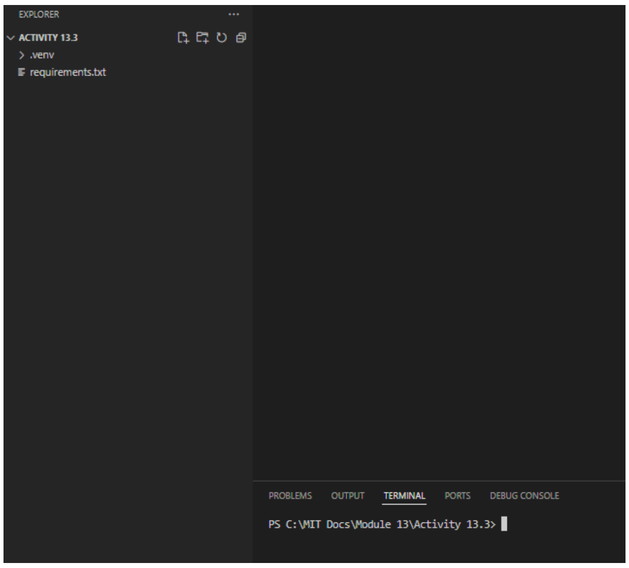
Activity 13.3: Performing CDC and Initializing MySQL and MongoDB Containers

Miguel Brito

1. Creating a folder called "Activity 13.3" and opening in on VS Code



```
PS C:\MIT Docs\Module 13\Activity 13.3>
```

2. Creating the "container.py" file

   I feel the need to explain myself here as I kept the same functionalities that were initially presented in the platform, but I went for a different approach to achieve the same goal.

   In this case, I created a container class using the "builder" design pattern, which consists of having a series of classes inside a file that can be called whenever I need for something to happen or to be "built".

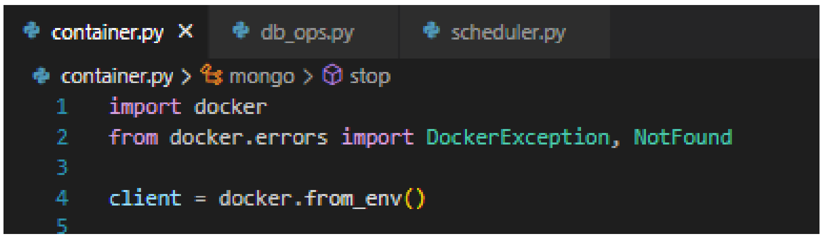
   Each type of container has three methods, which are:

   - __init__: Used to initialize a class of the type of container that I need.
   - Create: Used to create a container with a determined database type.
   - Stop: It stops the container when I don't need it anymore.
   - Delete: It's in charge of deleting the container that was created previously.

Hence, when applying this logic, we have a blueprint that can be replicated with all containers, making it easier to manage the logic in the program and to understand what each line of code does.

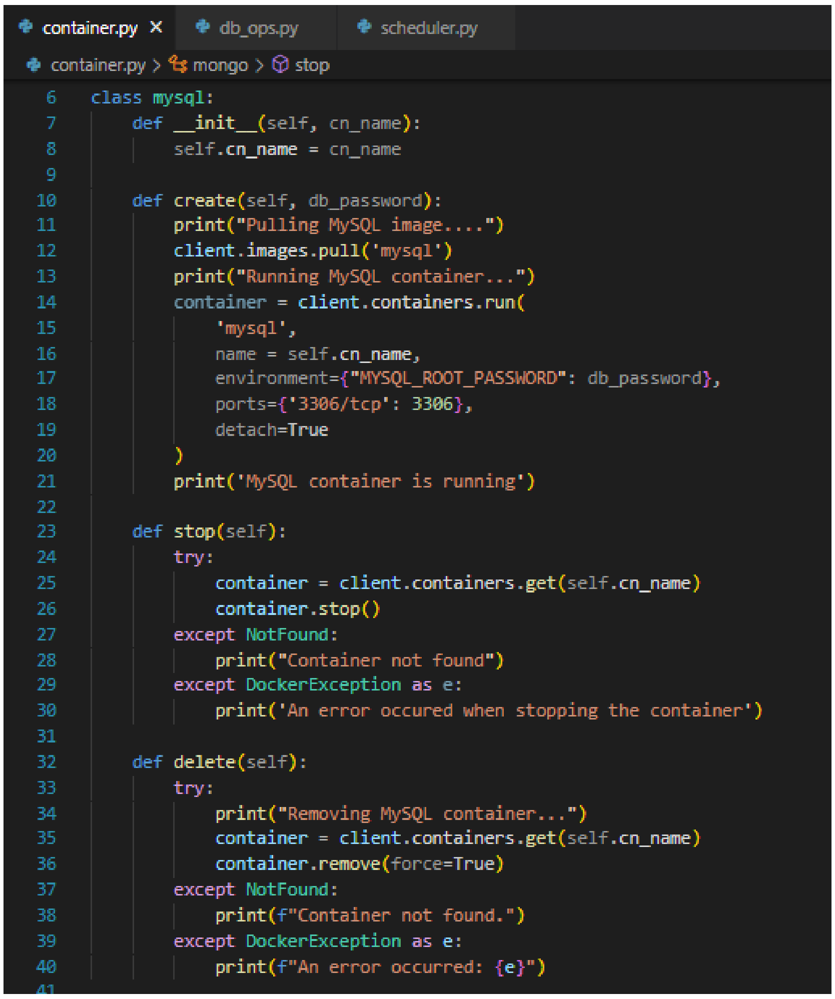
Now that this has been explained, I hereby add the screenshots of the "container.py" file:

Prerequisites:

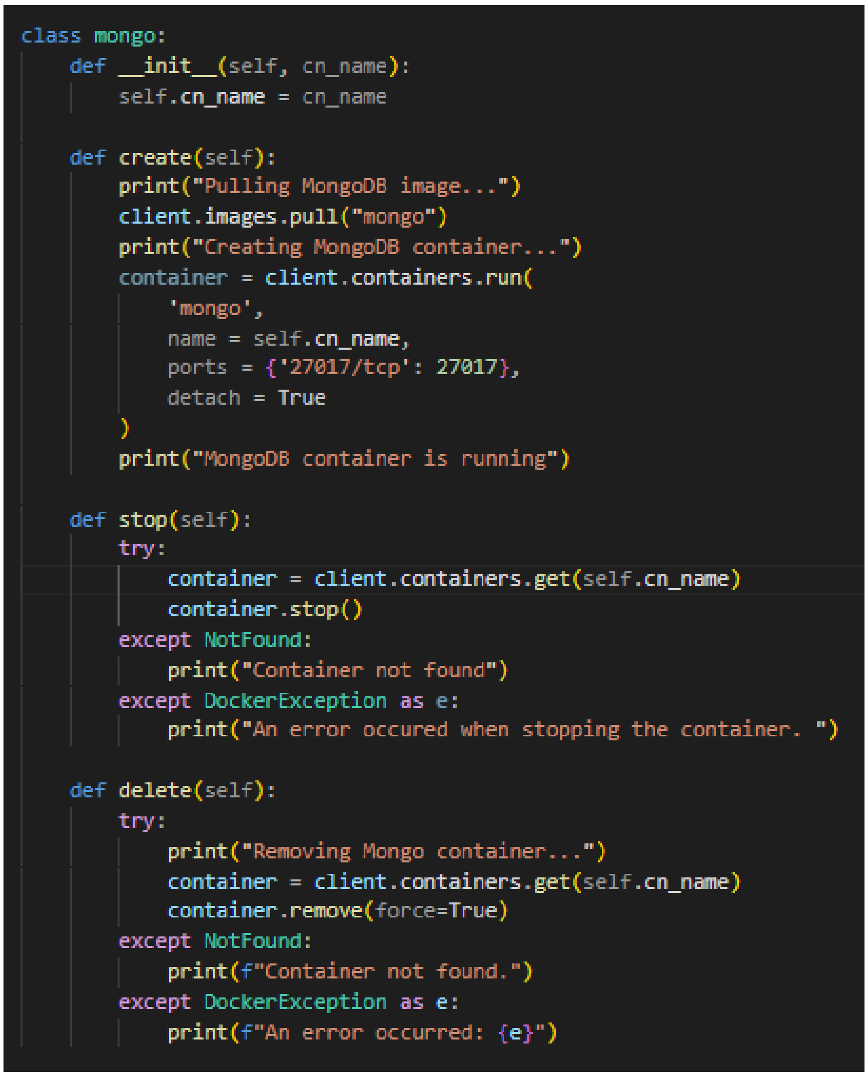


```python
import docker
from docker.errors import DockerException, NotFound

client = docker.from_env()
```

