EXERCISE 1: Correction System Using RabbitMQ

Task 1: Using Pika's syntax, declare and bind the necessary queues/exchanges and their connections as shown in Figure 1.

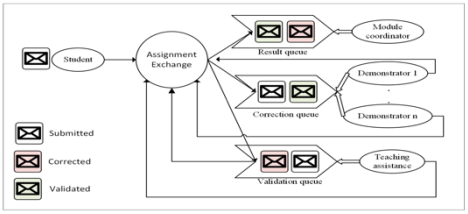
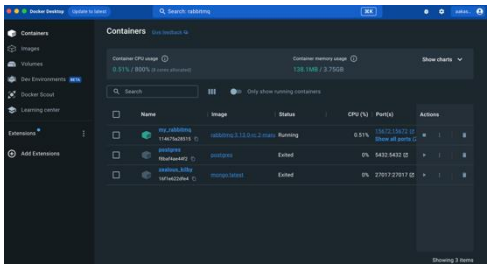


Figure 1: Practical correction through assignment broadcasting

Install RabbitMQ image in docker

Here we created docker container with RabbitMQ image for the exercises.



A. Declaration of Exchange:

```python
import pika
import json

# Connect to RabbitMQ Software through this command
connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()

# Task 1
# Using Pika's syntax, declare the necessary queues/exchanges and their connections as shown in Figure 1

channel.exchange_declare(exchange='assignment_exchange', exchange_type='direct')

channel.queue_declare(queue='correction_queue')
channel.queue_declare(queue='validation_queue')
channel.queue_declare(queue='results_queue')

channel.queue_bind(exchange='assignment_exchange', queue='correction_queue', routing_key='correction')
channel.queue_bind(exchange='assignment_exchange', queue='validation_queue', routing_key='validation')
channel.queue_bind(exchange='assignment_exchange', queue='results_queue', routing_key='results')

```

channel.exchange_declare(exchange='assignment_exchange', exchange_type='direct') creates a 'direct' exchange with the name 'assignment_exchange'.

B. Queue Declarations:

Using the following methods, three queues are established for results validation, correction.

The expressions channel.queue_declare(queue='correction_queue'), channel.queue_declare (queue='validation_queue'), and channel.queue_declare(queue='results_queue') represent t he queue operators.

C. Binding Queues:

Using channel.queue_bind(exchange='assignment_exchange', queue='correction_queue', routing_key='correction'), each queue is bound to the exchange with a unique routing key ('correction', 'validation','results').

Task 2: Using Pika's syntax, define where each actor (demonstrator, teaching assistant, module coordinator) should listen/consume.

```python
# Task 2
# Using Pika's syntax, declare where a demonstrator should listen/subscribe

def demonstrator_callback(ch, method, properties, body):
    assignment = json.loads(body.decode('utf-8'))
    print(f"Received message for correction: {assignment}")

    assignment['status'] = 'corrected'

    channel.basic_publish(
        exchange='assignment_exchange',
        routing_key='correction',
        body=json.dumps(assignment)
    )
    print("Assignment corrected")

channel.basic_consume(queue='correction_queue', on_message_callback=demonstrator_callback, auto_ack=True)
```

```python
# Task 3
# Using Pika's syntax, declare where a TA should listen/subscribe

def ta_callback(ch, method, properties, body):
    assignment = json.loads(body.decode('utf-8'))
    print(f"Received message for validation: {assignment}")

    if assignment['status'] == 'corrected':
        assignment['status'] = 'validated'

        channel.basic_publish(
            exchange='assignment_exchange',
            routing_key='validation',
            body=json.dumps(assignment)
        )
        print("Assignment validated")
    else:
        print("Ignoring the assignment as it's not in the corrected state.")

channel.basic_consume(queue='validation_queue', on_message_callback=ta_callback, auto_ack=True)
```

```python
# Task 4
# Using Pika's syntax, declare where the module coordinator should listen/subscribe

def coordinator_callback(ch, method, properties, body):
    assignment = json.loads(body.decode('utf-8'))
    print(f"Received message for confirmation: {assignment}")

    if assignment['status'] == 'validated':
        assignment['status'] = 'confirmed'

        print(f"Assignment confirmed: {assignment}")

channel.basic_consume(queue='results_queue', on_message_callback=coordinator_callback, auto_ack=True)

print('Waiting for messages')
channel.start_consuming()
```

Demonstrator Subscriber: Create a callback method (demonstrator_callback) to handle messages that the demonstrator gets from the "correction_queue."

Following message processing and correction logic, the function communicates the updated assignment back to the exchange using the routing key "correction."

Subscribes the demonstrator via channel.basic_consume to the 'correction_queue'.

TA Subscriber: Names a callback function (ta_callback) to handle messages from the 'validation_queue' that the TA receives.

Uses the routing key "validation" to publish the validated assignment back to the exchange after verifying that it is in the "corrected" state.

TA is subscribed to the 'validation_queue' through channel.basic_consume.

Module Coordinator Subscriber: Creates a callback function (coordinator_callback) to handle messages from the 'results_queue' that are received by the module coordinator.

Confirms whether the assignment is in the "validated" state, executes logic for confirmation, and generates a message for confirmation.

using channel.basic_consume, subscribes the coordinator to the'results_queue'.

