

Map reduce with docker swarm

Prepared by: Daniyal Akbar (25363)



June 2, 2022

Course: Big data analytics

Instructor: Dr. Tariq Mahmood

Table of Contents

[1. Introduction 2](#_Toc104999220)

[2. Technologies Used 2](#_Toc104999221)

[a) Docker 2](#_Toc104999222)

[b) Docker Swarm 2](#_Toc104999223)

[c) Networking in Docker Swarm: 2](#_Toc104999224)

[d) Hadoop 2](#_Toc104999225)

[e) HBase 3](#_Toc104999226)

[f) Python Flask 3](#_Toc104999227)

[g) Python HappyBase 3](#_Toc104999228)

[3. GitHub Repository 3](#_Toc104999229)

[4. Docker Hub Repository 3](#_Toc104999230)

[5. Datasets 3](#_Toc104999231)

[6. Prerequisites 3](#_Toc104999232)

[7. Setting Up Docker Swarm Environment 4](#_Toc104999233)

[8. Running Flask Web App Service in Docker Swarm 4](#_Toc104999234)

[9. Running Hadoop Cluster with HBase Using Docker 4](#_Toc104999235)

[10. Flask Frontend Framework 5](#_Toc104999236)

[A. Architecture 5](#_Toc104999237)

[B. API 5](#_Toc104999238)

[C. Flask Dockerfile 6](#_Toc104999239)

[D. MapReduce Examples 6](#_Toc104999240)

[i. Searching Product 6](#_Toc104999241)

[ii. Sorting and Filtering 6](#_Toc104999242)

[iii. Related Products Category 6](#_Toc104999243)

[iv. Which Product Sells Most in Which Season 6](#_Toc104999244)

[v. Spam Detection with Comments or Contact Form Messages 7](#_Toc104999245)

# Introduction

This project aims to accomplish both front-end and back-end services and process using big dataset. It is running Hadoop cluster and HBase database on top of it for the back-end services. For front-end, a Python web-framework Flask has been used; also deployed using docker container in a Docker Swarm environment. All these services work in conjunction to execute MapReduce queries performed on a large data set for real time querying purpose.

# Technologies Used

Here I am going to list and explain all the technologies that are working in conjunction to produce this project.

## Docker

Docker is a software platform that allows you to build, test, and deploy applications quickly. Docker packages software into standardized units called containers that have everything the software needs to run including libraries, system tools, code, and runtime. Using Docker, you can quickly deploy and scale applications into any environment and know your code will run.

## Docker Swarm

A Docker swarm is a group of Docker nodes that have been configured to join and interact together in a cluster. We do not need to manage each of the nodes individually, as if they are separate machines. We could just treat these as one large machine/node. If we need more resources, we can scale our swarm by adding more nodes to it.

## Networking in Docker Swarm:

There are two types of networking used to create Docker Swarm:

1. Bridge Networking:

* Bridge is a virtual switch inside Linux
* Bridge networking is used to connect two or more containers in a single node docker swarm.

1. Overlay Networking:

* Overlay network is a virtual extensible LAN (VxLAN) in Linux
* Overlay networking is used to connect two or more containers in a multinode docker swarm.

## Hadoop

Big data refers to any type of dataset which requires non-relational technologies to implement a data management framework. The framework involves data gathering and ingestion, data integration, data storage and data lakes, data engineering and wrangling, data analysis and processing, and extraction of valuable insights.

## HBase

HBase is a column-oriented (reference Vertica, Infobright) database designed for Big Data solutions, which means it stores data tables by columns rather than rows. This makes data retrieval faster. Tables in it are however, sorted row wise and table schema is defined only through column families which are in other words the attributes. The column families work as key-value pairs.

## Python Flask

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier. It gives developers flexibility and is a more accessible framework for new developers since you can build a web application quickly using only a single Python file.

## Python HappyBase

HappyBase is a developer-friendly Python library to interact with Apache HBase. HappyBase is designed for use in standard HBase setups, and offers application developers a Pythonic API to interact with HBase.

# GitHub Repository

[Click here to visit GitHub Repository for this Project.](https://github.com/Programatically/Flask_Hbase_Docker_Swarm.git) (https://github.com/Programatically/Flask\_Hbase\_Docker\_Swarm)

# Docker Hub Repository

[Click here to visit Docker Hub Repository for this Flask Front-end Image.](https://hub.docker.com/r/daniyal1217/flask_webapp) (https://hub.docker.com/r/daniyal1217/flask\_webapp)

# Datasets

For this project I have used two different datasets both of which I've found on Kaggle. I have uploaded and saved the datasets in my GitHub repository, [click here to check it out.](https://github.com/Programatically/Flask_Hbase_Docker_Swarm/tree/main/myvol)

# Prerequisites

* Docker should be installed (Docker for Desktop in windows OS)
* Docker compose should be installed

# Setting Up Docker Swarm Environment

* Download the repository that I've mentioned the link in Step 4
* Go to **Flask\_Hbase\_Docker\_Swarm/myvol/** and you will find the docker-compose.yml file for setting the swarm nodes.
* Execute this code: docker-compose up -d
* Once started, open interactive shell for node1 using this command: docker exec -it *<node1 container id>* sh
* Check node1 IP address with this command: ip add
* Now to make node1 the manager node, execute this command: docker swarm init --advertise-addr *<node1 ip address>*
* This will initiate the swarm mode and will display a joining token. Copy that token and then open interactive terminal for node2 and node3 and copy the token in them. This will make them the worker node.
* Your docker swarm environment has been set up. To check and list all the nodes, execute this command: docker node ls

# Running Flask Web App Service in Docker Swarm

- While being inside node1 interactive shell, execute this command: docker service create --name flask\_webapp --publish 5000:5000 daniyal1217/flask\_webapp:0.0.1

- The above command will download the Flask WebApp docker image from my DockerHub account and spin up its container.