Classes:



```python
class mysql:
    def __init__(self, cn_name):
        self.cn_name = cn_name

    def create(self, db_password):
        print("Pulling MySQL image....")
        client.images.pull('mysql')
        print("Running MySQL container...")
        container = client.containers.run(
            'mysql',
            name = self.cn_name,
            environment={"MYSQL_ROOT_PASSWORD": db_password},
            ports={'3306/tcp': 3306},
            detach=True
        )
        print('MySQL container is running')

    def stop(self):
        try:
            container = client.containers.get(self.cn_name)
            container.stop()
        except NotFound:
            print("Container not found")
        except DockerException as e:
            print('An error occured when stopping the container')

    def delete(self):
        try:
            print("Removing MySQL container...")
            container = client.containers.get(self.cn_name)
            container.remove(force=True)
        except NotFound:
            print(f"Container not found.")
        except DockerException as e:
            print(f"An error occurred: {e}")
```



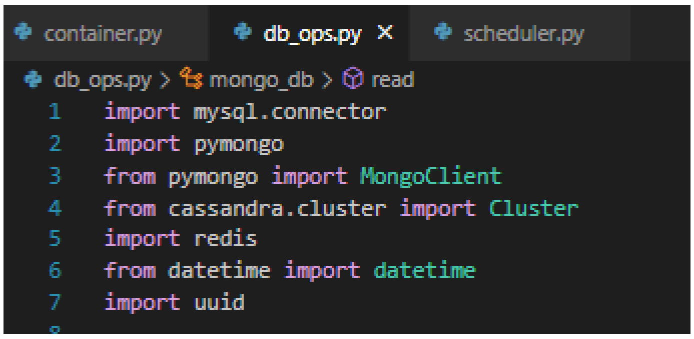
```python
class mongo:
    def __init__(self, cn_name):
        self.cn_name = cn_name

    def create(self):
        print("Pulling MongoDB image...")
        client.images.pull("mongo")
        print("Creating MongoDB container...")
        container = client.containers.run(
            'mongo',
            name = self.cn_name,
            ports = {'27017/tcp': 27017},
            detach = True
        )
        print("MongoDB container is running")

    def stop(self):
        try:
            container = client.containers.get(self.cn_name)
            container.stop()
        except NotFound:
            print("Container not found")
        except DockerException as e:
            print("An error occured when stopping the container. ")

    def delete(self):
        try:
            print("Removing Mongo container...")
            container = client.containers.get(self.cn_name)
            container.remove(force=True)
        except NotFound:
            print(f"Container not found.")
        except DockerException as e:
            print(f"An error occurred: {e}")
```

I also created another file called "db_ops.py" which oversees managing the databases together with all the available operations for each one of them. These methods are:

- __init__: Used to initialize a database connection when needed.
- Create: It's used to create the database itself, except for the case of the "mongo_db" class, which creates a collection as soon as the __init__ method is initialized.
- Write: It's used to add registries to the database.
- Read: It's used to display the last 5 added registries in the database.

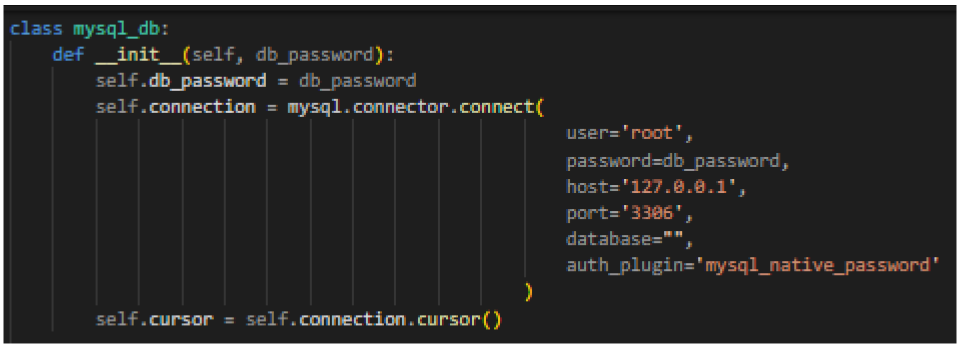
I hereby add screenshots of the file for the logic to be understood better:

Prerequisites:

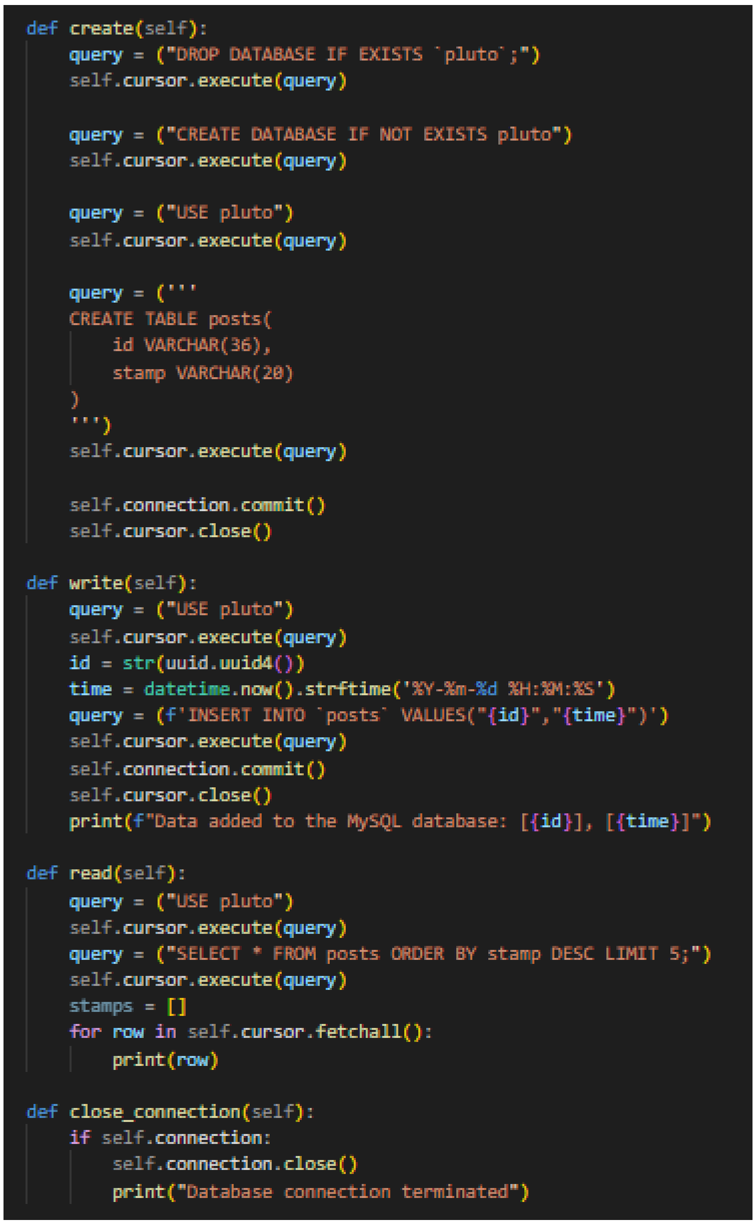


```python
import mysql.connector
import pymongo
from pymongo import MongoClient
from cassandra.cluster import Cluster
import redis
from datetime import datetime
import uuid
```

