📘 Comprehensive Hadoop Ecosystem with Docker Compose — Full Stack Setup Guide

This document provides a detailed guide to setting up and managing a **Hadoop Ecosystem** using **Docker Compose**. The setup includes essential components such as **HDFS**, **YARN**, **Kafka**, **HBase**, **Hive**, **Spark**, and **Zookeeper**. Each section outlines its configuration, ports, dependencies, and usage for distributed data processing.

Rizwan Iftikhar

25 April 2025

🧰 Prerequisites

✅ 1. Install Docker

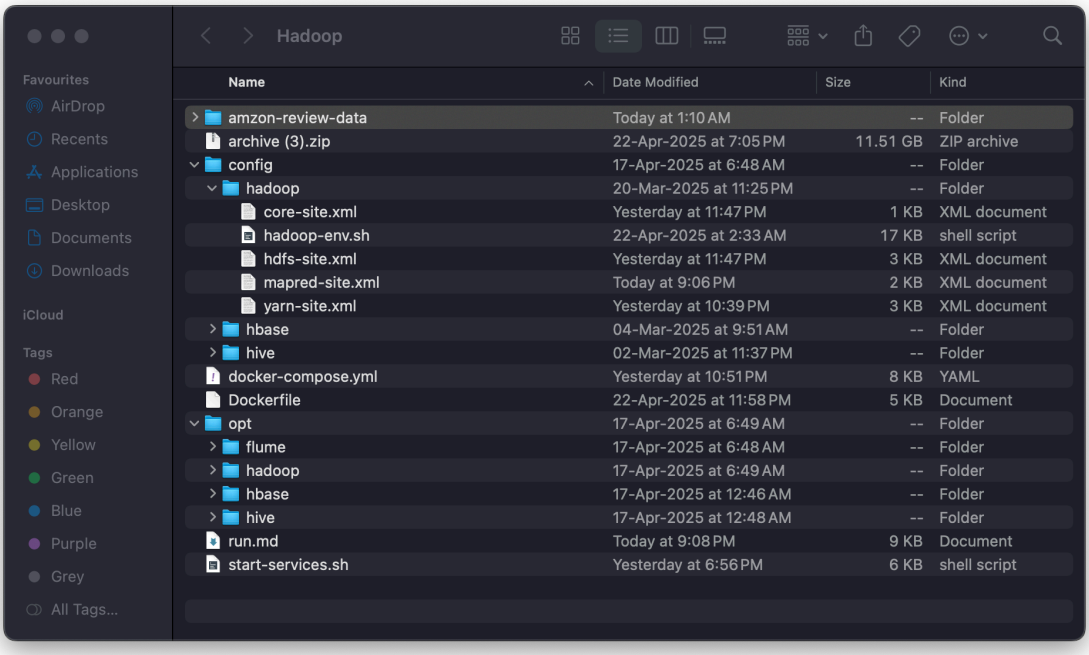
1. Go to: https://www.docker.com/products/docker-desktop
2. Download the version for **Mac or Window**.
3. Start Docker Desktop and allow permissions.

Verify Docker Installation:

`Run this command or terminal:`

```
docker –version
```

🔧 File Structure Overview



`Download or clone existing code from Github :` [`Hadoop-echo-system`]
`Download Hadoop, flume, base, hive from office site and copy into opt folder`

🐋 Step 1: Build Docker Image

Navigate to the directory where your `Dockerfile` exists:

If you're using Windows or Linux, make sure to update the `JAVA_HOME` path according to your operating system in the following files: `Dockerfile`, `start-services.sh`, `hadoop-env.sh`, and `hive-env.sh`.

```
cd /hadoop
docker build -t hadoop-ecosystem .
```

Check if image is created:

```
docker images
```

You should see something like:

```
REPOSITORY          TAG     IMAGE ID      CREATED     SIZE
hadoop-ecosystem    latest  a1b2c3d4e5f6  10 Sec ago  1.3GB
```

🌐 Step 3: Docker Network Explanation

Create a custom bridge network for communication:

```
docker network create hadoop-net
```

Why this is important:

- It allows all services (like HDFS, Spark, Kafka) to talk to each other.
- Docker assigns internal DNS names using container names.

