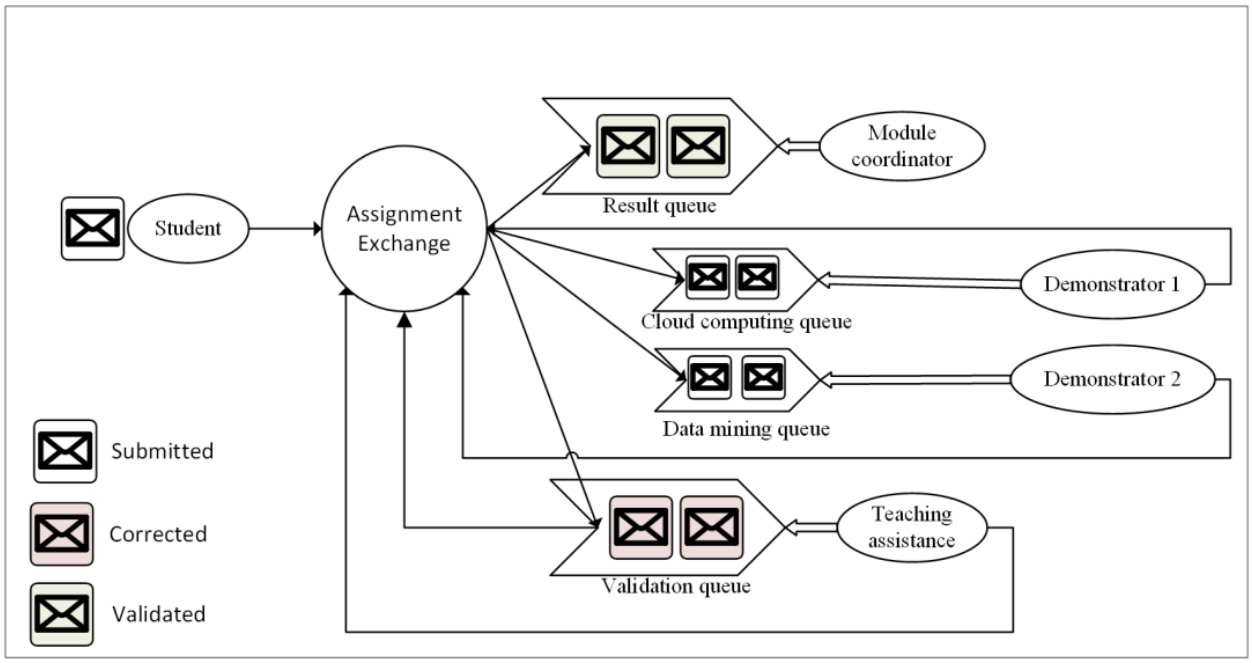
Assignment 2

Exercise One:

Exercise One tasks us with filling in the files `student.py`, `teaching_assistant.py`, `demonstrator.py` and `module_coordinator.py` which were provided with the assignment. After implementation, there should be a working scenario where a student sends two assignments for correction (a data mining assignment and a cloud computing assignment), and the flow of the assignment correction should be clearly stated in the terminal.

The flow of assignment correction should resemble the flow illustrated in the figure provided in the assignment brief below:



Code Changes:

Firstly, since there are two types of demonstrators, one for each module Cloud Computing and Data Mining, we need to split the `demonstrator` class into two subclasses, `demonstrator_cc` and `demonstrator_dm`. In order for this to function correctly we must declare new queues for each type of assignment, and bind them to the exchange `assignment_exchange`, as well as creating their corresponding routing key to decide how to route the messages to queues. This is achieved by replacing the `correction_queue` code in `module_coordinator.py` with the following:

```
channel.queue_declare(queue='cloud_queue')
channel.queue_bind(exchange='assignment_exchange',queue='cloud_queue', routing_key='cloud_computing.submitted')
channel.queue_declare(queue='data_mining_queue')
channel.queue_bind(exchange='assignment_exchange',queue='data_mining_queue', routing_key='data_mining.submitted')
```

We also don't want any messages being printed by any of the classes that should not be dealing with the respective state of flow that the message states. For example, a Module Coordinator should only be concerned with validated assignments and then publishing them, they shouldn't be concerned with assignments being sent for correction or being sent for validation etc. For this

reason, I moved the if statement that checks for the status of the assignment outside of the code being iterated through after inside the `callback` functions in the Module Coordinator, Teaching assistant and demonstrators classes. Now the Module Coordinators `callback` function checks if the assignments status is `validated`. Note that the Module Coordinator then changes the status of the assignment to `published`. The function now looks like this:

```
def callback(ch, method, properties, body):
  assignment = json.loads(body)
  if(assignment['status']=='validated'):
      print("Received "+str(assignment['StudentID'])+ "," + str(assignment['module']) + ","+str(assignment['status'])+", and submit it to B
      print("Publishing Assignment...")
      assignment['status']='published'
      print("Assignment Published")
```

