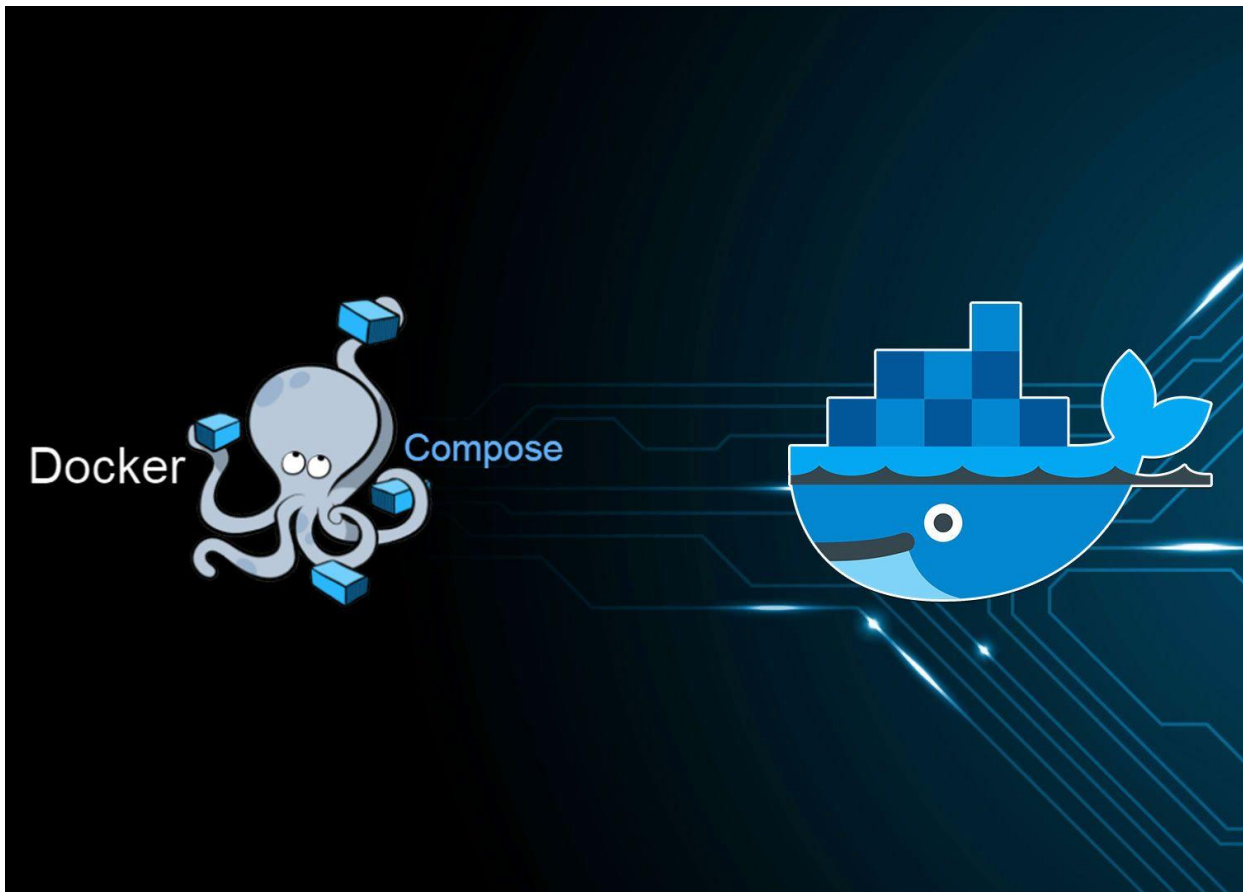


Containerization Report

Mohannad Atmeh



Abstract

This report is about an Admin-User System where only Admin can input numbers to MySQL database and Users can only see some statistics on them, such as maximum value, minimum, and average that were taken from MongoDB.

The System was built by multiple Docker images, and they are interacting with each other using Docker-Compose.