In `docker-compose.yml`:

```
networks:
  hadoop-net:
    driver: bridge
```

🚀 Step 4: Start All Hadoop Services

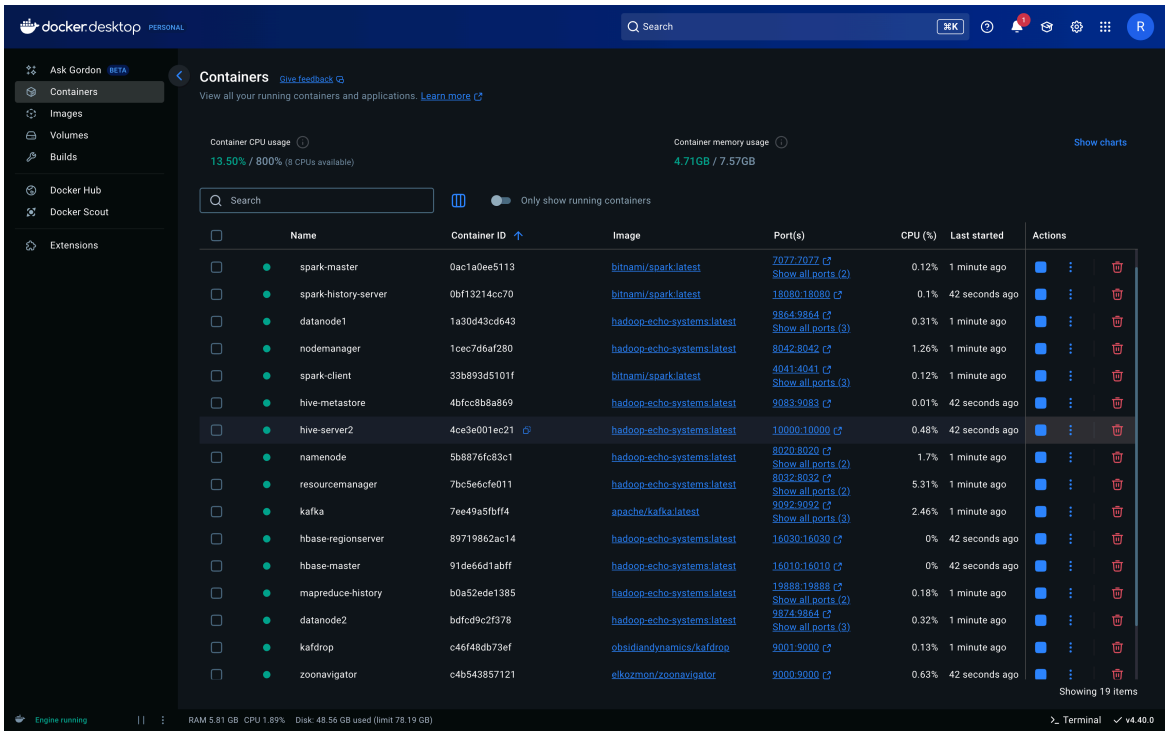
```
chmod +x start_services.sh

docker-compose up -d —build
```

To check services:

```
docker-compose ps
```

To stop services:

```
docker-compose down
```



To view the Docker container logs and verify whether the services are running, execute the following command:

```
docker logs <container_name>
```

```
Example: docker logs namenode
```

🔧 System Overview

The ecosystem consists of:

- **HDFS**: Hadoop Distributed File System for storing large datasets across distributed clusters.
- **YARN**: Resource management layer for the Hadoop ecosystem.

- **Kafka**: A distributed streaming platform for real-time data pipelines and streaming apps.
- **HBase**: A NoSQL distributed database built on top of HDFS.
- **Hive**: A data warehouse infrastructure built on top of HDFS for SQL-like querying.
- **Spark**: A powerful processing engine for big data workloads with support for batch and stream processing.
- **Zookeeper**: A distributed coordination service used for managing services like Kafka and HBase.

All components are interconnected via a Docker bridge network (`hadoop-net`), ensuring they can communicate seamlessly.