Now we can split the `demonstrator` classes up into `demonstrator_cc` and `demonstrator_dm`. The only way they differ is in their `callback` function (inputting the correct queue, checking the correct status, etc) and the following final lines of code in the `main`. The demonstrators then change the status of the assignment to `corrected` once they are finished with it. The code for both classes is as follows after making the respective changes:

`demonstrator_cc`:

```
  import pika, sys, os,json


  def main():
  #local
  connection = pika.BlockingConnection(
  pika.ConnectionParameters(host=os.environ.get("RABBITMQ_HOSTNAME", "localhost"),
                credentials=pika.PlainCredentials(os.environ.get("RABBITMQ_USERNAME", "guest"),
                os.environ.get("RABBITMQ_PASSWORD", "guest"))))



  channel = connection.channel()


  def callback(ch, method, properties, body):

      assignment = json.loads(body)
      if (assignment['module']=='Cloud Computing' and assignment['status'] == 'submitted'):
          print("Received Student "+str(assignment['StudentID'])+"'s Assignment for "+str(assignment['module'])+" as "+str(assignment['stat
          print("Correcting Assignment...")
          assignment['status']='corrected'
          print("Assignment Corrected")
          routing_key = assignment['module'].replace(" ","_").lower()+'.corrected'
          print("Sending Assignment for to TA for Validation...")
          channel.basic_publish(exchange='assignment_exchange',routing_key=routing_key,body=json.dumps(assignment))
          print("Assignment Sent for Validation")

  channel.basic_consume(queue='cloud_queue', on_message_callback=callback, auto_ack=False)
  print(' [*] Cloud Computing Demonstrator is Waiting...')
  channel.start_consuming()

  if __name__ == '__main__':
  try:
      main()
  except KeyboardInterrupt:
      print('Interrupted')
      try:
          sys.exit(0)
      except SystemExit:
          os._exit(0)
```

`demonstrator_dm`:

```
import pika, sys, os,json


def main():
  #local
  connection = pika.BlockingConnection(
  pika.ConnectionParameters(host=os.environ.get("RABBITMQ_HOSTNAME", "localhost"),
                credentials=pika.PlainCredentials(os.environ.get("RABBITMQ_USERNAME", "guest"),
                os.environ.get("RABBITMQ_PASSWORD", "guest"))))



  channel = connection.channel()


  def callback(ch, method, properties, body):

      assignment = json.loads(body)
      if (assignment['module']=='Data Mining' and assignment['status'] == 'submitted'):
          print("Received Student "+str(assignment['StudentID'])+"'s Assignment for "+str(assignment['module'])+" as "+str(assignment['stat
          print("Correcting Assignment...")
          assignment['status']='corrected'
          print("Assignment Corrected")
          routing_key = assignment['module'].replace(" ","_").lower()+'.corrected'
          print("Sending Assignment for to TA for Validation...")
          channel.basic_publish(exchange='assignment_exchange',routing_key=routing_key,body=json.dumps(assignment))
          print("Assignment Sent for Validation")

  channel.basic_consume(queue='data_mining_queue', on_message_callback=callback, auto_ack=False)
  print(' [*] Data Mining Demonstrator is Waiting...')
  channel.start_consuming()

if __name__ == '__main__':
  try:
      main()
  except KeyboardInterrupt:
      print('Interrupted')
      try:
          sys.exit(0)
      except SystemExit:
          os._exit(0)
```

The corresponding changes regarding the guard on the if statement in `teaching_assistment.py` also needed to be made and are as follows. Note that the teaching assistant checks if the status of the assignment is `corrected` because thats all the TA is concerned with, and then changes the status of the assignment to `validated` once they are finished.

```
def callback(ch, method, properties, body):
  assignment = json.loads(body)
  if(assignment['status']=='corrected'):
      print("Received "+str(assignment['StudentID'])+","+str(assignment['status'])+","+str(assignment['module']))
      print("Validating Assignment...")
      assignment['status']='validated'
      print("Assignment Validated")
      channel.basic_publish(exchange='assignment_exchange',routing_key=str(assignment['module'])+'.validated',body=json.dumps(assignment))
      print("Assignment send to Module Coordinator")
```