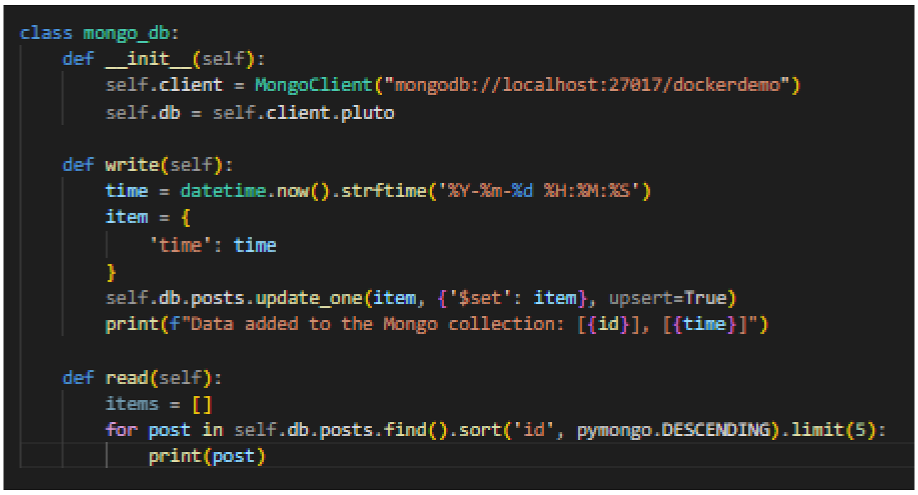
Classes:



```python
class mysql_db:
    def __init__(self, db_password):
        self.db_password = db_password
        self.connection = mysql.connector.connect(
                                            user='root',
                                            password=db_password,
                                            host='127.0.0.1',
                                            port='3306',
                                            database="",
                                            auth_plugin='mysql_native_password'
                                        )
        self.cursor = self.connection.cursor()
```

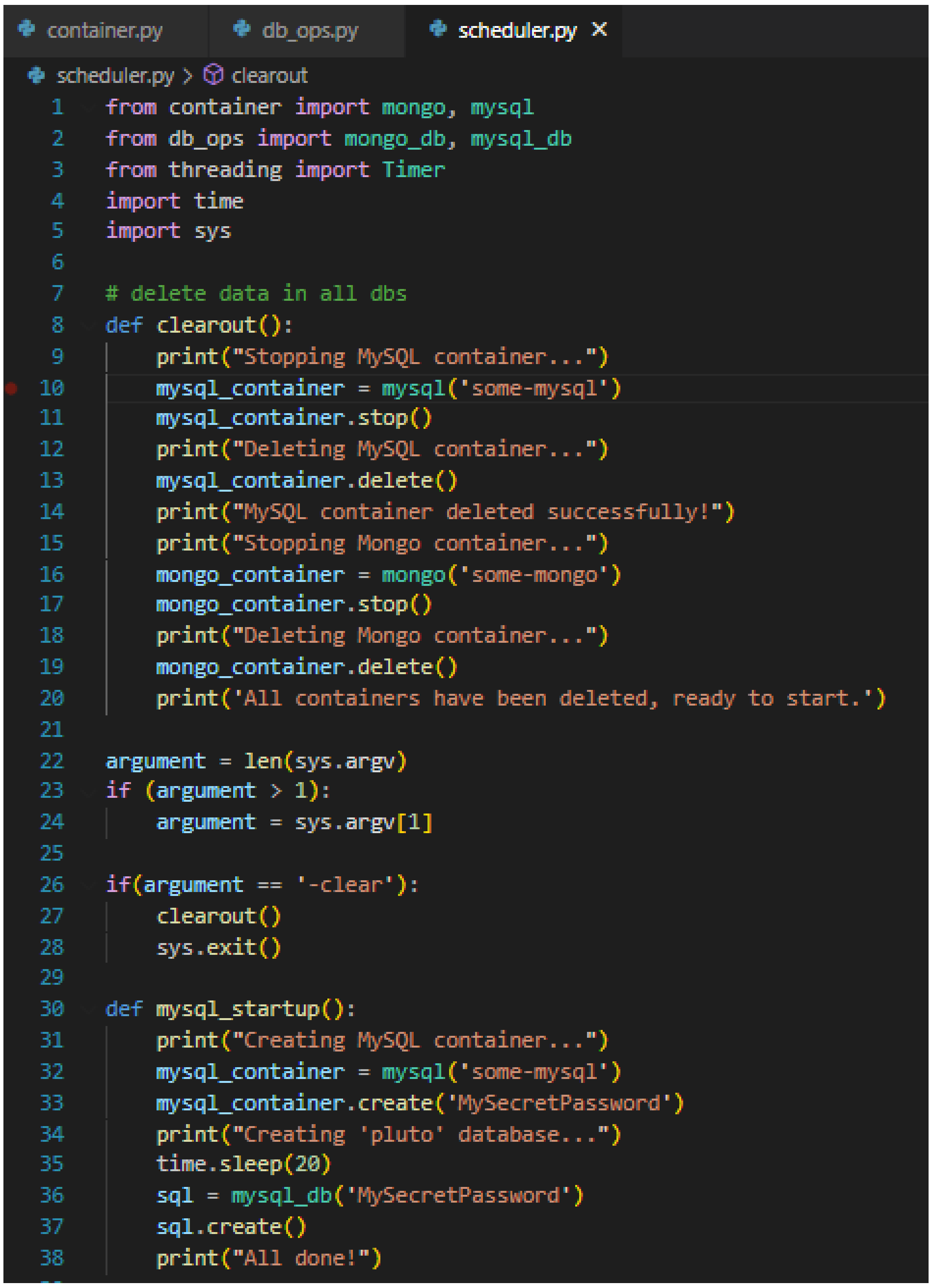


```python
def create(self):
    query = ("DROP DATABASE IF EXISTS `pluto`;")
    self.cursor.execute(query)

    query = ("CREATE DATABASE IF NOT EXISTS pluto")
    self.cursor.execute(query)

    query = ("USE pluto")
    self.cursor.execute(query)

    query = ('''
    CREATE TABLE posts(
        id VARCHAR(36),
        stamp VARCHAR(20)
    )
    ''')
    self.cursor.execute(query)

    self.connection.commit()
    self.cursor.close()

def write(self):
    query = ("USE pluto")
    self.cursor.execute(query)
    id = str(uuid.uuid4())
    time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
    query = (f'INSERT INTO `posts` VALUES("{id}","{time}")')
    self.cursor.execute(query)
    self.connection.commit()
    self.cursor.close()
    print(f"Data added to the MySQL database: [{id}], [{time}]")

def read(self):
    query = ("USE pluto")
    self.cursor.execute(query)
    query = ("SELECT * FROM posts ORDER BY stamp DESC LIMIT 5;")
    self.cursor.execute(query)
    stamps = []
    for row in self.cursor.fetchall():
        print(row)

def close_connection(self):
    if self.connection:
        self.connection.close()
        print("Database connection terminated")
```