📌 Services and Ports

🗃️ HDFS (Hadoop Distributed File System)

Port	Component	Description
9870	NameNode Web UI	Monitor HDFS NameNode
8020	NameNode RPC	Client communication with HDFS
9864	DataNode1 Web UI	Monitor DataNode1
9866	DataNode1 Transfer	Block data transfer
9867	DataNode1 IPC	Internal protocol communication
9874	DataNode2 Web UI	Monitor DataNode2
9876	DataNode2 Transfer	Block data transfer
9877	DataNode2 IPC	Internal protocol communication

Example Job Execution:
open namenode container terminal with following command.

docker exec -it namenode /bin/bash

```
Execute this name node terminal:

hadoop jar $HADOOP_HOME/share/hadoop/mapreduce/hadoop-
mapreduce-examples-*.jar pi 10 1000
```

🧠 YARN (Yet Another Resource Negotiator)

Port	Component	Description
8088	ResourceManager UI	Monitor cluster resources/jobs
8032	ResourceManager RPC	Scheduler communication
8042	NodeManager Web UI	Monitor NodeManager

🧵 Zookeeper

Port	Component	Description
2181	Zookeeper	Coordination for Kafka, HBase, etc.

📫 Kafka

Port	Component	Description
9092	Kafka Broker	Handles producer/consumer messaging
9093	Kafka Controller	Internal cluster control
9001	Kafdrop UI	Web UI for Kafka topic inspection

📊 HBase

Port	Component	Description
16010	HBase Master UI	Monitor HBase master node
16030	RegionServer UI	Monitor HBase region server

🍯 Hive

Port	Component	Description
10000	HiveServer2	Thrift server for JDBC/ODBC

9083	Hive Metastore	Stores metadata for Hive tables

⚡ Apache Spark

Port	Component	Description
7077	Spark Master	Driver/worker communication
8080	Spark Master Web UI	View running Spark jobs
8081	Spark Worker Web UI	Monitor individual workers
18080	Spark History Server UI	Review completed jobs

🔗 Docker Network Configuration

- **Network Name**: `hadoop-net`
- **Network Driver**: `bridge`

💾 Docker Volumes

Volume Name	Purpose
namenode-data	Store HDFS NameNode metadata
datanode1-data	Store HDFS DataNode1 block files
datanode2-data	Store HDFS DataNode2 block files
kafka-data	Persist Kafka logs and topic data

🧩 Service Dependency Tree

- **HDFS**
  - DataNodes depend on NameNode.

- **YARN**
  - NodeManager depends on ResourceManager.
- **Kafka**
  - Kafka Broker depends on Zookeeper.
- **HBase**
  - HBase Master depends on NameNode and Zookeeper.
  - RegionServer depends on HBase Master.
- **Hive**
  - HiveServer2 depends on Hive Metastore.
- **Spark**
  - Spark Worker depends on Spark Master.
  - Spark History Server depends on Spark Master.

🌐 Web UI Access

Service	URL
HDFS NameNode	http://localhost:9870
YARN ResourceManager	http://localhost:8088
Spark Master	http://localhost:8080
Spark Worker	http://localhost:8081
Spark History UI	http://localhost:18080
Kafka Broker	`localhost:9092`
Zookeeper	`localhost:2181`
Kafdrop UI	http://localhost:9001
HBase UI	http://localhost:16010
HiveServer2	`localhost:10000`
Hive Metastore	`localhost:9083`

🚀 Submitting Spark Jobs from Host

1. Build your Spark JAR

Ensure version compatibility between **Scala** and **Spark**:

- **Scala version**: 2.12
- **Spark version**: 3.5.5

In `project/plugins.sbt`, add:

```
addSbtPlugin("com.eed3si9n" % "sbt-assembly" % “2.1.3")
```

To compile and package your Spark job:

```
sbt clean assembly
```

2. Copy JAR into Spark Client Container

```
docker cp target/scala-2.13/scalatest_2.13-0.1.0-
SNAPSHOT.jar spark-client:/opt/bitnami
```

3. Submit the Spark Job

Submit the job with dependencies (e.g., `spark-sql-kafka`):

```
docker exec -it spark-client /opt/bitnami/spark/bin/
spark-submit \
  --master spark://spark-master:7077 \
  --deploy-mode client \
  --class com.scala.hadoop.SparkKafka \
  --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:3.5.5 \
  /opt/bitnami/ScalaTest-assembly-0.1.0-SNAPSHOT.jar
```

Explanation:

- `--master spark://spark-master:7077`: Specifies the Spark Master URL.

