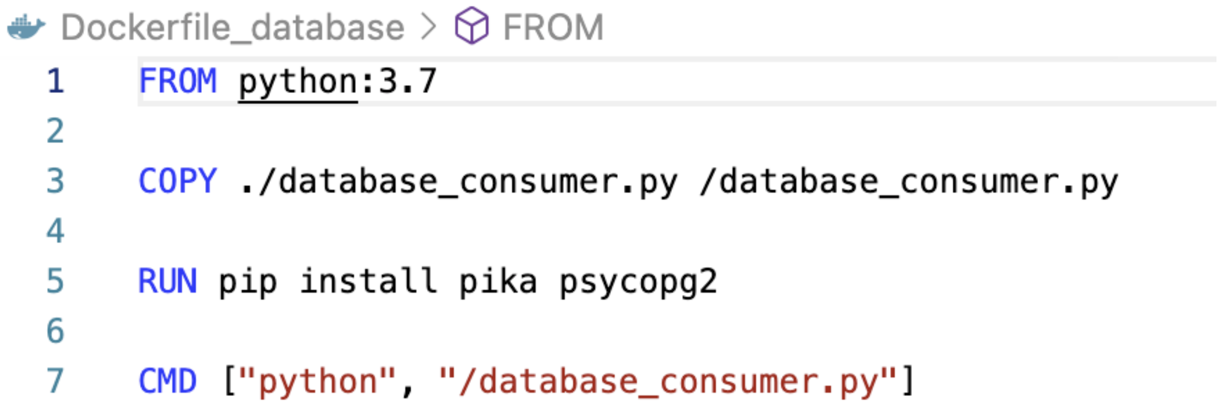
General Evaluation P2

This section contains a preliminary evaluation to ensure the students have implemented all the components specified in the documentation before being asked for variations.

1. There must be 2 POST Endpoints in the Producer Server. The endpoint for registering a new consumer should end with new_ride_matching_consumer and the endpoint for requesting a new ride should end with new_ride
   To look for this, in the producer file, there should be 2 endpoints configured. The students can be asked to show the endpoints, and they should be of type POST only.
   If the students are using Flask, then in the producer file, there should be 2 routes configured as POST similar to this below lines
   @app.route('/new_ride', methods=['POST'])
   @app.route('/new_ride_matching_consumer', methods=['POST'])

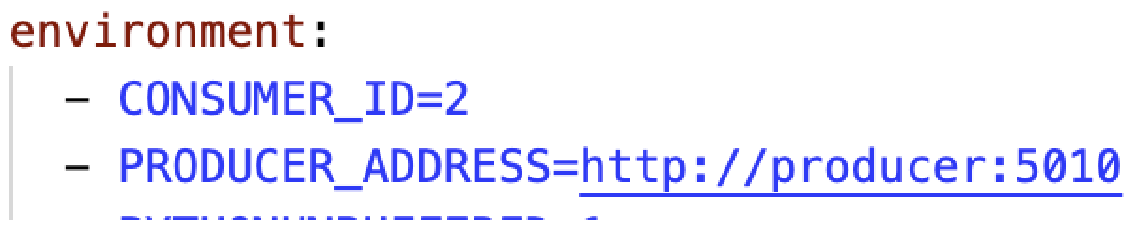
2. There must be 3 Dockerfiles, one for the producer, one for database consumer, and one for the ride-matching consumer. Each Dockerfile should approximately have the same contents - copying the files, installing dependencies, and running the script.
   If the students are using Python, it should look similar to the below screenshot



```
FROM python:3.7

COPY ./database_consumer.py /database_consumer.py

RUN pip install pika psycopg2

CMD ["python", "/database_consumer.py"]
```

   If the students decide to use some other language, the exact steps in the Dockerfile might vary but should be achieving the same goal.

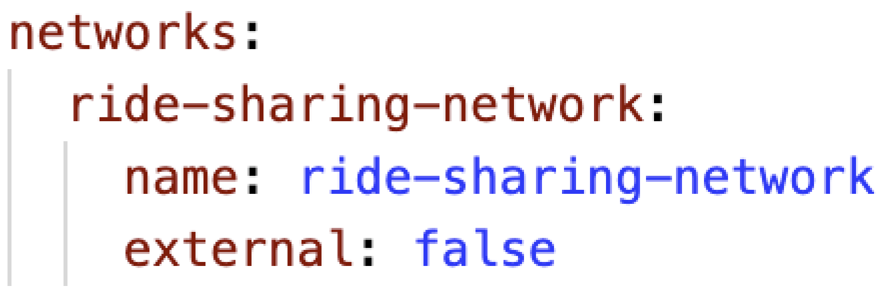
3. For the ride-matching consumer, the Consumer ID and the Producer Server IP Address must not be hard coded, but passed in through environment variables in the docker-compose file. The students can call the variable anything, but it must be used in the code.



```
environment:
  - CONSUMER_ID=2
  - PRODUCER_ADDRESS=http://producer:5010
```