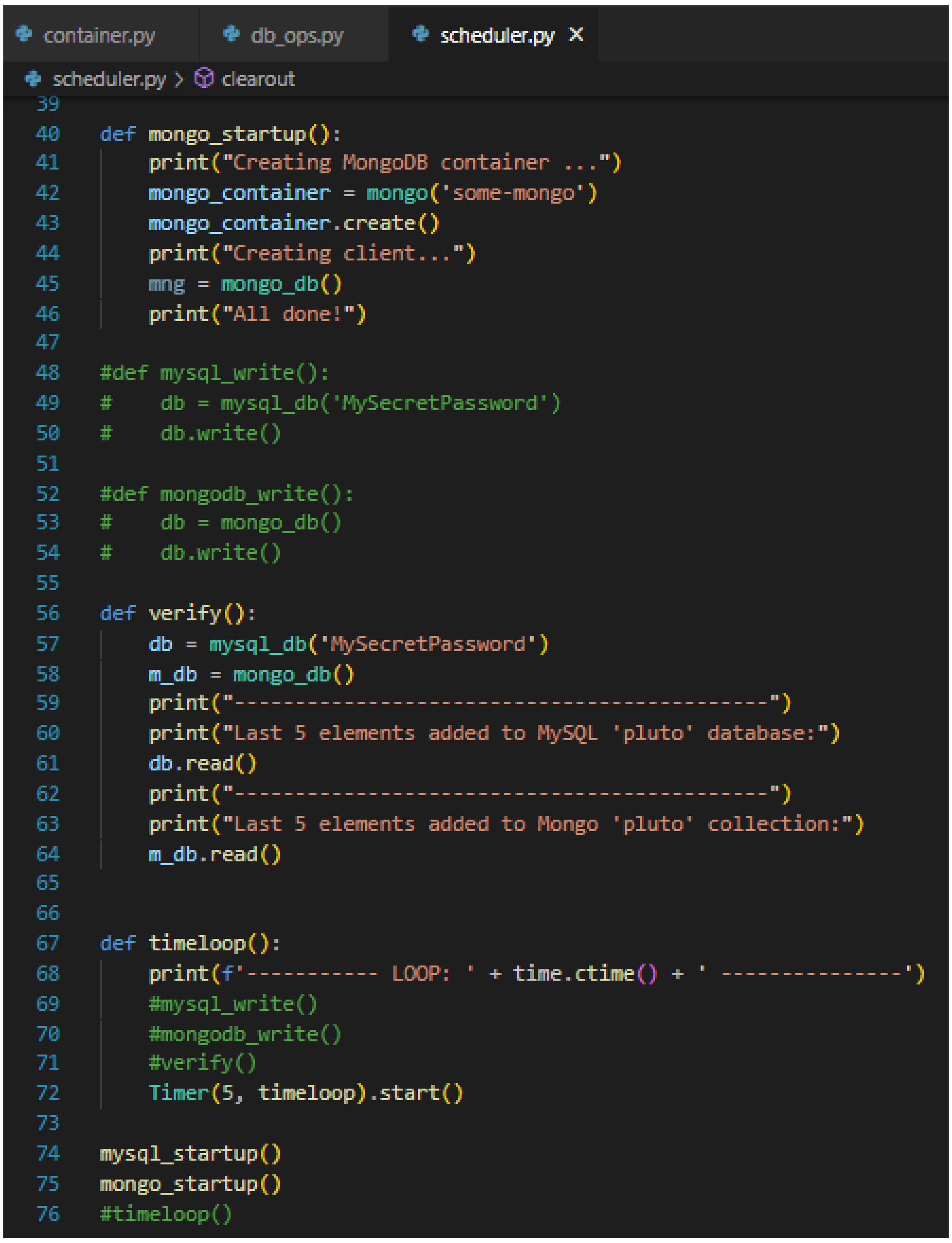


```python
class mongo_db:
    def __init__(self):
        self.client = MongoClient("mongodb://localhost:27017/dockerdemo")
        self.db = self.client.pluto

    def write(self):
        time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        item = {
            'time': time
        }
        self.db.posts.update_one(item, {'$set': item}, upsert=True)
        print(f"Data added to the Mongo collection: [{id}], [{time}]")

    def read(self):
        items = []
        for post in self.db.posts.find().sort('id', pymongo.DESCENDING).limit(5):
            print(post)
```

Once this is done, I can modify the “scheduler.py” file to call these class methods and create all the containers needed for this activity.



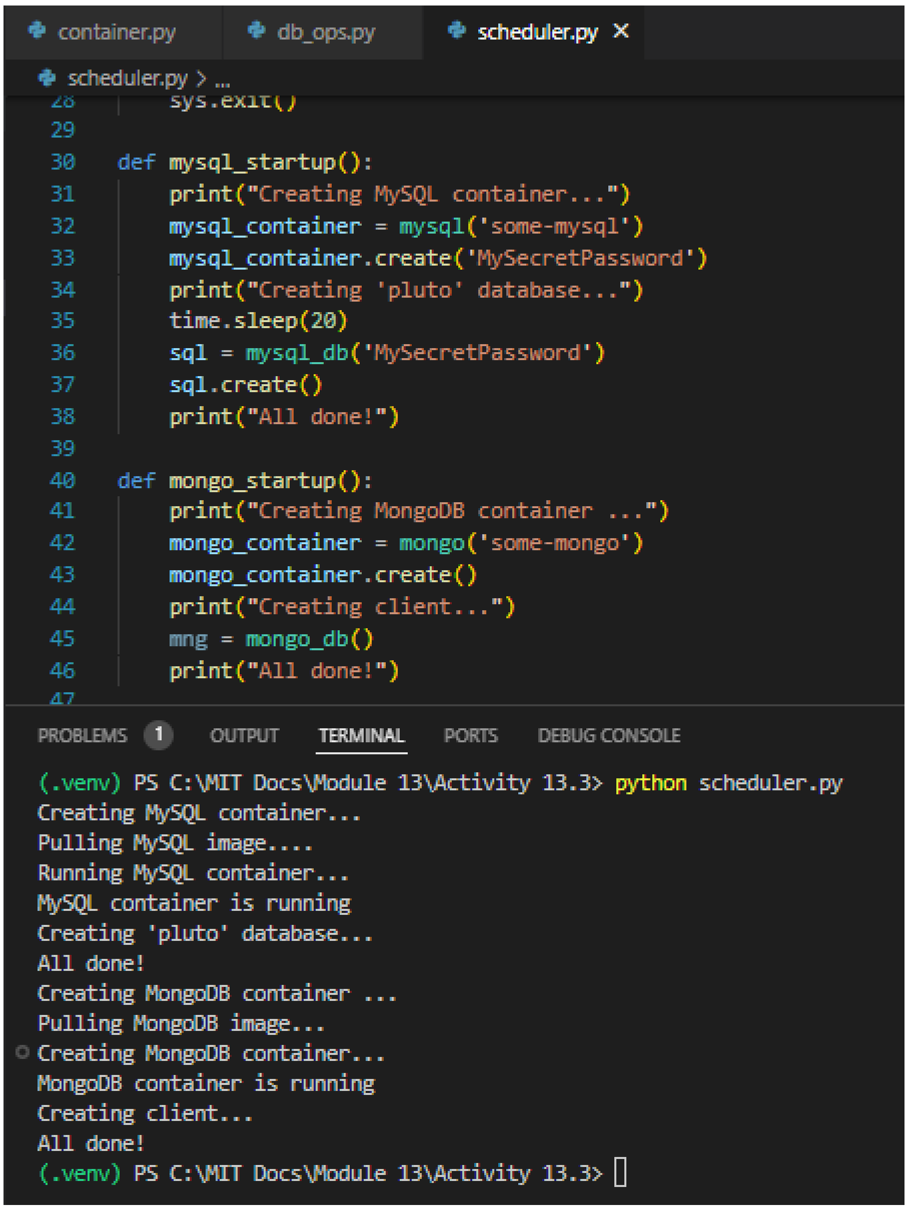
```python
from container import mongo, mysql
from db_ops import mongo_db, mysql_db
from threading import Timer
import time
import sys

# delete data in all dbs
def clearout():
    print("Stopping MySQL container...")
    mysql_container = mysql('some-mysql')
    mysql_container.stop()
    print("Deleting MySQL container...")
    mysql_container.delete()
    print("MySQL container deleted successfully!")
    print("Stopping Mongo container...")
    mongo_container = mongo('some-mongo')
    mongo_container.stop()
    print("Deleting Mongo container...")
    mongo_container.delete()
    print('All containers have been deleted, ready to start.')

argument = len(sys.argv)
if (argument > 1):
    argument = sys.argv[1]

if(argument == '-clear'):
    clearout()
    sys.exit()

def mysql_startup():
    print("Creating MySQL container...")
    mysql_container = mysql('some-mysql')
    mysql_container.create('MySecretPassword')
    print("Creating 'pluto' database...")
    time.sleep(20)
    sql = mysql_db('MySecretPassword')
    sql.create()
    print("All done!")
```



```python

def mongo_startup():
    print("Creating MongoDB container ...")
    mongo_container = mongo('some-mongo')
    mongo_container.create()
    print("Creating client...")
    mng = mongo_db()
    print("All done!")

#def mysql_write():
#    db = mysql_db('MySecretPassword')
#    db.write()

#def mongodb_write():
#    db = mongo_db()
#    db.write()

def verify():
    db = mysql_db('MySecretPassword')
    m_db = mongo_db()
    print("--------------------------------------------")
    print("Last 5 elements added to MySQL 'pluto' database:")
    db.read()
    print("--------------------------------------------")
    print("Last 5 elements added to Mongo 'pluto' collection:")
    m_db.read()


def timeloop():
    print(f'----------- LOOP: ' + time.ctime() + ' --------------')
    #mysql_write()
    #mongodb_write()
    #verify()
    Timer(5, timeloop).start()

mysql_startup()
mongo_startup()
#timeloop()
```