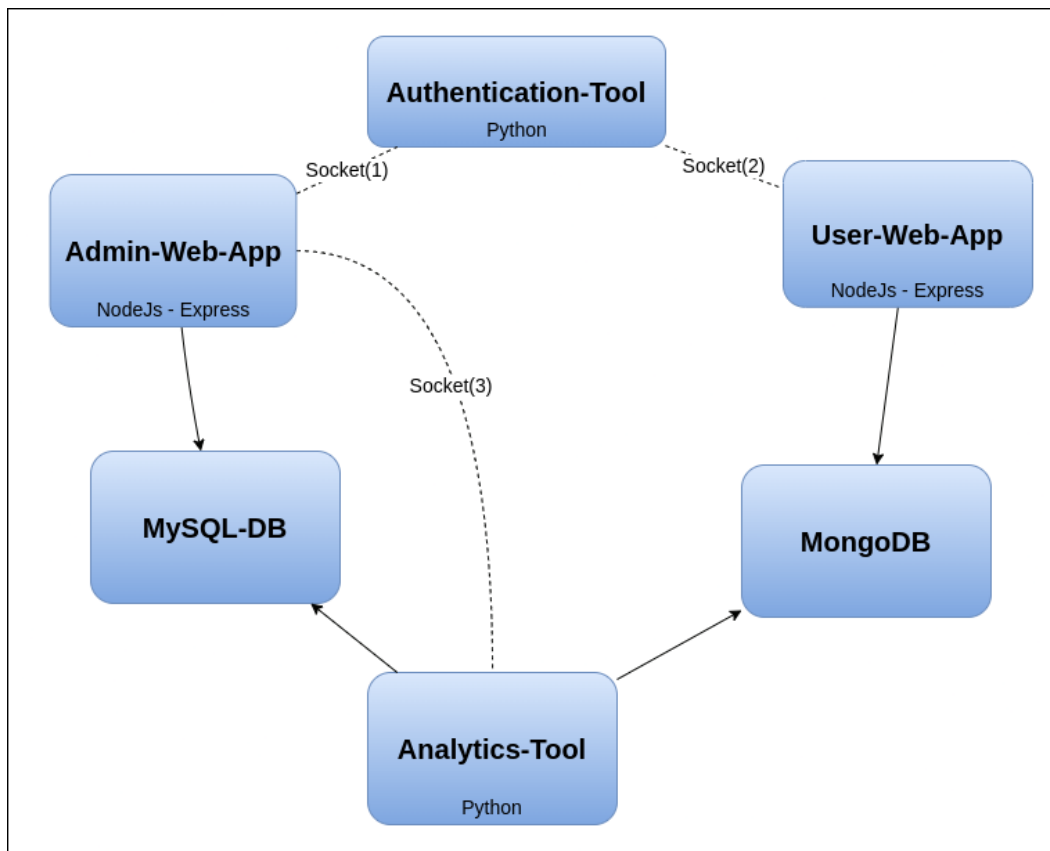
Introduction	3
Quick View At The System	4
MySQL-DB (mysql_server)	6
DockerCompose	6
MongoDB (mongodb)	7
DockerCompose	7
Admin-Web-App (app)	8
DockerFile	8
DockerCompose	9
User-Web-App (app)	10
DockerFile	10
DockerCompose	11
Analytics-Tool (analytics-service)	12
DockerFile	12
DockerCompose	13
Authentication-Tool (authenticator)	14
DockerFile	14
DockerCompose	15

Introduction

We were asked to make a Docker-Compose system that will run **Admin Web App**, **User Web App**, **Authentication Service**, **MySQL**, **MongoDB**, and **Analytical Service** images.

Admins will access the system by logging in to **Admin Web App** so they can add specific numbers or random numbers to **MySQL** Database, also they can delete all data. Adding numbers to MySQL will automatically trigger an **Analytical Service** which will calculate the Maximum number, minimum number, and the average of all numbers, sending these informations and storing them into **MongoDB**. All of that so users can log in to **User Web App** and fetch that information from **MongoDB**.

Quick View At The System



- **Admin-Web-App (app)**: Container with an image of the NodeJs Admin-Web-App scripts. We can access it from the browser after `docker-compose up` using <http://localhost:3000/> .
 - **User-Web-App (user_app)**: Container with an image of the NodeJs User-Web-App scripts. We can access it from the browser after `docker-compose up` using <http://localhost:3002/> .
 - **MySQL-DB (mysql_server)**: Container with `mysql` image, we can access it from `host="mysql_server"`, `port=3306`, database and tables are generated from the Admin-Web-App if not already exists; it has a volume file to save data even after it is closed.
-

-
- **MongoDB (*mongodb*):** Container with *mongo* image, we can access it from host="**mongodb**", port=**27017**, database and tables are generated from the Analytics-Service if not already exists; It doesn't need volume file to save its data in this project since it will be updated constantly, but we can add one easily.
 - **Authentication-Tool (*authenticator*):** Container with an image of python script that will receive name and password by sockets to authenticate admins and users whenever they logged in. [name: admin . pass: admin | | name: user . pass: user].
 - **Analytics-Tool (*analytics-service*):** Container with an image of python script that will fetch all numbers from *mysql_server* to get min, max and avg of them, then store the output in *mongodb*. Must be executed by a network communication (socket) to do the job.
 - **Socket(1): *authenticator*** is listening to *app* using this socket. Localhost: "authenticator", port: 6001 to authenticate name and password. [must be admin admin].
 - **Socket(2): *authenticator*** is listening to *user app* using this socket. Localhost: "authenticator", port: 6000 to authenticate name and password. [must be user user].
 - **Socket(3): *analytics service*** is listening to *app* using this socket. Localhost: "analytics_server", port: 65432. Once connection only (without communication) the analytics-tool will to job of fetching data from mysql, calculate and store output to mongodb.
-