- It will expose the service port to node1 container port at 5000. Which is already been exposed to the local host machine through docker-compose.yml file.

- Simply go to a browser and type in the URL localhost:5000 this will open up the last web app on your localhost machine.

- NOTE: The site has quite a lot of dummy data. Among which I’ve add my MapReduce tasks and examples.

# Running Hadoop Cluster with HBase Using Docker

* While being inside node1 interactive shell, execute this command: docker service create --name hbase-docker --publish 16010:16010 dajobe/hbase:latest
* You should have already downloaded my GitHub repository in which the data set already exists in **Flask\_Hbase\_Docker\_Swarm/myvol/** folder.
* Go into that folder where the datasets are and execute the following command to run a docker multi node container for Hadoop and HBase: docker service create --name=hbase-docker -h hbase-docker -d -v //myvol/://data dajobe/hbase
* The above command will download a container from DockerHub and run its container on your machine and spin up all the multi node container which are the dependencies for running Hadoop and HBase cluster (i.e. Thrift, REST server, Zookeeper etc)
* Next, we need to copy the dataset files inside the running HBase container.
* Copy the HBase docker container ID by the following command: docker ps
* Firstly, stay inside the **/myvol** folder and execute the following commands to copy the dataset files inside the container:
  + docker cp Products\_Dataset.csv *container\_ID*:/data/
  + docker cp Spam\_Dataset.csv *container\_ID*:/data/
* Now that we have copied the dataset files inside the container, we need to open interactive shell of the HBase container and then import these datasets
* After opening up the interactive shell, execute the following command: hbase shell
* now we need to create two tables for the data set to import into. for that execute the following two queries:
  + create 'spam', 'Message'
  + create 'productss', 'CustomerID', 'Name', 'Quantity', 'Date', 'Price'
* After this exit the HBase shell by pressing CTRL + C
* To import the datasets, execute the following 2 queries, while remaining inside the container interactive shell:
  + hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=',' -Dimporttsv.columns=HBASE\_ROW\_KEY,Message spam /data/Spam\_Dataset.csv
  + hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.separator=',' -Dimporttsv.columns=HBASE\_ROW\_KEY,CustomerID,Name,Quantity,Date,Price productss /data/Products\_Dataset.csv
* And you are done with setting up Hadoop and HBase cluster with all the required configurations for this project.
* Go to frontend webapp URL and test out the various MapReduce examples.

# Flask Frontend Framework

## Architecture

For the frontend framework, I've kept it very straightforward and simple. I have used bootstrap 4.4.1 and jQuery 3+. I have imported various dummy content and media files, that I have all kept in a static folder within the **Flask\_Hbase\_Docker\_Swarm/flask\_webapp/** folder in GitHub repository.

My main Python startup file for flask web app is named **view.py** which is in the same directory. Next, I have created a library with the name Hbase\_API.py, also kept in the same folder view.py file. There is a **templates** folder in which I've kept all HTML layout files for the frontend web pages

## API

For all the MapReduce examples listed below in step D, I have created various API's in Python language using **HappyBase** library. HappyBase library is a Python library used to interact with HBase services using simple python scripts. Here is the list of API method names that I've created which are present in **HBase\_API.py** file in the repository

* fetch\_table()
* search\_products()
* filter\_low\_price()
* filter\_latest()
* related\_products()
* seasonal\_products()
* detect\_spam()

## Flask Dockerfile

All the required services in creating this flask webapp have all been compiled into one docker container using the Dockerfile, that you can find in **Flask\_Hbase\_Docker\_Swarm/flask\_webapp/** directory.

## MapReduce Examples

### Searching Product

This is a feature implemented in the **Services** page of the frontend. There is a search bar where a customer can type any text, not necessarily be exact name of the product and hit search. This will hit the API and start a MapReduce process to fetch all the records which contains that text. The list of fetched products are then populated below into proper card format for better representation and visualization

### Sorting and Filtering

This is a feature implemented in the **Services** page of the frontend. It's a continuation of the previous feature in which you there are two radio buttons on the right side of the search bar, to choose which filter you want to apply in fetching the records. The two filters are:

- search by latest products

- search products from lowest to high price.

### Related Products Category

This is a feature implemented in the **single\_product** page of the frontend. When any of the search product is clicked, it will redirect to this page and well show you more information of that specific product, along with a related product list at the bottom of this page. This is done using map reduce function that fetches those products which share the same category as the one clicked.

### Which Product Sells Most in Which Season

This is a feature implemented in the **seasonal\_products** page of the frontend. In this page, there is a list of products shown and at the top there are four options to select from. These four options are all the four seasons around the year. Upon clicking them the product lists updates and shows only those specific products which sells the most in that particular season according to the price and quantity sold.

### Spam Detection with Comments or Contact Form Messages

This is a feature implemented in the **contact** page of the frontend. When anyone fills out the form fields and hit send, the message body is then checked If it contains any spam words. It is being cross-checked from the data inside the spam dataset which we imported earlier. If the message body contains any of those spam words, then it shows us and alert weather if the message body is spam or not.