Module 13 Final Assignment: Applying CDC to Databases on Different Platforms

Miguel Brito

1. Creating a folder for the final assignment with the name “Assignment 13”

   I’m still going to use the “Activity 13.3” folder as all the necessary logic is available there. At the end of this activity, I will add a link to GitHub so that the implementation that I used for this activity is available to be fully seen.

2. Displaying the logic used to create both “MySQL” and “MongoDB” containers and showing the code running to create them.

container.py | .gitignore | db_ops.py | scheduler.py

container.py > mysql

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

client = docker.from_env()

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:
            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}")
```

container.py | db_ops.py | scheduler.py

db_ops.py > ...

```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()

    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")
```

container.py | .gitignore | db_ops.py | scheduler.py

container.py > mysql

```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:
            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}")
```

container.py | db_ops.py | scheduler.py

db_ops.py > ...

```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({})
```

container.py .gitignore db_ops.py scheduler.py

scheduler.py > ...

```
42    def mysql_startup():
43        print("Creating MySQL container...")
44        mysql_container = mysql('some-mysql')
45        mysql_container.create('MySecretPassword')
46        print("Creating 'pluto' database...")
47        time.sleep(20)
48        sql = mysql_db('MySecretPassword')
49        sql.create()
50        print("All done!")
51
52    def mongo_startup():
53        print("Creating MongoDB container ...")
54        mongo_container = mongo('some-mongo')
55        mongo_container.create()
56        print("Creating client...")
57        mng = mongo_db()
58        print("All done!")
```

container.py .gitignore db_ops.py scheduler.py

scheduler.py > ...

```
120   def timeloop():
121       print(f'----------- LOOP: ' + time.ctime() + ' ---------------')
122       mysql_write()
123       mongodb_write()
124       cassandra_write()
125       verify()
126       Timer(5, timeloop).start()
127
128   mysql_startup()
129   mongo_startup()
130   #redis_startup()
131   #cassandra_startup()
132   #timeloop()
```

container.py | .gitignore | db_ops.py | scheduler.py

scheduler.py > ...

```
120  def timeloop():
121      print(f'----------- LOOP: ' + time.ctime() + ' ---------------
122      mysql_write()
123      mongodb_write()
124      cassandra_write()
125      verify()
126      Timer(5, timeloop).start()
127
128  mysql_startup()
129  mongo_startup()
130  #redis_startup()
131  #cassandra_startup()
132  #timeloop()
```

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

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

Search

	NAME	STARTED	STATUS
	mongo 699c5215f105 (some-mongo)	1 minute ago	running
	mysql 4fd7292cc5a5 (some-mysql)	1 minute ago	running

3. Displaying the logic used to create both “Redis” and “Cassandra” containers and showing the code running to create them.

container.py | db_ops.py | scheduler.py

container.py > mysql

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

    def create(self):
        print("Pulling Cassandra image...")
        client.images.pull("cassandra")
        print("Running Cassandra container...")
        container = client.containers.run(
            'cassandra',
            name = self.cn_name,
            ports = {'9042/tcp': 9042},
            detach = True
        )
        print("Cassandra 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:
            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}")

```

container.py | db_ops.py | scheduler.py

db_ops.py > ...

```python
class cassandra_db:
    def __init__(self):
        self.cluster = Cluster(['localhost'],port=9042)
        self.session = self.cluster.connect()

    def create(self):
        self.session.execute("""CREATE KEYSPACE IF NOT EXISTS pluto WITH REPLICATION = {
                                'class': 'SimpleStrategy', 'replication_factor': 1 }""")
        self.session.set_keyspace('pluto')

        self.session.execute("""
            CREATE TABLE IF NOT EXISTS post (
                id text PRIMARY KEY,
                stamp text
            );
        """
        )
        self.session.shutdown()
        self.cluster.shutdown()

    def write(self):
        self.session.execute('USE pluto')
        id = str(uuid.uuid4())
        time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        self.session.execute("""INSERT INTO post (id, stamp) VALUES (%s, %s)""", (id, time))
        self.session.shutdown()
        self.cluster.shutdown()


    def read(self):
        self.session.execute('USE pluto')
        rows = self.session.execute('SELECT * FROM post')
        for row in rows:
            print(row)

    def delete(self):
        query = f"DELETE FROM pluto"
        self.session.execute(query)

```

container.py | db_ops.py | scheduler.py

container.py > mysql

```python

class redis:
    def __init__(self, cn_name):
        self.cn_name = cn_name

    def create(self):
        print("Pulling Redis image")
        client.images.pull("redis")
        print("Running Redis container...")
        container = client.containers.run(
            'redis',
            name = self.cn_name,
            ports = {'6379/tcp': 6379},
            detach = True
        )
        print("Cassandra 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:
            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}")