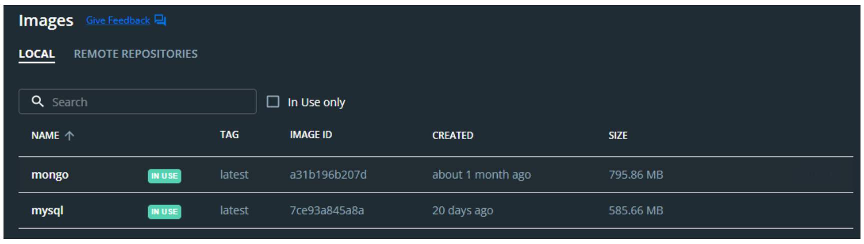
3. Displaying the execution results of the file and showing the recently created containers

   With the code commented, I only limit the scheduler to create the containers together with the databases. The result of the executed code can be seen here:

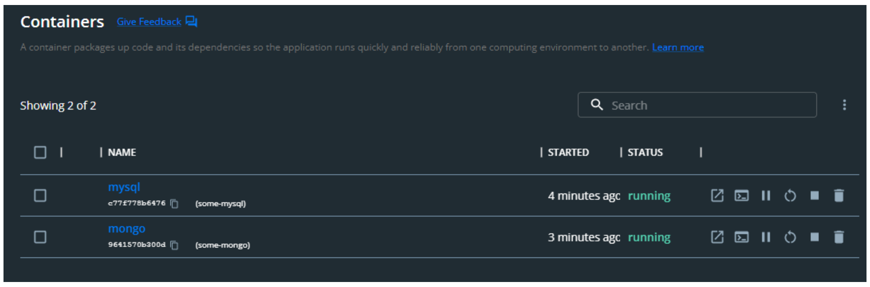


```python
    sys.exit()

def mysql_startup():
    print("Creating MySQL container...")
    mysql_container = mysql('some-mysql')
    mysql_container.create('MySecretPassword')
    print("Creating 'pluto' database...")
    time.sleep(20)
    sql = mysql_db('MySecretPassword')
    sql.create()
    print("All done!")

def mongo_startup():
    print("Creating MongoDB container ...")
    mongo_container = mongo('some-mongo')
    mongo_container.create()
    print("Creating client...")
    mng = mongo_db()
    print("All done!")
```

```
(.venv) PS C:\MIT Docs\Module 13\Activity 13.3> python scheduler.py
Creating MySQL container...
Pulling MySQL image....
Running MySQL container...
MySQL container is running
Creating 'pluto' database...
All done!
Creating MongoDB container ...
Pulling MongoDB image...
Creating MongoDB container...
MongoDB container is running
Creating client...
All done!
(.venv) PS C:\MIT Docs\Module 13\Activity 13.3>
```


Images

NAME		TAG	IMAGE ID	CREATED	SIZE
mongo	IN USE	latest	a31b196b207d	about 1 month ago	795.86 MB
mysql	IN USE	latest	7ce93a845a8a	20 days ago	585.66 MB


Containers

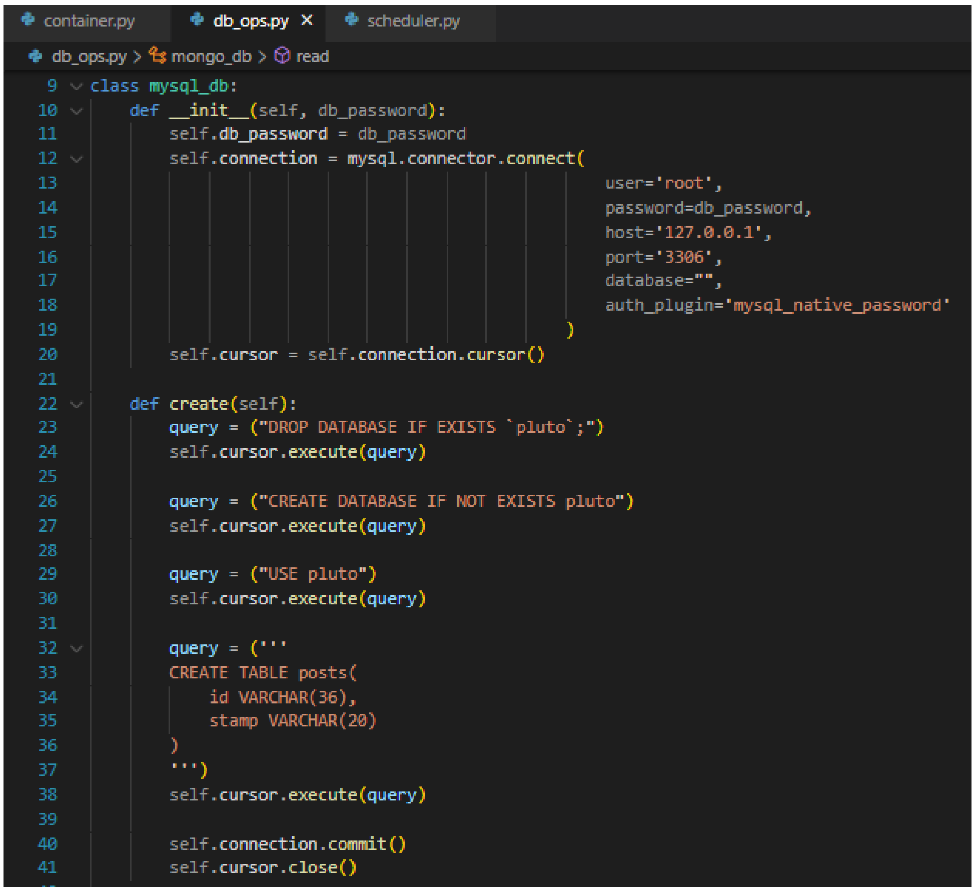
A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 of 2

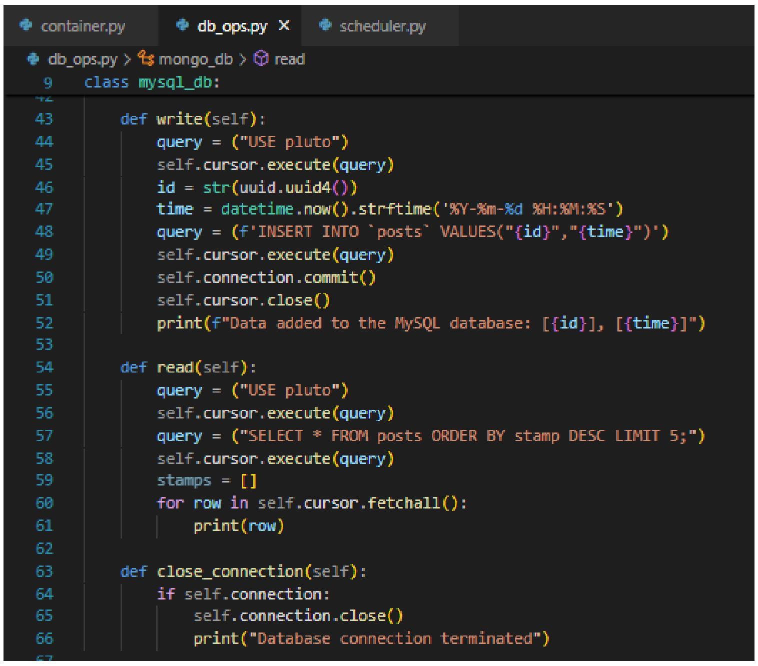
NAME	STARTED	STATUS
mysql c77f778b6476 (some-mysql)	4 minutes agc	running
mongo 9641570b300d (some-mongo)	3 minutes agc	running

4. Creating the “mysqldb.py” file to stablish connections to the mysql database.

   As I said before, I created a class in the case of existence of a “MySQL” database.