Finally, for the `studnet.py` class, I defined the `sendAssignment` function to jumpstart the entire flow of assignment correction. The function takes the details of the assignment as parameters:

- `id` : Student ID
- `module` : The module from which the assignment belongs
- `answers` : The students answers to the assignment

The function then sets the assignments status to `submitted` in order for it to be received okay by the demonstrator, because both demonstrator classes check for this being the status of the assignment to make sure they are doing their tasks at the correct state in the flow of correction. It then creates its corresponding routing key by taking the name of the module in lower case and appending 'submitted' to it. Finally, the message is then published (sent) to the message queue system. The code for `sendAssignment` is as follows:

```
def sendAssignment(id, module, answers):

  assignment = json.dumps({
      'StudentID': id,
      'module': module,
      'answer': answers,
      'status':'submitted'
  })

  routing_key = module.replace(" ","_").lower()+'.submitted'
  channel.basic_publish(exchange='assignment_exchange',routing_key=routing_key, body=assignment)
  print(f"{module} Assignment sent for correction")
```

We will test this with with a scenario where a student is submitting two assignments, one for cloud computing and one for data mining:

```
sendAssignment(1, "Cloud Computing", "answers")
sendAssignment(1, "Data Mining", "answers")
```

Commands:

Now that the python code is correctly implemented, we can test it.

Firstly we need to pull the RabbitMQ image we need from the Docker Hub registry.

`docker pull rabbitmq:3.13-rc-management`

Then we need to run the container based on the RabbitMQ image with the specified version `3.13-rc-management` in detached mode (`-d`) while mapping the port 5672 to port 5672 in the container

`docker run -d -p 5672:5672 -p 15672:15672 rabbitmq:3.13-rc-management`

We can observe the status the activity on http://localhost:15672/#/

We then run python files in following order in separate terminals:

`python3 module_coorindator.py`

```
alexodonnell@dhcp-892be89e ex1 % ls
demonstrator_cc.py      demonstrator_dm.py      module_coordinator.py   student.py              teaching_assistant.py
alexodonnell@dhcp-892be89e ex1 % python3 module_coordinator.py
 [*] Module Coodinator is Waiting...
```

`python3 teaching_assitant.py`

```
alexodonnell@dhcp-892be89e ex1 % ls
demonstrator_cc.py      demonstrator_dm.py      module_coordinator.py   student.py              teaching_assistant.py
alexodonnell@dhcp-892be89e ex1 % python3 teaching_assistant.py
 [*] Teaching assistant is Waiting...
```

`python3 demonstrator_cc.py`

```
alexodonnell@dhcp-892be89e ex1 % ls
demonstrator_cc.py      demonstrator_dm.py      module_coordinator.py   student.py              teaching_assistant.py
alexodonnell@dhcp-892be89e ex1 % python3 demonstrator_cc.py
 [*] Cloud Computing Demonstrator is Waiting...
```

`python3 demonstrator_dm.py`

```
alexodonnell@dhcp-892be89e ex1 % ls
 demonstrator_cc.py      demonstrator_dm.py      module_coordinator.py   student.py              teaching_assistant.py
alexodonnell@dhcp-892be89e ex1 % cd demonstrator_dm.py
 cd: not a directory: demonstrator_dm.py
alexodonnell@dhcp-892be89e ex1 % python3 demonstrator_dm.py
  [*] Data Mining Demonstrator is Waiting...
```

`python3 student.py`

```
alexodonnell@dhcp-892be89e ex1 % python3 student.py
Cloud Computing Assignment sent for correction
Data Mining Assignment sent for correction
Assignment(s) sent for correction
alexodonnell@dhcp-892be89e ex1 %
```

We can see here that once `student.py` is run, both assignments are sent for correction

Output:

We can see immediately that both demonstrators received the correct assignment that they should be correcting, the assignments were corrected and sent to be validated by the TA

```
alexodonnell@dhcp-892be89e ex1 % python3 demonstrator_dm.py
 [*] Data Mining Demonstrator is Waiting...
Received Student 1's Assignment for Data Mining as submitted
Correcting Assignment...
Assignment Corrected
Sending Assignment for to TA for Validation...
Assignment Sent for Validation
```

```
alexodonnell@dhcp-892be89e ex1 % python3 demonstrator_cc.py
  [*] Cloud Computing Demonstrator is Waiting...
 Received Student 1's Assignment for Cloud Computing as submitted
 Correcting Assignment...
 Assignment Corrected
 Sending Assignment for to TA for Validation...
 Assignment Sent for Validation
```