```

container.py | db_ops.py | scheduler.py

db_ops.py > redis_db > read

```python
class redis_db:
    def __init__(self):
        self.r = redis.Redis(host='localhost', port=6379, db=0)

    def write(self):
        id = str(uuid.uuid4())
        time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        values = {
            "id": id,
            "time": time
        }
        self.r.mset(values)

    def read(self):
        cursor = '0'
        while cursor != 0:
            cursor, keys = self.r.scan(cursor=cursor, count=10)
            for key in keys:
                value = self.r.get(key)
                print(f"{key.decode('utf-8')}: {value.decode('utf-8')}")

    def delete(self):
        keys = self.r.keys('*')
        if keys:
            self.r.delete(*keys)
            print("All keys deleted.")
        else:
            print("No keys found to delete.")
```

container.py | db_ops.py | scheduler.py

scheduler.py > ...

```python
def redis_startup():
    print("Creating Redis container...")
    redis_container = redis('some-redis')
    redis_container.create()
    print("Creating client...")
    rds = redis_db()
    print("All done!")

def cassandra_startup(retries=10, delay=20):
    print("Creating Cassandra container...")
    css_container = cassandra('some-cassandra')
    css_container.create()

    for attempt in range(retries):
        try:
            print(f'Attempting to create client... (Attempt {attempt + 1} of {retries})')
            css = cassandra_db()
            css.create()
            print('Cassandra setup completed successfully.')
            break
        except NoHostAvailable:
            print(f'Connection attempt {attempt + 1} failed. Retrying in {delay} seconds...')
            time.sleep(delay)
    else:
        print('Failed to connect to Cassandra after multiple attempts.')
```

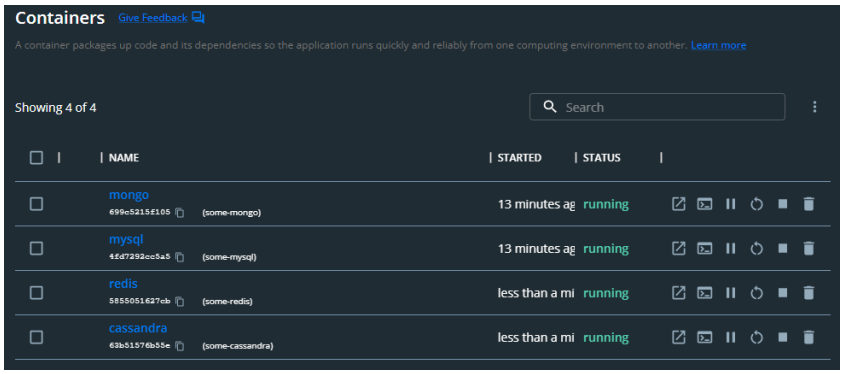
container.py | db_ops.py | scheduler.py

scheduler.py > ...

```
120    def timeloop():
121        print('---------- LOOP: ' + time.ctime() + ' ----------')
122        mysql_write()
123        mongodb_write()
124        cassandra_write()
125        verify()
126        Timer(5, timeloop).start()
127
128    #mysql_startup()
129    #mongo_startup()
130    redis_startup()
131    cassandra_startup()
132    #timeloop()
```

PROBLEMS 1 | OUTPUT | TERMINAL | PORTS | DEBUG CONSOLE

```
PS C:\MIT Docs\Module 13\Activity 13.3> python scheduler.py
Creating Redis container...
Pulling Redis image
Running Redis container...
Cassandra container is running
Creating client...
All done!
Creating Cassandra container...
Pulling Cassandra image...
Running Cassandra container...
Cassandra container is running
Attempting to create client... (Attempt 1 of 10)
Connection attempt 1 failed. Retrying in 20 seconds...
Attempting to create client... (Attempt 2 of 10)
Connection attempt 2 failed. Retrying in 20 seconds...
Attempting to create client... (Attempt 3 of 10)
Connection attempt 3 failed. Retrying in 20 seconds...
Attempting to create client... (Attempt 4 of 10)
Connection attempt 4 failed. Retrying in 20 seconds...
Attempting to create client... (Attempt 5 of 10)
Cassandra setup completed successfully.
PS C:\MIT Docs\Module 13\Activity 13.3>
```