- `--deploy-mode client`: Specifies that the job is run on the client machine (use `cluster` for remote).
- `--class com.scala.hadoop.SparkKafka`: Defines the main class of the Spark application.
- `--packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.5`: Adds the necessary Kafka connector for Spark (version 3.5.5 for Spark and 2.12 for Scala).
- ✅ Ensure that the version of `spark-sql-kafka` matches the version of Apache Spark and Scala in your environment.

🧪 HDFS Output Access & MapReduce Example

1. View MapReduce Output in HDFS

To view the results of MapReduce jobs stored in HDFS:

```
hdfs dfs -cat /output/amazon_reviews/part-*
```

2. Run a MapReduce Job

Submit a custom MapReduce job:

```
hadoop jar target/AmazonReviewAnalysis-1.0-SNAPSHOT.jar
\
  com.amazon.AverageRatingPerCategory \
  /user/hadoop/amazon_reviews \
  /user/hadoop/output
```

3. Copy JAR into NameNode Container

```
docker cp target/Amazon-Review-1.0-SNAPSHOT.jar
namenode:/home/ubuntu/
```

🗃️ MapReduce History Server

For tracking and viewing the history of MapReduce jobs, ensure the **MapReduce History Server** is configured correctly.

Access via the **YARN ResourceManager UI** or a dedicated URL:
➡️ http://localhost:19888

To enable this,

add the following to your `mapred-site.xml`:

```
<property>
  <name>mapreduce.jobhistory.webapp.address</name>
  <value>mapreduce-history:19888</value>
</property>
<property>
  <name>mapreduce.jobhistory.enabled</name>
  <value>true</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>
```

🌐 Hostname-Based Networking in Configuration Files

Important Note:
In all Hadoop-related configuration files (such as `core-site.xml`, `hdfs-site.xml`, `yarn-site.xml`, `hive-site.xml`, etc.), we use **Docker container hostnames** (e.g., `namenode`, `datanode1`, `resourcemanager`) instead of static IP addresses.

💡 Why We Avoid Static IPs

- Docker dynamically assigns IP addresses to containers each time they are started.
- If configuration files use static IPs, the Hadoop ecosystem would fail to connect properly after a system or container restart, as the IPs will likely change.
- By using container **hostnames**, Docker's internal DNS ensures that services always resolve correctly, no matter how many times containers are restarted.

🔁 **Think of it Like This:**

Each container is given a unique, predictable **domain name** instead of relying on a changing **IP address**.

Example (`core-site.xml`):

```
<property>
  <name>fs.defaultFS</name>
  <value>hdfs://namenode:9000</value>
</property>
```

Instead of:

```
<value>hdfs://172.18.0.3:9000</value> <!--❌ not recommended-->
```

✅ **Benefit:**

- Improved reliability
- Easier cluster configuration
- Seamless container orchestration

📦 Pre-Built Official Docker Images (Zookeeper, Kafka, Spark)

Note:
In this setup, for certain services like **Zookeeper**, **Kafka**, and **Spark**, we have used **official pre-built Docker images** provided by trusted vendors such as Bitnami or Apache.

🔧 **Configuration Handling:**

Unlike Hadoop, Hive, or HBase (which require manual configuration through XML files), these services are configured primarily through **environment variables** defined in the `docker-compose.yml` file.

This includes settings like:

- **Zookeeper:**
  ```
  ZOO_MY_ID: 1
  ALLOW_ANONYMOUS_LOGIN: yes
  ```
- **Kafka:**
  ```
  KAFKA_CFG_ZOOKEEPER_CONNECT: zookeeper:2181
  KAFKA_CFG_ADVERTISED_LISTENERS: PLAINTEXT://
  kafka:9092
  ```

- **Spark:**
  ```
  SPARK_MODE: master/worker
  SPARK_MASTER_URL: spark://spark-master:7077
  ```

✅ **Why This Works:**

- These images are **production-ready** and come with **built-in service logic**, so we don't need to manually configure properties through separate files.
- Using environment variables keeps the setup **simple**, **clean**, and **easy to manage**.
- Configuration changes can be made easily by updating the `docker-compose.yml` without modifying container internals.