We can then see that the TA received both corrected assignments, validated both of them, and sent both of them to the Module Coordinator

```
(alexodonnell@dhcp-892be89e ex1 % python3 teaching_assistant.py
 [*] Teaching assistant is Waiting...
Received 1,corrected,Cloud Computing
Validating Assignment...
Assignment Validated
Assignment send to Module Coordinator
Received 1,corrected,Data Mining
Validating Assignment...
Assignment Validated
Assignment send to Module Coordinator
```

Finally, we can see that the Module Coordinator received both validated corrections, and both the Cloud Computing assignment and Data Mining assignments were then finally published.

```
(alexodonnell@dhcp-892be89e ex1 % python3 module_coordinator.py
 [*] Module Coodinator is Waiting...
Received 1,Cloud Computing,validated, and submit it to Brightspace
Publishing Assignment...
Assignment Published
Received 1,Data Mining,validated, and submit it to Brightspace
Publishing Assignment...
Assignment Published
```

Its clear from this output that the flow of assignment correction stated in the question as been correctly implemented.

Exercise 2:

The task here is to dockerize our solution from Exercise 1 by using a `docker-compose` file to run 6 individual containers. One for the `RabbitMQ` container, and one for each of our classes:

- `student.py`
- `demonstrator_cc.py`
- `demonstrator_dm.py`
- `teaching_assitant.py`
- `module_coordinator.py`

This involves creating a `Dockerfile` for each python class, and also a `docker-compose.yaml` to define and manage the multi container Docker application. All of the Dockerfiles will follow the same format and implementation bar the specifications for the corresponding file. The Dockerfile implementation is as follows:

```
# Dockerfile for the Module Coordinator role
FROM python:3.8
ENV PYTHONUNBUFFERED=1
COPY module_coordinator.py /
RUN pip install pika
CMD ["sh", "-c", "sleep 60 && python3 module_coordinator.py"]
```

Each Dockerfile follows this exact format, the only changes are the python file specification. The breakdown of this code is as follows:

`FROM python:3.8`

This line specifies the base Docker image to use as the starting point for building this image. In this case, it's using the official Python 3.8 image from Docker Hub

`ENV PYTHONUNBUFFERED=1`

This line sets the `PYTHONUNBUFFERED` environment variable to a non-empty value. This is done to ensure that Python output is sent directly to the terminal without buffering, making it easier to troubleshoot issues

`COPY module_coordinator.py /`

This line copies the `module_coordinator.py` file from the build context (the directory where the Dockerfile is located) to the root directory (`/`) inside the Docker image.

`RUN pip install pika`

This line installs the `pika` Python library using `pip`. `pika` is a Python library for interacting with RabbitMQ, the message broker

`CMD ["sh", "-c", "sleep 60 && python3 module_coordinator.py"]`

This line specifies the default command to run when a container based on this image is started.

It uses `sh` to execute a shell command, which sleeps for 60 seconds (`sleep 60`) before running the Python script (`python3 module_coordinator.py`).

This can be useful in scenarios where dependencies need time to initialise

Next we need to create and implement our `docker-compose.yaml` to manage these containers. The implementation is as follows:

```
version: '3'

networks:
  rabbitmq_go_net:
    driver: bridge

services:
  student:
    build:
      context: ./Student
      dockerfile: Dockerfile
    restart: unless-stopped
    depends_on:
      - rabbitmq
      - demonstrator_cc
      - demonstrator_dm
    networks:
      - rabbitmq_go_net

  demonstrator_dm:
    build:
      context: ./DemonstratorDM
      dockerfile: Dockerfile
    restart: unless-stopped
    depends_on:
      - rabbitmq
      - teaching_assistant
    networks:
      - rabbitmq_go_net

  demonstrator_cc:
    build:
      context: ./DemonstratorCC
      dockerfile: Dockerfile
    restart: unless-stopped
    depends_on:
      - rabbitmq
      - teaching_assistant
    networks:
      - rabbitmq_go_net


  teaching_assistant:
    build:
      context: ./TeachingAssistant
```

```
      dockerfile: Dockerfile
    restart: unless-stopped
    depends_on:
      - rabbitmq
      - module_coordinator
    networks:
      - rabbitmq_go_net

  module_coordinator:
    build:
      context: ./ModuleCoordinator
      dockerfile: Dockerfile
    restart: unless-stopped
    depends_on:
      - rabbitmq
    networks:
      - rabbitmq_go_net

  rabbitmq:
    image: "rabbitmq:3.13-rc-management"
    container_name: rabbitmq
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest
    networks:
      - rabbitmq_go_net
    healthcheck:
      test: rabbitmq-diagnostics check_port_connectivity
      interval: 5s
      timeout: 60s
      retries: 30
```

The breakdown of this code is as follows:

```
rabbitmq:
    image: "rabbitmq:3.13-rc-management"
    container_name: rabbitmq
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest
    networks:
      - rabbitmq_go_net
    healthcheck:
      test: rabbitmq-diagnostics check_port_connectivity
      interval: 5s
      timeout: 60s
      retries: 30
```

This service uses the official RabbitMQ Docker image with version 3.13-rc-management.