MySQL-DB (mysql_server)

DockerCompose

```
mysql_server:
  image: mysql:8.0
  environment:
    - MYSQL_DATABASE=test_db
    - MYSQL_USER=dan
    - MYSQL_PASSWORD=secret
    - MYSQL_ROOT_PASSWORD=secret
  volumes:
    - ./MySqlDb:/var/lib/mysql
  ports:
    - 3306:3306
```

Translation:

- This container is named *mysql_server*
- Build its image from mysql:8.0 that can be downloaded from DockerHub
- Send these environment data when building that image...
- Let the container's port mysql_server:3306 be exposed to localhost:3306
- Let the directory in the host machine “./MySqlDb” be copied inside the container in “/var/lib” with the name “/mysql” instead and make it a volume (so we don't lose the data even if the container is closed).

MongoDB (mongodb)

DockerCompose

```
mongodb:
  image: mongo
  container_name: mongodb
  environment:
    - PUID=1000
    - PGID=1000
  volumes:
    - './mongodb/database:/data/db'
  ports:
    - 27017:27017
  restart: "unless-stopped"
```

Translation:

- This container is named *mongodb*
- Build its image from *mongo* that can be downloaded from DockerHub
- Send these environment data when building that image...
- Let the container's port `mongodb:27017` be exposed to `localhost:27017`
- Let the directory in the host machine `"./mongodb/database"` be copied inside the container in `"/data"` with the name `"/db"` instead and make it as a volume (so we don't lose data even if the container is closed).
- If the container ever crashes, restart it until the user closes it.

Admin-Web-App (app)

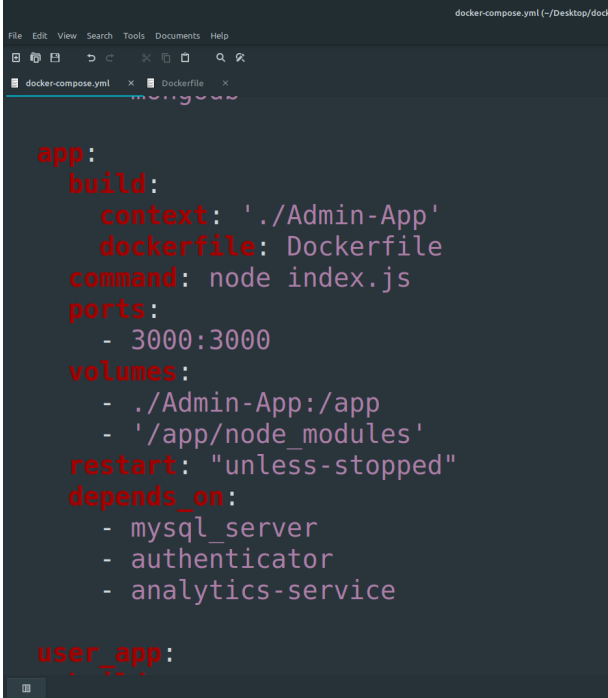
DockerFile

```
FROM node:14
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 3000
VOLUME [ "/app/node_modules" ]
```

Translation:

- Get an image of node version 14 from DockerHub
 - Inside this container go to directory /app, then copy the file "package.json" from the current directory on the host machine to /app directory in the container (package.json has the name of all modules needed in the scripts)
 - Then run the command "npm install" (this command is to download all modules required from npmjsHub)
 - Then copy all files in this host directory to /app directory in the container (this line where scripts goes to the container)
 - Then make this container exposed to port 300
 - Finally make let the directory node_modules be a volume (so we don't need to re-download all modules again every time the container is closed)
-

DockerCompose

A screenshot of a code editor window titled 'docker-compose.yml (-/Desktop/docker)'. The editor shows a Docker Compose configuration for a service named 'app'. The configuration includes build instructions (context, dockerfile), a command to run 'node index.js', port mapping (3000:3000), volume mounts for local directories, a restart policy of 'unless-stopped', and dependencies on 'mysql_server', 'authenticator', and 'analytics-service'. A second service 'user_app' is partially visible at the bottom.

```
app:
  build:
    context: './Admin-App'
    dockerfile: Dockerfile
  command: node index.js
  ports:
    - 3000:3000
  volumes:
    - ./Admin-App:/app
    - '/app/node_modules'
  restart: "unless-stopped"
  depends_on:
    - mysql_server
    - authenticator
    - analytics-service

user_app:
```