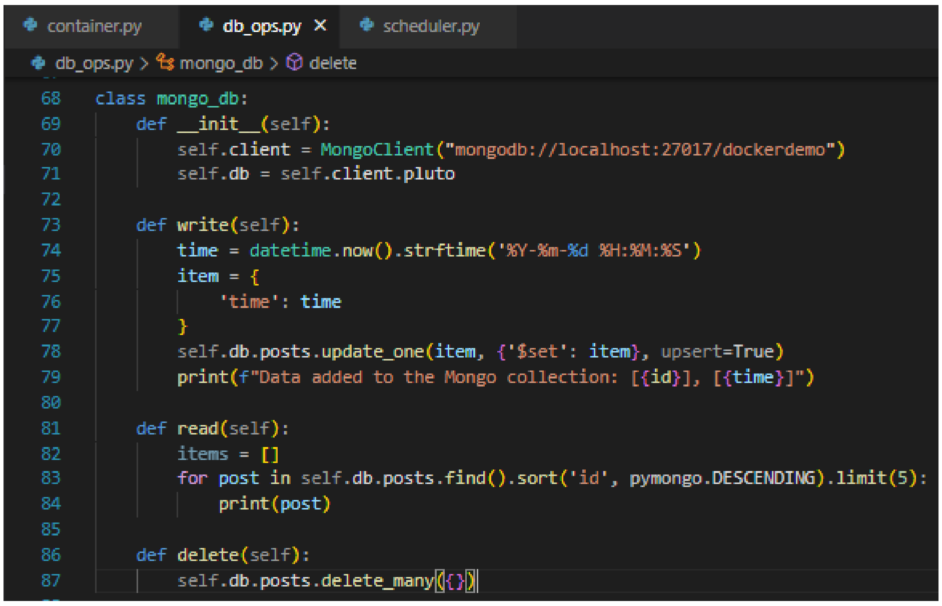
```python
class mysql_db:
    def __init__(self, db_password):
        self.db_password = db_password
        self.connection = mysql.connector.connect(
                                            user='root',
                                            password=db_password,
                                            host='127.0.0.1',
                                            port='3306',
                                            database="",
                                            auth_plugin='mysql_native_password'
                                           )
        self.cursor = self.connection.cursor()

    def create(self):
        query = ("DROP DATABASE IF EXISTS `pluto`;")
        self.cursor.execute(query)

        query = ("CREATE DATABASE IF NOT EXISTS pluto")
        self.cursor.execute(query)

        query = ("USE pluto")
        self.cursor.execute(query)

        query = ('''
        CREATE TABLE posts(
            id VARCHAR(36),
            stamp VARCHAR(20)
        )
        ''')
        self.cursor.execute(query)

        self.connection.commit()
        self.cursor.close()
```



```python
class mysql_db:

    def write(self):
        query = ("USE pluto")
        self.cursor.execute(query)
        id = str(uuid.uuid4())
        time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        query = (f'INSERT INTO `posts` VALUES("{id}","{time}")')
        self.cursor.execute(query)
        self.connection.commit()
        self.cursor.close()
        print(f"Data added to the MySQL database: [{id}], [{time}]")

    def read(self):
        query = ("USE pluto")
        self.cursor.execute(query)
        query = ("SELECT * FROM posts ORDER BY stamp DESC LIMIT 5;")
        self.cursor.execute(query)
        stamps = []
        for row in self.cursor.fetchall():
            print(row)

    def close_connection(self):
        if self.connection:
            self.connection.close()
            print("Database connection terminated")
```

5. Creating the “mongodb.py” file to stablish connections to the mongo database.

   Just like with the mysql database, I created a class to contain all the possible operations to be applied to a mongo database.

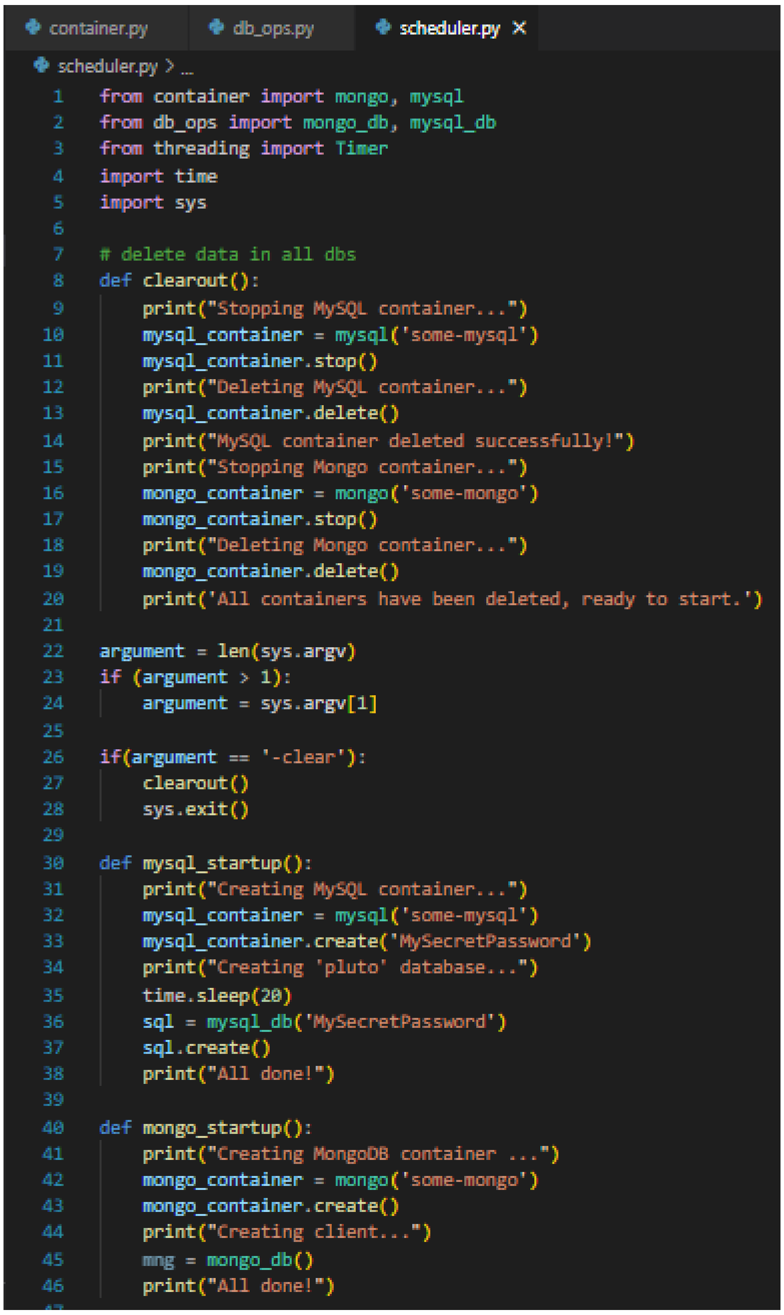


```python
class mongo_db:
    def __init__(self):
        self.client = MongoClient("mongodb://localhost:27017/dockerdemo")
        self.db = self.client.pluto

    def write(self):
        time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        item = {
            'time': time
        }
        self.db.posts.update_one(item, {'$set': item}, upsert=True)
        print(f"Data added to the Mongo collection: [{id}], [{time}]")

    def read(self):
        items = []
        for post in self.db.posts.find().sort('id', pymongo.DESCENDING).limit(5):
            print(post)

    def delete(self):
        self.db.posts.delete_many({})
```