It maps the ports 5672 and 15672 from the container to the same ports on the host.

Environment variables are set for the RabbitMQ default user and password.

The service is connected to the `rabbitmq_go_net` network.

A health check is defined to test the RabbitMQ port connectivity

```
networks:
  rabbitmq_go_net:
    driver: bridge
```

This section defines a network named `rabbitmq_go_net` with the bridge driver. The bridge driver is the default network driver for Docker containers.

We then have service for each class, which the implementation is very similar for each class. The breakdown of the Student service for example is as follows:

```
student:
    build:
      context: ./Student
      dockerfile: Dockerfile
    restart: unless-stopped
    depends_on:
      - rabbitmq
      - demonstrator_cc
      - demonstrator_dm
    networks:
      - rabbitmq_go_net
```

This service is named `student`

It builds an image using the Dockerfile in the `./Student` directory

The service is set to restart unless explicitly stopped

It depends on the services `rabbitmq`, `demonstrator_cc`, and `demonstrator_dm`

It is connected to the `rabbitmq_go_net` network

Each of the other services have nearly identical implementation except for the services they depend on, for example the `demonstrator_cc` and `demonstrator_dm` both depend on the `teaching_assitant` instead of the demonstrators as the `student` does. The dependencies follow the same format of the assignment correction flow.

There are also some minor code changes that need to be made to each of the python classes so that they are compatible with the dockerisation we are trying to implement. Firstly, each class needs to have its own folder or directory with its corresponding `Dockerfile` in it with its python script in this case, due to how the Dockerfiles are implemented. One of the changes necessary is replace the code for the `local` configuration with the `docker configuration` in each python class.

Original `local` configuration:

```
#local
connection = pika.BlockingConnection(
pika.ConnectionParameters(host=os.environ.get("RABBITMQ_HOSTNAME", "localhost"),
                credentials=pika.PlainCredentials(os.environ.get("RABBITMQ_USERNAME", "guest"),
                os.environ.get("RABBITMQ_PASSWORD", "guest"))))
```

Replacement `docker` configuration:

```
#docker
credentials = pika.PlainCredentials("guest", "guest")
connection = pika.BlockingConnection(pika.ConnectionParameters("rabbitmq", 5672, "/", credentials))
```

This code simply defines the credentials for a default user for `RabbitMQ` with the username and password set to “guest”, and then establishes a blocking connection to `RabbitMQ` using the “rabbitmq” server host name, the port 5672, the default virtual host “/” and the credentials we just defined.

Another change that is necessary for this configuration to work is setting `durable=True` inside our `module_coordinator.py` class in our queue declarations**.** In the context of RabbitMQ exchanges and queues ensures that they persist and survive a broker restart. This is particularly important for maintaining the state of queues and exchanges in scenarios where the message broker may need to restart, ensuring that important data is not lost during such events.

```
channel.queue_declare(queue='cloud_queue', durable=True)
channel.queue_declare(queue='data_mining_queue', durable=True)
channel.queue_declare(queue='validation_queue', durable=True)
channel.queue_declare(queue='result_queue', durable=True)
```

Finally, with these changes implemented, we are ready to test the dockerised version of our project. The scenario we will be testing it with is a student who sends five assignments for correction, two data mining assignments and three cloud assignments:

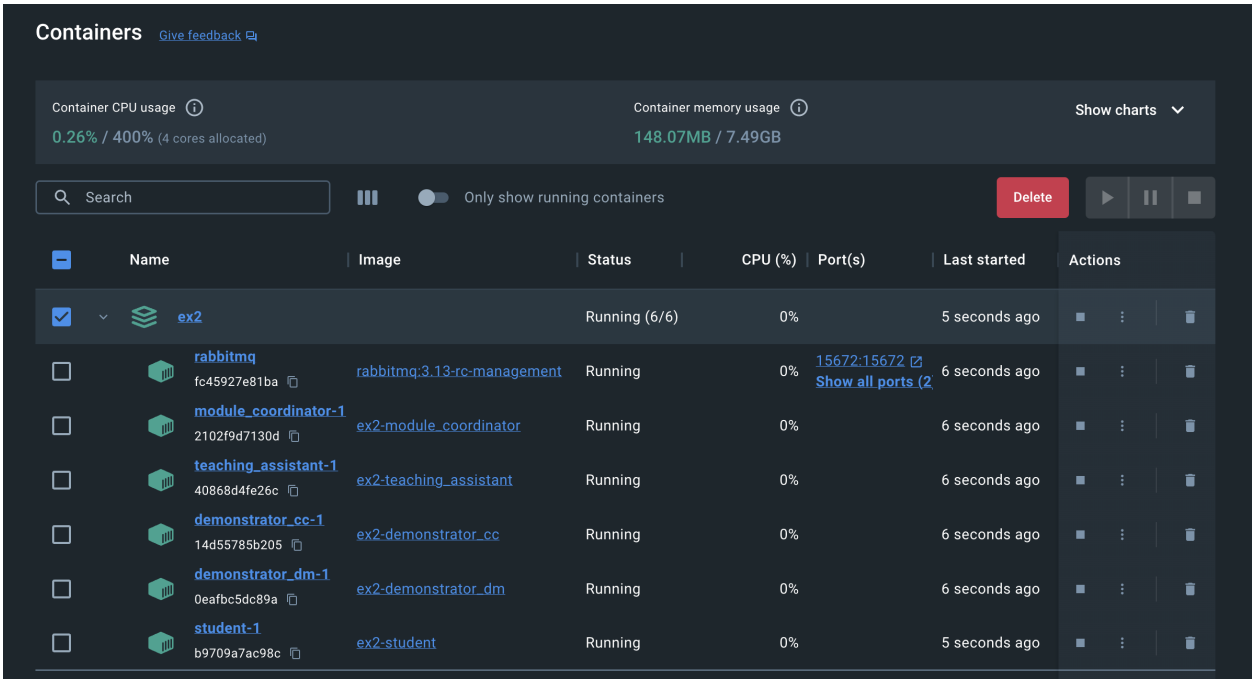
```
sendAssignment(1, "Data Mining", "answers")
sendAssignment(1, "Data Mining", "answers")
sendAssignment(1, "Cloud Computing", "answers")
sendAssignment(1, "Cloud Computing", "answers")
sendAssignment(1, "Cloud Computing", "answers")
```

After navigating to the exercise 2 folder, the only command needed to test our code is:

`docker-compose up`

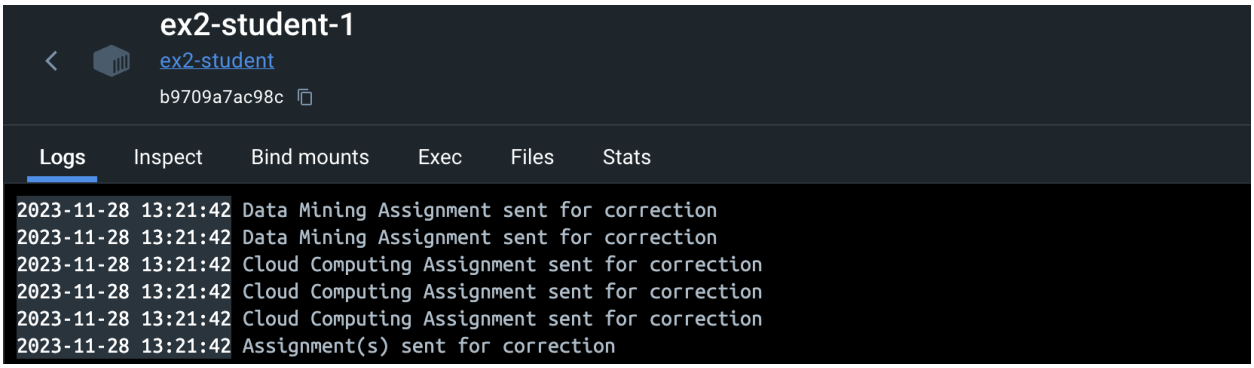
Output:

After running this command, we can see all 6 of our containers are running as planned.

