin/bash
```

Now you are inside the Hadoop environment.

Check Hadoop version:

```
hadoop version
```

List contents of HDFS root:

```
hadoop fs -ls /
```

---

9. Creating and Uploading a Sample Text File

9.1 Create a Local Text File Inside the Container

Inside the container:

```
mkdir /wordcount
cd /wordcount
echo "hello world bye world hello hadoop mapreduce world" > input.txt
```

9.2 Create an HDFS Input Directory

```
hadoop fs -mkdir /input
```

9.3 Upload the File to HDFS

```
hadoop fs -put input.txt /input/
```

9.4 Verify Upload

```
hadoop fs -ls /input
```

You should see the file listed.

You can also confirm through the **HDFS NameNode UI** at http://localhost:9870.

---

10. Installing Python and mrjob

The container already includes Python, but it may not have `pip`. Install it if missing, then install `mrjob`.

```
apt-get update
apt-get install -y python3-pip
pip3 install mrjob
```

Verify installation:

```
python3 -m pip show mrjob
```

11. Creating a Python MapReduce Script

We will create a word count example using the `mrjob` library.

Inside the container, in `/wordcount`, create a file:

```
cat << EOF > wordcount_mrjob.py
from mrjob.job import MRJob

class MRWordCount(MRJob):
    def mapper(self, _, line):
        for word in line.split():
            yield word, 1

    def reducer(self, word, counts):
        yield word, sum(counts)

if __name__ == '__main__':
    MRWordCount.run()
EOF
```

12. Running the MapReduce Job Locally (for Testing)

You can first test the script locally (not on Hadoop yet):

```
python3 wordcount_mrjob.py input.txt
```

Expected output:

```
"bye"    1
"hadoop" 1
"hello"  2
"mapreduce" 1
"world"  3
```

13. Running the Job on Hadoop

Now that the script works locally, we will run it on Hadoop through `mrjob`.

Command:

```
python3 wordcount_mrjob.py -r hadoop hdfs:///input/input.txt -o
hdfs:///output_mrjob
```

Explanation:

Option	Meaning
`-r hadoop`	Run on Hadoop using Hadoop Streaming
`hdfs:///input/input.txt`	Input file path in HDFS
`-o hdfs:///output_mrjob`	Output directory in HDFS

---

14. Viewing Job Progress and Logs

1. Open YARN UI: http://localhost:8088
2. Check running applications. Your job should appear here.
3. You can see logs, counters, and resource usage.

---

15. Viewing Output Data

Once the job completes, view results with:

```
hadoop fs -ls /output_mrjob
hadoop fs -cat /output_mrjob/part-00000
```

Expected output:

```
"bye"    1
"hadoop" 1
"hello"  2
"mapreduce" 1
"world"  3
```

---

16. Rerunning Jobs

If you want to rerun the job, delete the old output folder:

```
hadoop fs -rm -r /output_mrjob
```

Then re-execute the MapReduce command.

---

17. Understanding What Happened

- The `mrjob` library converted your Python code into a Hadoop Streaming job.
- Hadoop split the input text into chunks.
- Each chunk was processed by the **Mapper**, which emitted `(word, 1)` pairs.
- The **Reducer** aggregated all counts for each word and wrote them to `/output_mrjob` in HDFS.
- The results were combined into one file, `part-00000`.

---

18. Stopping and Removing the Cluster

When finished, exit the container and stop Docker:

```
exit
docker-compose down
```

This stops and removes the Hadoop container cleanly.

---

19. Troubleshooting

Problem	Possible Cause	Solution
`Cannot connect to Docker daemon`	Docker not running	Start Docker Desktop.
`hadoop-streaming*.jar not found`	Image not fully initialized	Recreate container (`docker-compose down && docker-compose up -d`).
`Permission denied` on Python scripts	Missing execute flag	`chmod +x script.py`
Output directory exists	Hadoop prevents overwrite	Delete with `hadoop fs -rm -r /output_mrjob`
Job fails with memory errors	Low memory allocation	Increase Docker memory to 6 GB or reduce dataset size.

---

20. Summary

You have now:

1. Installed and configured Docker and Hadoop on Windows 10.
2. Understood the basic structure of HDFS and YARN.
3. Created, uploaded, and viewed files in HDFS.
4. Written and tested a Python MapReduce job using `mrjob`.
5. Run and analyzed the job's output through both command line and Hadoop web interfaces.

---

21. Visualizing HDFS Data Through the Web Interface

Hadoop provides a simple **web-based file browser** built into the **NameNode UI**. It allows you to explore directories, upload and download files, and verify job outputs without using command-line tools.

21.1 Accessing the HDFS NameNode Web UI

1. Ensure your Hadoop container is running:

   ```
   docker ps
   ```

   You should see the container named `hadoop` in the list.

2. Open your web browser (Chrome, Edge, Firefox, etc.).

3. Navigate to:

   ```
   http://localhost:9870
   ```

4. You will see the Hadoop **NameNode web interface**.

21.2 Understanding the Interface

The NameNode UI provides several sections:

Section	Description
Overview	Displays cluster summary — total and remaining HDFS capacity, number of live/dead nodes.
Datanodes	Lists all nodes storing data blocks (in a single-node setup, you will see only one).
Utilities → Browse the File System	Opens a graphical file browser for HDFS directories and files.
Startup Progress	Shows NameNode startup and initialization progress.

21.3 Browsing HDFS Files

1. In the left navigation panel, click **Utilities → Browse the File System**.
2. You will see a directory listing similar to what `hadoop fs -ls /` shows on the command line.

You should find directories such as:

```

```

```
/input
/output_mrjob
/tmp
```

21.4 Viewing Your Uploaded File

1. Click on the `/input` directory.
2. Inside, you should see the `input.txt` file you uploaded earlier.
3. Click **input.txt** to open it.
4. The file's contents should appear, confirming the data is stored in HDFS.

This verifies your input was uploaded correctly.

21.5 Inspecting MapReduce Output

1. Go back to **Browse the File System**.
2. Navigate to the `/output_mrjob` directory (or `/output` if you used Hadoop streaming).
3. Inside, you will see files such as:

   ```
   part-00000
   _SUCCESS
   ```

4. Click on `part-00000` to view your job results in the browser.

You should see content similar to:

```
"bye"    1
"hadoop" 1
"hello"  2
"mapreduce" 1
"world"  3
```

This confirms that the MapReduce job executed correctly and saved results to HDFS.

21.6 Downloading Files from HDFS

If you want to download files from HDFS (for example, to view the results on your Windows machine):

1. In the browser, right-click on the file (e.g., `part-00000`).
2. Choose **Save link as...** or **Download Linked File** (depends on browser).
3. Save it locally.

This allows you to retrieve output data or logs for local analysis.

21.7 Uploading Files via the Web Interface

Although most Hadoop users upload data via the command line, you can also do it visually.

1. In the NameNode web interface, open the directory where you want to upload data (for example, `/input`).
2. Click **Upload File** in the top-right corner.
3. Browse for your local text file and confirm.
4. Once uploaded, it will appear in the directory listing.

This feature is especially helpful for beginners who prefer a graphical workflow.

21.8 Checking System Health

From the **Overview** or **Datanodes** tabs, you can monitor the cluster's health:

- Total HDFS capacity
- Remaining disk space
- Status of live and dead DataNodes
- Under-replicated or missing blocks (should be none in a single-node setup)

These details are useful to introduce students to the operational side of Hadoop.

21.9 YARN ResourceManager Web UI

In addition to the NameNode interface, Hadoop provides a second dashboard for job tracking and resource management.

Open:

```
http://localhost:8088
```

This shows:

- Active and completed MapReduce jobs
- Resource allocation (memory, CPU)
- Application logs
- History of previous job runs

You can click on your job name (e.g., *wordcount_mrjob.py*) to view counters, map/reduce progress, and diagnostics.

21.10 Summary of Web Interfaces

Interface	URL	Purpose
HDFS NameNode UI	http://localhost:9870	Explore and manage files stored in HDFS.
YARN ResourceManager UI	http://localhost:8088	Monitor running and completed jobs.

Both interfaces are valuable for demonstrating Hadoop visually in class and verifying each step of your workflow.

21.11 Common Issues When Using the Web UI

Problem	Likely Cause	Fix
Page not loading	Container not running	Run `docker ps` and restart container if needed.
"Connection refused" error	Docker port conflict	Ensure ports 9870 and 8088 are free or adjust in `docker-compose.yml`.
File not visible	Wrong HDFS path	Verify path using `hadoop fs -ls /` inside the container.

21.12 Practical Exercise (Optional for Students)

Ask students to:

1. Upload a new text file (e.g., `poem.txt`) via the web interface.
2. Run the same Python MapReduce word count on that new file.
3. Verify both input and output files visually using the HDFS browser.
4. Observe job execution details through the YARN UI.

This helps them connect the **conceptual** workflow (MapReduce stages) to **real system behavior** visible through the browser.

End of Section 21

At this point, your students should:

- Understand both command-line and visual workflows.
- Be comfortable navigating HDFS and YARN web interfaces.
- See real-time feedback when running Python-based Hadoop jobs.
