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

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. 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.

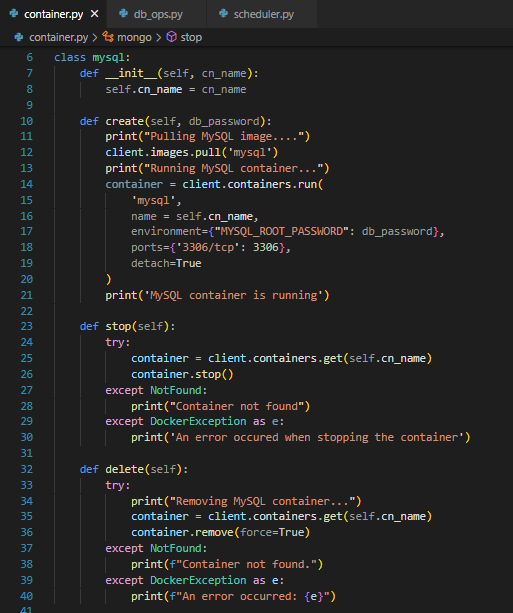
Now that this has been explained, I hereby add the screenshots of the “container.py” file:

Prerequisites:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Classes:



Texto

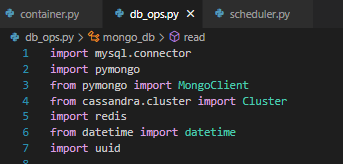
Descripción generada automáticamente

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:



Classes:

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

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.

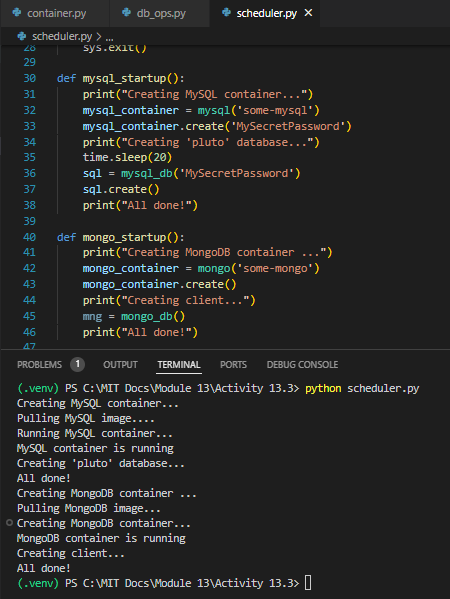


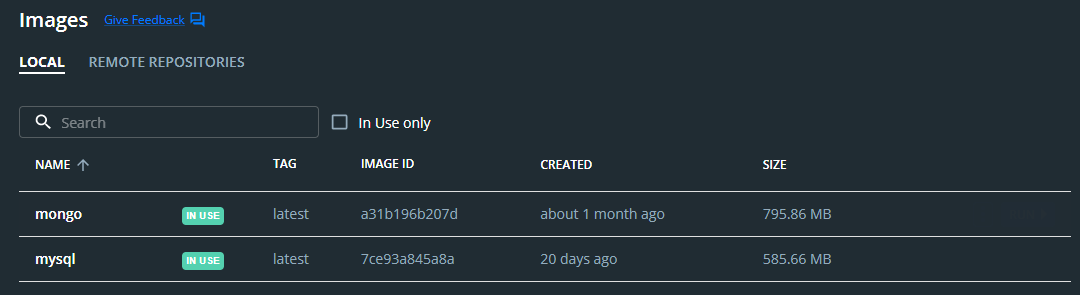
Texto

Descripción generada automáticamente

1. 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:



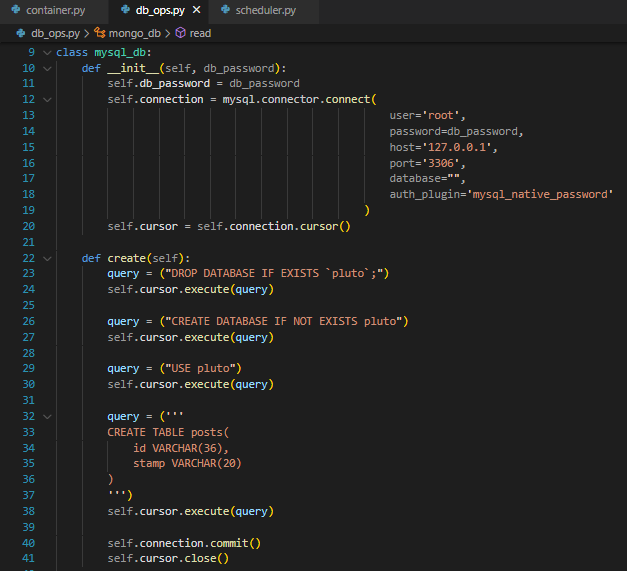


Captura de pantalla de un celular

Descripción generada automáticamente

1. 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.

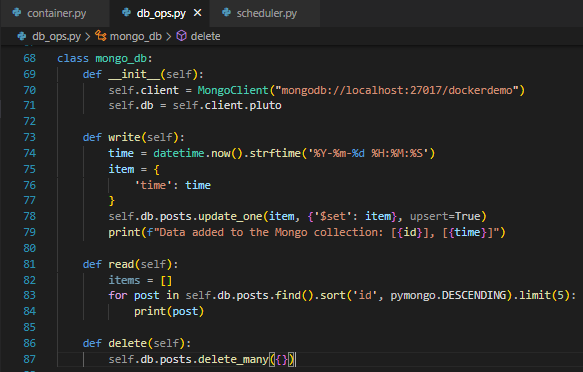


Texto

Descripción generada automáticamente

1. 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.



1. 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.

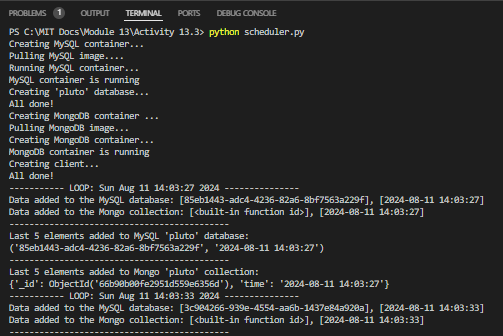
Texto

Descripción generada automáticamente

1. 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.

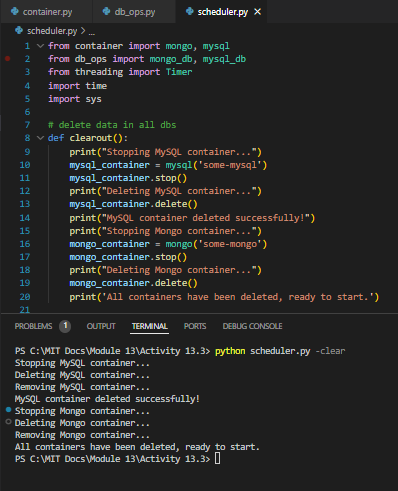
1. Running the “scheduler.py” and showing it working correctly.



Captura de pantalla de un celular

Descripción generada automáticamente

1. 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.



1. Displaying that the containers have been disposed of

Captura de pantalla de computadora

Descripción generada automáticamente