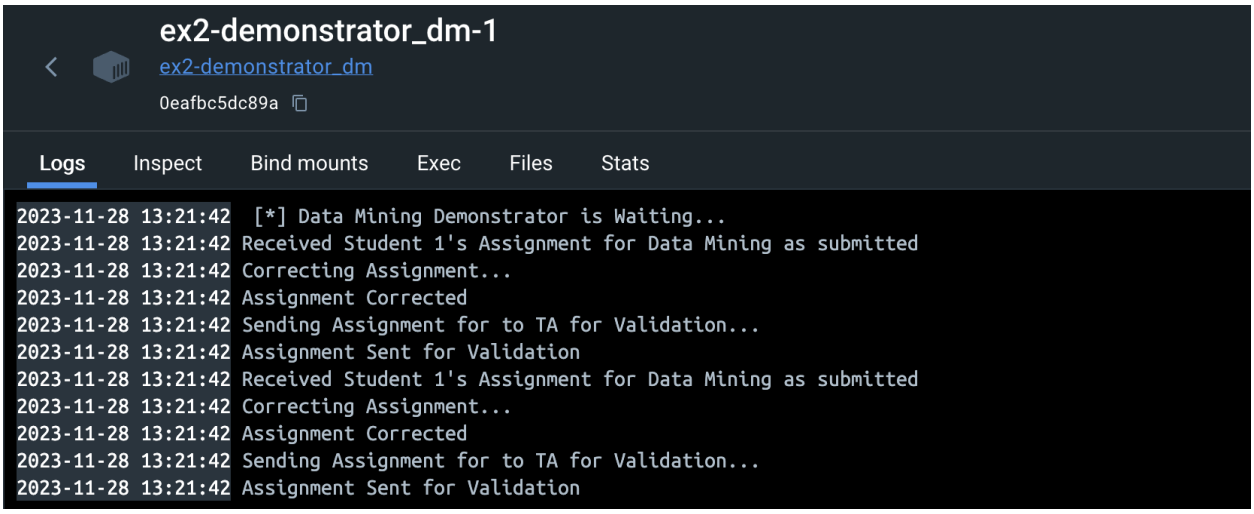


We can see that within each container, we got the expected output representing the desired flow of assignment correction to confirm that we have correctly dockerised this project:

The Student sends 5 assignments for correction, 2 data mining and 3 cloud computing



The Data Mining Demonstrator receives 2 assignments, corrects them and sends them to the Teaching Assistant for validation



The Cloud Computing Demonstrator receives 3 assignments, corrects them and sends them to the Teaching Assistant for validation

ex2-demonstrator_cc-1

ex2-demonstrator_cc

14d55785b205

Logs | Inspect | Bind mounts | Exec | Files | Stats

```
2023-11-28 13:21:42  [*] Cloud Computing Demonstrator is Waiting...
2023-11-28 13:21:42 Received Student 1's Assignment for Cloud Computing as submitted
2023-11-28 13:21:42 Correcting Assignment...
2023-11-28 13:21:42 Assignment Corrected
2023-11-28 13:21:42 Sending Assignment for to TA for Validation...
2023-11-28 13:21:42 Assignment Sent for Validation
2023-11-28 13:21:42 Received Student 1's Assignment for Cloud Computing as submitted
2023-11-28 13:21:42 Correcting Assignment...
2023-11-28 13:21:42 Assignment Corrected
2023-11-28 13:21:42 Sending Assignment for to TA for Validation...
2023-11-28 13:21:42 Assignment Sent for Validation
2023-11-28 13:21:42 Received Student 1's Assignment for Cloud Computing as submitted
2023-11-28 13:21:42 Correcting Assignment...
2023-11-28 13:21:42 Assignment Corrected
2023-11-28 13:21:42 Sending Assignment for to TA for Validation...
2023-11-28 13:21:42 Assignment Sent for Validation
```

The Teaching Assistant receives 5 corrected assignments, validates all of them and sends them to the Module Coordinator