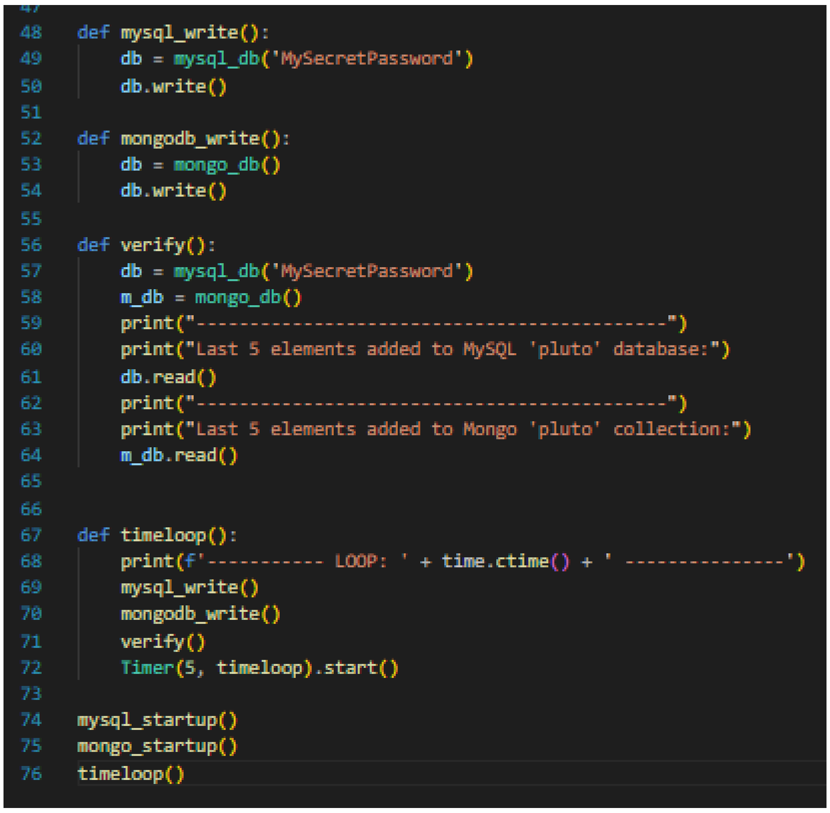
6. Add the “scheduler.py” file to the project.

   As I said before, I created a modified version of the “scheduler.py” file to simplify the ways in which both the databases and containers are managed so it looks very different to the one presented in the MIT xPRO platform, however it keeps all the functionalities that were initially defined for this class.

   The only difference is that instead of reseting the databases in the “clearout()” method I stop and delete the containers instead, as it is more friendly with the structure of classes previously defined and because there isn’t a dire need to keep the databases’ existence.



```python
from container import mongo, mysql
from db_ops import mongo_db, mysql_db
from threading import Timer
import time
import sys

# delete data in all dbs
def clearout():
    print("Stopping MySQL container...")
    mysql_container = mysql('some-mysql')
    mysql_container.stop()
    print("Deleting MySQL container...")
    mysql_container.delete()
    print("MySQL container deleted successfully!")
    print("Stopping Mongo container...")
    mongo_container = mongo('some-mongo')
    mongo_container.stop()
    print("Deleting Mongo container...")
    mongo_container.delete()
    print('All containers have been deleted, ready to start.')

argument = len(sys.argv)
if (argument > 1):
    argument = sys.argv[1]

if(argument == '-clear'):
    clearout()
    sys.exit()

def mysql_startup():
    print("Creating MySQL container...")
    mysql_container = mysql('some-mysql')
    mysql_container.create('MySecretPassword')
    print("Creating 'pluto' database...")
    time.sleep(20)
    sql = mysql_db('MySecretPassword')
    sql.create()
    print("All done!")

def mongo_startup():
    print("Creating MongoDB container ...")
    mongo_container = mongo('some-mongo')
    mongo_container.create()
    print("Creating client...")
    mng = mongo_db()
    print("All done!")
```

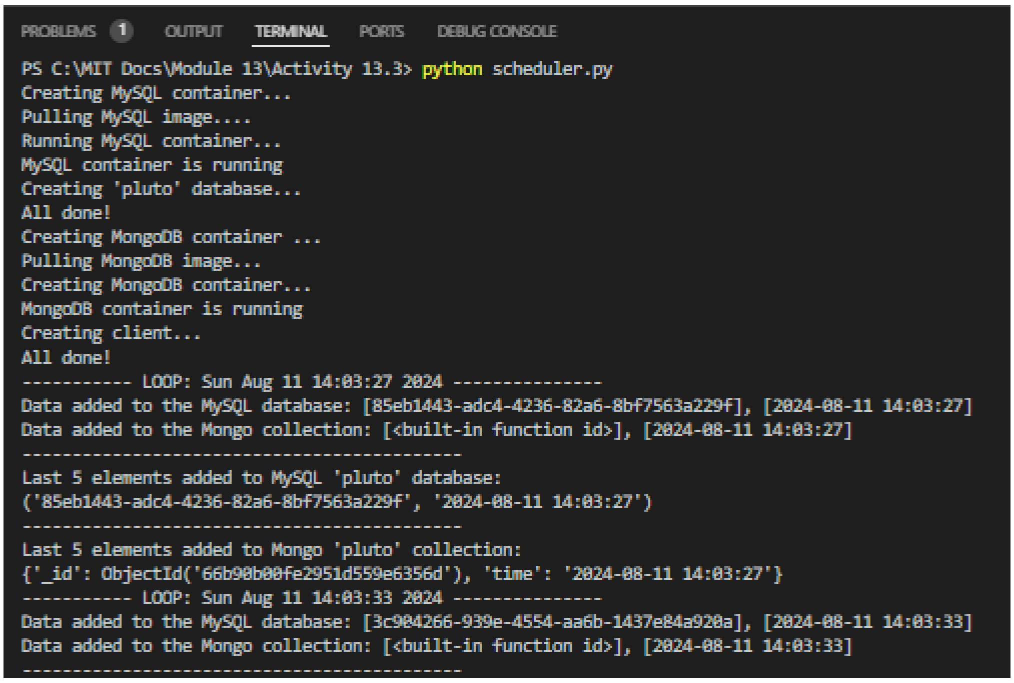


```python
def mysql_write():
    db = mysql_db('MySecretPassword')
    db.write()

def mongodb_write():
    db = mongo_db()
    db.write()

def verify():
    db = mysql_db('MySecretPassword')
    m_db = mongo_db()
    print("--------------------------------------------")
    print("Last 5 elements added to MySQL 'pluto' database:")
    db.read()
    print("--------------------------------------------")
    print("Last 5 elements added to Mongo 'pluto' collection:")
    m_db.read()


def timeloop():
    print(f'---------- LOOP: ' + time.ctime() + ' --------------')
    mysql_write()
    mongodb_write()
    verify()
    Timer(5, timeloop).start()

mysql_startup()
mongo_startup()
timeloop()
```

7. Running the “container.py” and “scheduler.py” files to make sure everything works well.

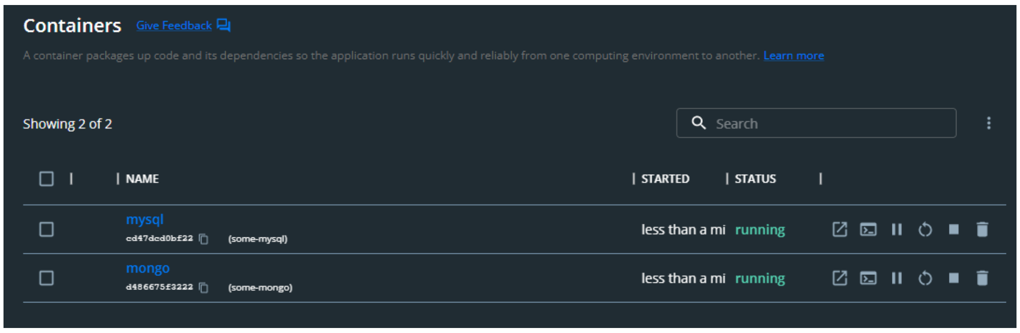
   As the “container.py” file only contains class methods now, it can’t be directly executed; however, by declaring an object and calling the class methods from the “scheduler.py” file, we can achieve this as desired. However, because the “scheduler.py” file is responsible of creating the containers and database operations now, the screenshot in part 8 is going to provide screenshots that demonstrate that everything in part 7 was also executed correctly.