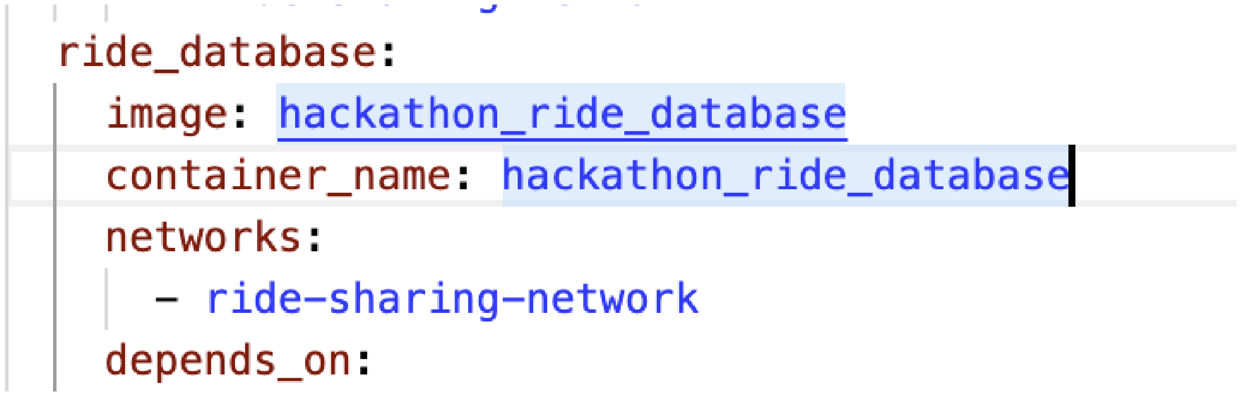
In the docker-compose file for the ride matching consumers, there should be these two variables.
The Variable Name and value might change, but they should be passed in the docker-compose file.

4. There must be a custom docker network in the docker-compose file which all the containers should use. This will ensure that all the containers are on the same network, and can easily be accessed.



```
networks:
  ride-sharing-network:
    name: ride-sharing-network
    external: false
```

A network similar to this must be configured. The name can vary, but this network must be used in all the containers.

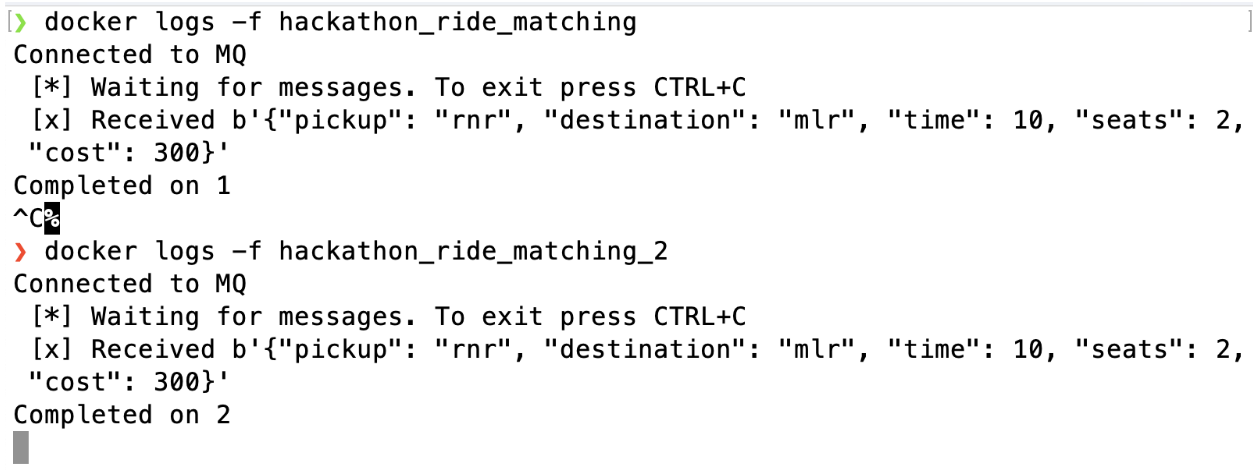


```
  ride_database:
    image: hackathon_ride_database
    container_name: hackathon_ride_database
    networks:
      - ride-sharing-network
    depends_on:
```

It should be passed under networks for all the containers as in the above screenshot.

5. When a new ride request is sent (students must send a request using an API tester tool like Postman or CLI like cURL) the request must be distributed to the ride-matching consumers in round-robin fashion. So if Req 1 is sent to Consumer 1, Req 2 must be sent to consumer 2, and so on. The output should be printed on the terminal.
   This can be viewed using `docker logs -f <container name>`

The container logs should show that the requests are being distributed to all the ride matching containers in round robin fashion.

