# Setup for assignment 2 on MongoDB

This assignment can be performed on your local machines, i.e., without any need for using Google Cloud resources. The exercise is tested on Mac with Apple Silicon, and Ubuntu operating systems but not on Windows. If you are using Windows, or are otherwise unsure, you can use an `e2-micro` VM instance on Google Cloud Platform with an Ubuntu 24.04 LTS image for this assignment.

You are provided the following files:

- A Docker compose file to run MongoDB (`docker-compose.yml`, `Dockerfile`)
- A script containing shell commands (`exercise2_script.sh`)
- Code to run scenario 3 (`scenario3.py`)

In general, the sequence of tasks for the scenarios in this exercise will involve the following:

- Start MongoDB using the provided Docker compose file
- Start monitoring the nodes
- Perform actions as per the scenario - i.e., introduce faults, perform writes, perform reads, etc.
- Explain your observations and answer related questions

> ⚠️ It is mandatory to use Python and PyMongo for this exercise

## Installing Docker

First, install Docker on your machine. You can install Docker Desktop on Mac or Ubuntu machines with a graphical interface.

On an Ubuntu VM without a GUI, you can install Docker Engine by following the steps here. On the Ubuntu VM, you also need to follow steps 1 to 4 of the post-install steps.

## Setting up Docker compose

The Docker compose setup creates three containers of MongoDB with names `mongo1`, `mongo2`, and `mongo3`. In order to be able to reach the containers from your machine or VM, add the following entries to the `/etc/hosts` file following the steps below on your terminal.

```
# Add entries to /etc/hosts
sudo sh -c 'echo "127.0.0.1 mongo1" >> /etc/hosts'
sudo sh -c 'echo "127.0.0.1 mongo2" >> /etc/hosts'
sudo sh -c 'echo "127.0.0.1 mongo3" >> /etc/hosts'
```

Run the following command from within the directory where the `Dockerfile` and `docker-compose.yml` files are stored.

```
# Start docker compose
docker compose up -d
```

The first time you run this command, you should see a similar output as below. In particular, pay attention to the last few lines of the output, i.e., the network, volumes, and containers `mongo1`, `mongo2` and `mongo3` are started.

```
[+] Running 0/0
[+] Building 40.3s (8/10)
...
[+] Running 13/13to docker.io/library/exercise2_files-mongo2:latest
0.1s
 ✔ Service mongo3                          Built
89.8s
 ✔ Service mongo2                          Built
89.8s
 ✔ Service mongo1                          Built
89.8s
 ✔ Network mongo_network                   Created
0.0s
 ✔ Volume "exercise2_files_mongo1_data"    Created
0.0s
 ✔ Volume "exercise2_files_mongo1_config"  Created
0.0s
 ✔ Volume "exercise2_files_mongo2_data"    Created
0.0s
 ✔ Volume "exercise2_files_mongo2_config"  Created
0.0s
 ✔ Volume "exercise2_files_mongo3_data"    Created
0.0s
 ✔ Volume "exercise2_files_mongo3_config"  Created
0.0s
 ✔ Container mongo1                        Started
0.4s
 ✔ Container mongo2                        Started
0.4s
 ✔ Container mongo3                        Started
0.4s
```

You can stop the containers, using the following command:

```
docker compose down -v
```

You should see the following output after running the command. This indicates that all the containers are stopped correctly and removed. This enables you to start from a clean environment again.

```
[+] Running 10/3
✔ Container mongo1                Removed
0.8s
✔ Container mongo2                Removed
0.8s
✔ Container mongo3                Removed
0.7s
```

```
✔ Volume exercise2_mongo1_config  Removed
0.0s
✔ Volume exercise2_mongo2_config  Removed
0.0s
✔ Volume exercise2_mongo3_config  Removed
0.0s
✔ Volume exercise2_mongo2_data    Removed
0.0s
✔ Volume exercise2_mongo1_data    Removed
0.0s
✔ Volume exercise2_mongo3_data    Removed
0.0s
✔ Network mongo_network           Removed
0.1s
```

## Script with common functions

We provide a script `exercise2_script.sh` with functions to monitor the status of the mongoDB nodes, create a replica set of the three mongoDB containers, and modify the network between the containers.

You will make use of these functions while running the scenarios described in the exercise. There is no need to modify the script.

To use the script, on your terminal, run the following command where the file is stored.

```
source exercise2_script.sh
```

To execute one of the functions within the script, first start your mongoDB containers using the Docker compose command.

```
# Start your environment
docker compose up -d

# Run one of the functions from the script
e2_rs_status_getall
```

Since we haven't actually created a replica set, you should get the following output:

```
Status on mongo1:
MongoServerError: no replset config has been received
Status on mongo2:
MongoServerError: no replset config has been received
Status on mongo3:
MongoServerError: no replset config has been received
```

## Documentation for Python code in scenario 3

The provided Python code (`scenario3.py`) is needed to run scenario 3. You do not need to modify the code for scenario 3.

Three classes are provided:

- A `MongoClientFactory` class to create instances of PyMongo MongoClient

- A `MongoWriter` class with methods to write to MongoDB
- A `MongoReader` class with methods to read to MongoDB

**Setting up the Python environment:**

```
python3 -m venv venv
source venv/bin/activate
pip install pymongo
```

**Initializing clients:**

Two types of clients are needed for the exercise:

- **Direct clients**: they connect exactly to one MongoDB node and are unaware of replica set configuration.
- **Replicaset clients:** they are aware of the replicaset configuration and connect to all three nodes.

```
MONGODB_RS_URI = "mongodb://mongo1:27017,mongo2:27018,mongo3:27019/"
MONGODB1_URI = "mongodb://mongo1:27017/"

client_replicaset = MongoClientFactory.create_client(
    "replica",
    MONGODB_RS_URI
)

client_mongo1_direct = MongoClientFactory.create_client(
    "direct",
    MONGODB1_URI
)
```

**Reading and writing:**

The readers and writers require an instance of `MongoClient`, and MongoDB collection information.

```
writer = MongoWriter(client_replicaset, DB_NAME, COLLECTION_NAME)
reader = MongoReader("client_mongo1_direct", client_mongo1_direct, DB_NAME, COLL
ECTION_NAME)
```

Reads and writes can be performed as below:

```
duration = writer.write_document(data, w=1)
duration = reader.read_documents(read_concern="local")
```

Both `write_document` and `read_documents` log useful information, and return how long the operation takes to complete. This can be useful for developing code required for scenario 2.