4. Providing a screenshot of the “scheduler.py” running the timeloop

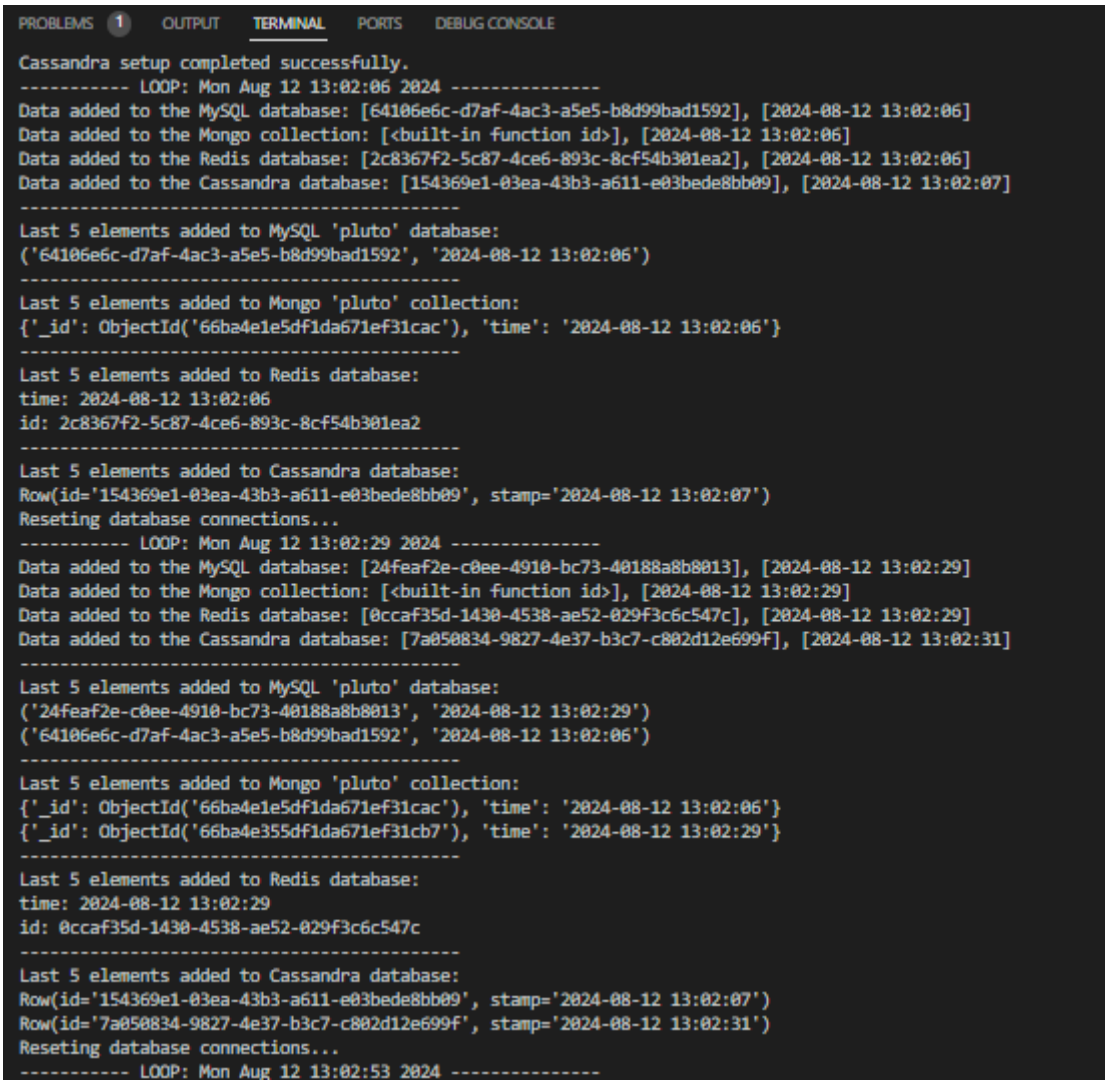
   I was having problems with the Timer class and had to change the way the loop works by simplifying the implementation, instead of using the “Timer” class I am using a “while True” condition as the use of this external library was provoking threading issues.

```
def timeloop():
    while True:
        print(f'----------- LOOP: ' + time.ctime() + ' ---------------')
        mysql_write()
        mongodb_write()
        redis_write()
        cassandra_write()
        verify()
        print("Reseting database connections...")
        time.sleep(20)

mysql_startup()
mongo_startup()
redis_startup()
cassandra_startup()
timeloop()
```

   I also added a 20 second timeout in order for all the database connections to be reset and prevent errors from happening in the case of having closing connections that haven’t ended their process.

   However, everything still works gracefully and the registries are still added periodically.



GitHub repo: https://github.com/MiguelBackAtItAgain/DockerContainers