ex2-teaching_assistant-1

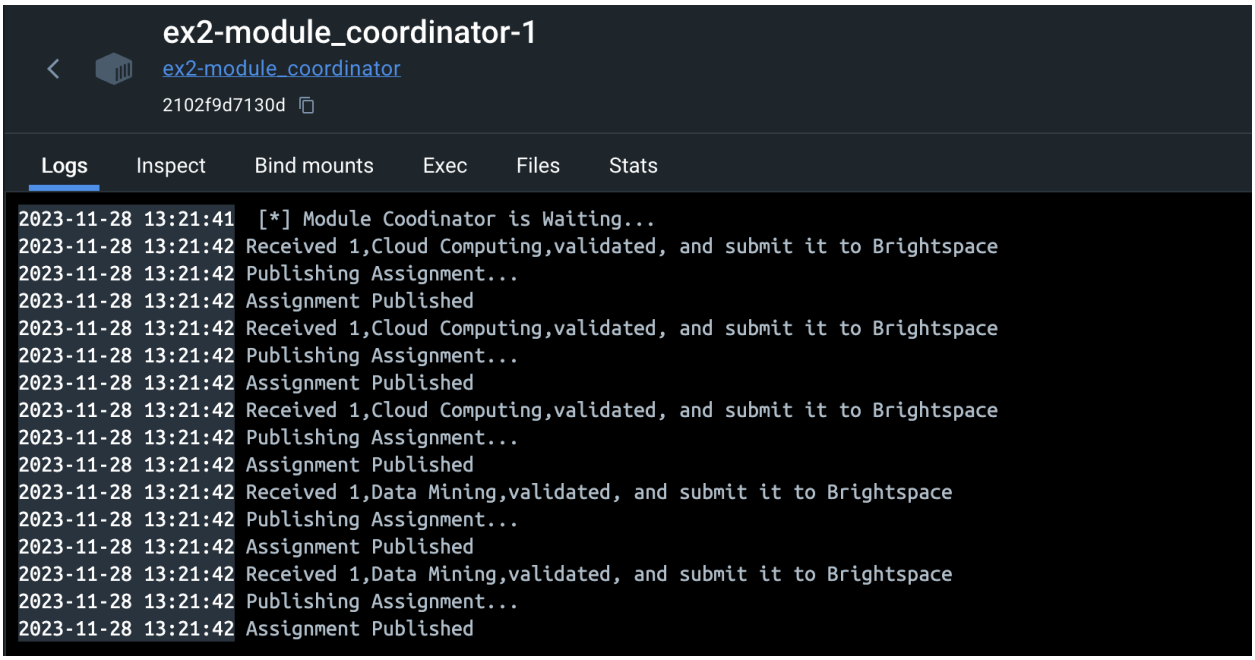
ex2-teaching_assistant

40868d4fe26c

Logs | Inspect | Bind mounts | Exec | Files | Stats

```
2023-11-28 13:21:41  [*] Teaching assistant is Waiting...
2023-11-28 13:21:42 Received 1,corrected,Cloud Computing
2023-11-28 13:21:42 Validating Assignment...
2023-11-28 13:21:42 Assignment Validated
2023-11-28 13:21:42 Assignment send to Module Coordinator
2023-11-28 13:21:42 Received 1,corrected,Cloud Computing
2023-11-28 13:21:42 Validating Assignment...
2023-11-28 13:21:42 Assignment Validated
2023-11-28 13:21:42 Assignment send to Module Coordinator
2023-11-28 13:21:42 Received 1,corrected,Cloud Computing
2023-11-28 13:21:42 Validating Assignment...
2023-11-28 13:21:42 Assignment Validated
2023-11-28 13:21:42 Assignment send to Module Coordinator
2023-11-28 13:21:42 Received 1,corrected,Data Mining
2023-11-28 13:21:42 Validating Assignment...
2023-11-28 13:21:42 Assignment Validated
2023-11-28 13:21:42 Assignment send to Module Coordinator
2023-11-28 13:21:42 Received 1,corrected,Data Mining
2023-11-28 13:21:42 Validating Assignment...
2023-11-28 13:21:42 Assignment Validated
2023-11-28 13:21:42 Assignment send to Module Coordinator
```

And finally, the Module Coordinator receives 5 validated assignment, and publishes all 5 of them.


ex2-module_coordinator-1
ex2-module_coordinator
2102f9d7130d
Logs
Inspect
Bind mounts
Exec
Files
Stats
2023-11-28 13:21:41 [*] Module Coodinator is Waiting...
2023-11-28 13:21:42 Received 1,Cloud Computing,validated, and submit it to Brightspace
2023-11-28 13:21:42 Publishing Assignment...
2023-11-28 13:21:42 Assignment Published
2023-11-28 13:21:42 Received 1,Cloud Computing,validated, and submit it to Brightspace
2023-11-28 13:21:42 Publishing Assignment...
2023-11-28 13:21:42 Assignment Published
2023-11-28 13:21:42 Received 1,Cloud Computing,validated, and submit it to Brightspace
2023-11-28 13:21:42 Publishing Assignment...
2023-11-28 13:21:42 Assignment Published
2023-11-28 13:21:42 Received 1,Data Mining,validated, and submit it to Brightspace
2023-11-28 13:21:42 Publishing Assignment...
2023-11-28 13:21:42 Assignment Published
2023-11-28 13:21:42 Received 1,Data Mining,validated, and submit it to Brightspace
2023-11-28 13:21:42 Publishing Assignment...
2023-11-28 13:21:42 Assignment Published