8. Running the “scheduler.py” and showing it working correctly.



PROBLEMS 1 OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
PS C:\MIT Docs\Module 13\Activity 13.3> python scheduler.py
Creating MySQL container...
Pulling MySQL image....
Running MySQL container...
MySQL container is running
Creating 'pluto' database...
All done!
Creating MongoDB container ...
Pulling MongoDB image...
Creating MongoDB container...
MongoDB container is running
Creating client...
All done!
----------- LOOP: Sun Aug 11 14:03:27 2024 ---------------
Data added to the MySQL database: [85eb1443-adc4-4236-82a6-8bf7563a229f], [2024-08-11 14:03:27]
Data added to the Mongo collection: [<built-in function id>], [2024-08-11 14:03:27]
---------------------------------------------
Last 5 elements added to MySQL 'pluto' database:
('85eb1443-adc4-4236-82a6-8bf7563a229f', '2024-08-11 14:03:27')
---------------------------------------------
Last 5 elements added to Mongo 'pluto' collection:
{'_id': ObjectId('66b90b00fe2951d559e6356d'), 'time': '2024-08-11 14:03:27'}
----------- LOOP: Sun Aug 11 14:03:33 2024 ---------------
Data added to the MySQL database: [3c904266-939e-4554-aa6b-1437e84a920a], [2024-08-11 14:03:33]
Data added to the Mongo collection: [<built-in function id>], [2024-08-11 14:03:33]
---------------------------------------------
```



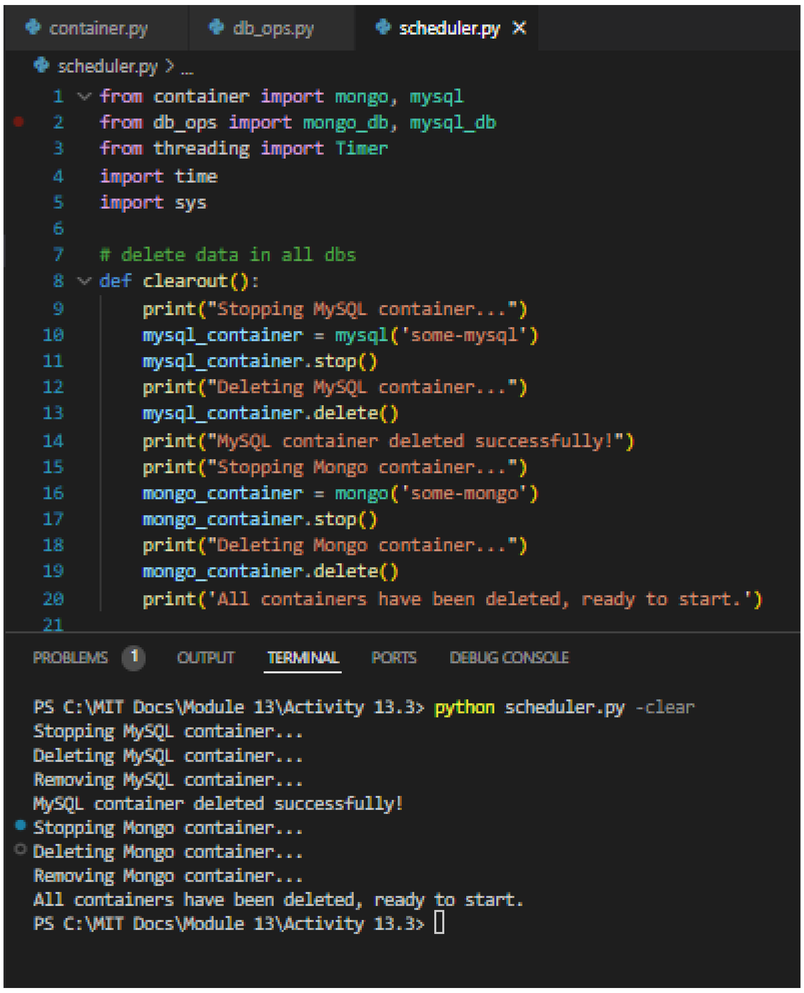
Containers Give Feedback

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Showing 2 of 2

NAME	STARTED	STATUS
mysql cd47dcd0bf22 (some-mysql)	less than a mi	running
mongo d486675f3222 (some-mongo)	less than a mi	running

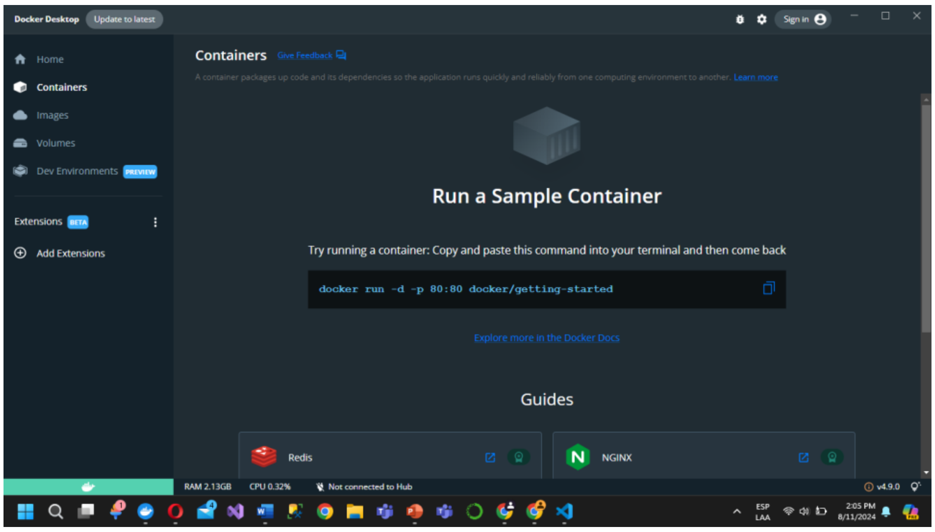
9. Stop the scheduler.py program by closing the command prompt window. Run the container.py command from the command prompt passing -delete as a parameter. Provide a screenshot of the command prompt showing that you successfully ran the command.



```python
from container import mongo, mysql
from db_ops import mongo_db, mysql_db
from threading import Timer
import time
import sys

# delete data in all dbs
def clearout():
    print("Stopping MySQL container...")
    mysql_container = mysql('some-mysql')
    mysql_container.stop()
    print("Deleting MySQL container...")
    mysql_container.delete()
    print("MySQL container deleted successfully!")
    print("Stopping Mongo container...")
    mongo_container = mongo('some-mongo')
    mongo_container.stop()
    print("Deleting Mongo container...")
    mongo_container.delete()
    print('All containers have been deleted, ready to start.')

```

```
PS C:\MIT Docs\Module 13\Activity 13.3> python scheduler.py -clear
Stopping MySQL container...
Deleting MySQL container...
Removing MySQL container...
MySQL container deleted successfully!
Stopping Mongo container...
Deleting Mongo container...
Removing Mongo container...
All containers have been deleted, ready to start.
PS C:\MIT Docs\Module 13\Activity 13.3>
```

10. Displaying that the containers have been disposed of



Containers

A container packages up code and its dependencies so the application runs quickly and reliably from one computing environment to another. Learn more

Run a Sample Container

Try running a container: Copy and paste this command into your terminal and then come back

```
docker run -d -p 80:80 docker/getting-started
```

Explore more in the Docker Docs