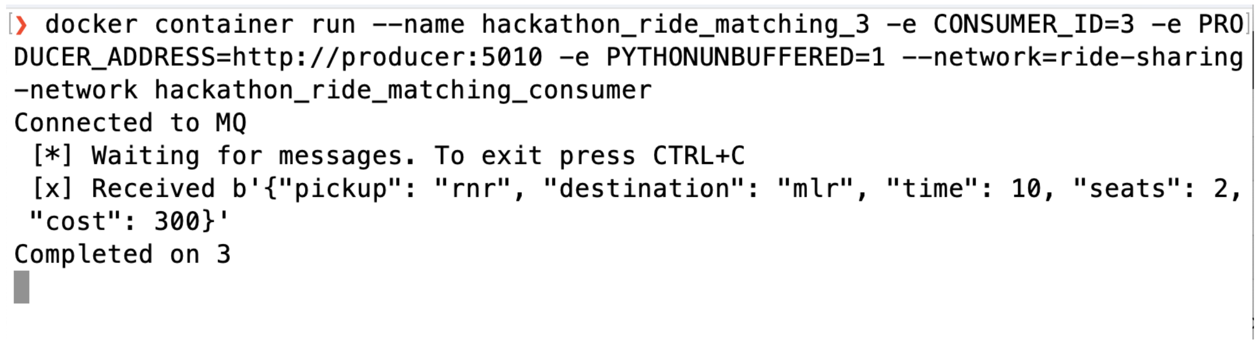


```
❯ docker logs -f hackathon_ride_matching
Connected to MQ
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'{"pickup": "rnr", "destination": "mlr", "time": 10, "seats": 2,
 "cost": 300}'
Completed on 1
^C%
❯ docker logs -f hackathon_ride_matching_2
Connected to MQ
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'{"pickup": "rnr", "destination": "mlr", "time": 10, "seats": 2,
 "cost": 300}'
Completed on 2
```

6. Ask a student to add a third ride-matching consumer to the setup. This will require them to manually spin up the container, and connect it to the same service as the rest. Ask them to run this third container from the command line, by passing env variables as command line arguments, and not by editing the docker-compose file. The command to run the container externally but connect to the other services is -

```
docker container run --name hackathon_ride_matching_3 -e
CONSUMER_ID=3 -e PRODUCER_ADDRESS=http://producer:5010 -e
PYTHONUNBUFFERED=1 --network=ride-sharing-network <Image Name>
```

It might take some time for it to start and connect, but once it's done, when requests are sent to the producer, this consumer must also receive them. 2-3 requests might need to be sent before this receives it since it's distributed to all the consumers.



```
❯ docker container run --name hackathon_ride_matching_3 -e CONSUMER_ID=3 -e PRO
DUCER_ADDRESS=http://producer:5010 -e PYTHONUNBUFFERED=1 --network=ride-sharing
-network hackathon_ride_matching_consumer
Connected to MQ
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'{"pickup": "rnr", "destination": "mlr", "time": 10, "seats": 2,
 "cost": 300}'
Completed on 3
```

Variations

1. **Change the Producer Server Port**

   The students should change the producer server port number. Due to this change, they will be expected to do two things for the app to run successfully.

   a. Rebuild the docker images so the updated files are used in the containers
   b. Change the Server IP environment variable in the docker-compose file so that the consumer uses the updated port number

   These steps can be provided as hints in case the students are not sure what to do

2. **Change the queue names**

   Currently the expected names for the queues used in the programs are “database” and “ride-match” for database microservice and ride matching microservice. These queue names are used in 3 places
   a. In the producer to decide which queue to send what request
   b. In the database microservice to tell it which queue to listen to
   c. In the ride-matching microservice to tell it which queue to listen to
   The students can be asked to change the queue names to something else, like “database” to “insertion” and “ride_match” to “matching-service”. The students will have to know where all to change the queue name, and then rebuild the docker images and re-deploy to get it running with the new names

3. **Change Environment Variable Names**
   The students will have to pass the Consumer ID and Producer URL through the docker compose file. Ask the students to change the variable names to some other name, and re-run the microservice. The students will have to do the following
   a. Change the variable names in the docker-compose file from their current name to any new name
   b. Change the Ride Matching Consumer code to read the new names
   c. Rebuild the images and re-run the compose file to bring up all the services
   d. Send a New Ride request and show that the service still runs and the new variables are being used.

4. **Change the docker network**

   The students will be expected to change the name of the docker network in the docker-compose files. They will then have to change the same name for all the other containers being spawned. They should then be asked to re-spawn a third ride-sharing consumer container externally like in Step 6 of the general evaluation, and this container should also take the new network name.

5. **Change the Image Names and re-compile**
   The students could be asked to rebuild the images, but with new names and update the docker-compose file to reflect the new change. To ensure the images they use are the new ones, they can be asked to delete existing ones by using `docker image rm <existing image name>`
   Since we have 3 images, we can ask them to name them image1, image2 and image3 - Any service can be mapped to any image name, the students will just have to keep track of that.
   So instead of `docker image build  -t hackathon_ride_database -f Dockerfile_database .` they will now have to use the new name for the image `docker image build  -t image1 -f Dockerfile_database .`
   Once that is done, they need to update the image names in the docker-compose file and re-run the service and show everything is working