Translation:

- This container is named *app*
 - Take the build constructions for this container using the file “DockerFile” in the directory ./Admin-App
 - Let the container’s port app:3000 be exposed to localhost:3000
 - Let the directory in the host machine “./Admin-App” be copied inside the container with the name “/app” instead and make it as a volume.
 - Keep the data (files) inside “/app/node_modules” that is inside the container be saved even if the container is closed.
 - After building this container with its volumes start the command “node index.js” (which is the command used to execute Nodejs/Express files)
 - Whenever the container is closed restart it until it’s being closed by user
 - Don’t start this container until mysql_server, authenticator, and analytics_service are already built.
-

User-Web-App (app)

DockerFile

```
FROM node:14
WORKDIR /app
COPY package.json .
RUN npm install
COPY . .
EXPOSE 3002
VOLUME [ "/app/node_modules" ]
```

Translation:

Same as the Admin-Web-App (app) above, the only difference is the port number which is 3002 for user_app instead of 3000 that was for Admin app.

DockerCompose

```
user_app:
  build:
    context: './User-App'
    dockerfile: Dockerfile
  command: node app.js
  ports:
    - 3002:3002
  volumes:
    - ./User-App:/app
    - '/app/node_modules'
  restart: "unless-stopped"
  depends_on:
    - analytics-service
    - mongodb
```

Translation:

Same as the Amind-Web-App (app) above, the only difference is the port number which is 3002 for user_app instead of 3000 that was for Admin app. And it doesn't need to wait for mysql-server (analytical-service will handle the depends_on mysql server as we will see next)

DAalytics-Tool (analytics-service)

DockerFile

```
FROM python
WORKDIR /app
RUN pip install mysql-connector-python
RUN pip install pymongo
COPY . .
EXPOSE 3001
CMD [ "python3", "script.py" ]
```

Translation:

- Get an image of python from DockerHub
 - Inside this container go to directory /app
 - Then run the commands “pip install ...” (these commands are to download the libraries that was used in the python script to the container environment)
 - Then copy all files in this host directory to /app directory in the container (this line where the script goes to the container)
 - Then make this container exposed to port 3001
 - Finally run the command “python3 script.py” (this will execute the python code)
-

DockerCompose

```
analytics-service:
  build:
    context: './Analytics-Service'
    dockerfile: Dockerfile
  command: sh -c "sleep 4s ; python3 ./script.py"
  ports:
    - 3001:3001
  volumes:
    - ./Analytics-Service:/app
  restart: unless-stopped
  depends_on:
    - mysql_server
    - mongodb
```

Translation:

- This container is named *analytics-service*
 - Take the build constructions for this container using the file "DockerFile" in the directory ./Analytics-Service
 - Let the container's port analytical-service:3001 be exposed to localhost:3001 (useful to test the code)
 - Let the directory in the host machine "./Analytics-Service" be copied inside the container with the name "/app" instead.
 - After building this container start the command "sh -c ..." (This command will execute the python script after 4 seconds of waiting, we wait just to make sure that MySQL and mongodb databases took the time needed to be ready to connect to)
 - Whenever the container is closed restart it until it's being closed by user
 - Don't start this container until mysql_server, and mongodb are already built.
-

Authentication-Tool (authenticator)

DockerFile

```
FROM python
WORKDIR /app

COPY . .

EXPOSE 3005|

CMD [ "python3", "script.py" ]
```

Translation:

- Get an image of python from DockerHub
- Inside this container go to directory /app
- Then copy all files in this host directory to /app directory in the container (this line where the script goes to the container)
- Then make this container exposed to port 3005
- Finally run the command "python3 script.py" (this will execute the python code)

DockerCompose

```
authenticator:
  build:
    context: './Authentication'
    dockerfile: Dockerfile
  command: python3 ./script.py
  ports:
    - 3005:3005
  volumes:
    - ./Authentication:/app
  restart: unless-stopped
```

Translation:

- This container is named *analytics-service*
- Take the build constructions for this container using the file “Dockerfile” in the directory ./Authentication
- Let the container’s port analytical-service:3005 be exposed to localhost:3005 (useful to test the code)
- Let the directory in the host machine “./Authentication” be copied inside the container with the name “/app” instead.
- After building this container start the command “python3 ...” (This command will execute the python)
- Whenever the container is closed restart it until it’s being closed by user