Summary:

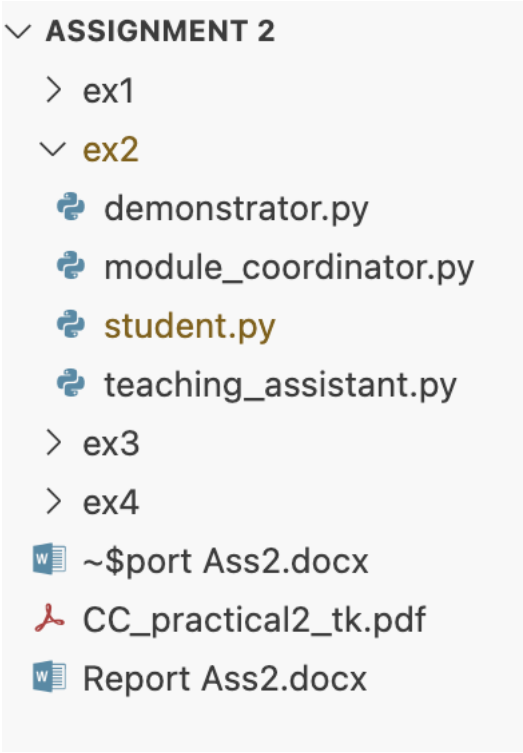
- Message Queues and Exchanges: Knowing how to use RabbitMQ to set up queues and exchanges for communication in a distributed system.
- Publish-Subscribe Pattern: This pattern allows for flexible communication by allowing entities to both publish and subscribe to messages.
- Asynchronous Communication: Understanding the advantages of asynchronous communication can lead to increased system responsiveness and efficiency.
- Routing messages: Ensure correct processing by assigning messages to designated queues according to exchanges and routing keys.
- RabbitMQ with Pika: Getting familiar with RabbitMQ and the Python Pika module for RabbitMQ interaction.
- Understanding Message Flow: Gain knowledge about how messages move across the system by interacting with producer and consumer scripts.
- Developing abilities to recognize and fix problems that arise during the activity is known as debugging and troubleshooting.

EXERCISE 2 :

Steps:

1. Create 4 files and add those data in that files. In ex2 folder.

   Student, demonstrator,
   module_cordinator ,
   teaching_assistance



1. student.py -

The submit_assignment method of the Student class, defined in this script, submits assignments via RabbitMQ.

It connects, starts an exchange labeled "assignment_exchange," publishes an assignment message to the exchange's "correction" queue, and so on.

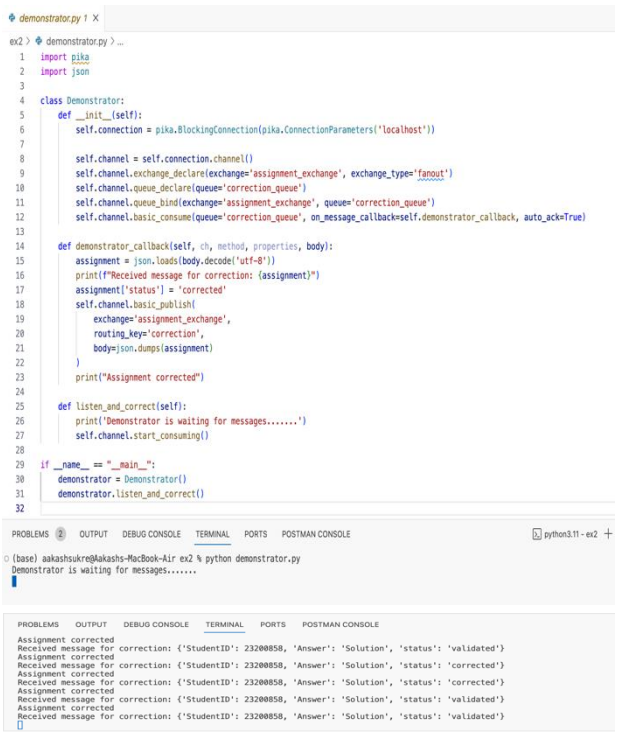
The script then prints a confirmation message and terminates the RabbitMQ connection.

2. Demonstrator.py -

This script defines a Demonstrator class that listens for assignment messages in RabbitMQ.

It processes incoming messages by marking them as 'corrected' and sends them back to the 'assignment_exchange'.

The class sets up a RabbitMQ connection, declares an exchange and a queue, and starts listening for messages until interrupted.



3. teaching_assitance.py -

The script creates a class called `TeachingAssistant}, which serves as a receiver for assignment messages in the 'validation_queue' of RabbitMQ.

As messages are received, it determines if the assignment has been "corrected." If so, it returns it to the "assignment_exchange" after marking it as "validated."

```python
import pika
import json

class TeachingAssistant:
    def __init__(self):
        connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
        self.channel = connection.channel()
        self.channel.exchange_declare(exchange='assignment_exchange', exchange_type='fanout')
        self.channel.queue_declare(queue='validation_queue')
        self.channel.queue_bind(exchange='assignment_exchange', queue='validation_queue')
        self.channel.basic_consume(queue='validation_queue', on_message_callback=self.ta_callback, auto_ack=True)

    def ta_callback(self, ch, method, properties, body):
        assignment = json.loads(body.decode('utf-8'))
        print(f"Received message for validation: {assignment}")

        if assignment['status'] == 'corrected':
            assignment['status'] = 'validated'
            self.channel.basic_publish(
                exchange='assignment_exchange',
                routing_key='validation',
                body=json.dumps(assignment)
            )
            print("Assignment validated")
        else:
            print("Ignoring the assignment as it's not in the 'corrected' state.")

    def start_listening(self):
        print('Teaching Assistant is waiting for messages......')
        self.channel.start_consuming()

if __name__ == "__main__":
```

```
(base) aakashsukre@Aakashs-MacBook-Air ex2 % python teaching_assistant.py
Teaching Assistant is waiting for messages......
```

```
Received message for validation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'corrected'}
Assignment validated
Received message for validation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Ignoring the assignment as it's not in the 'corrected' state.
Received message for validation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Ignoring the assignment as it's not in the 'corrected' state.
Received message for validation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'corrected'}
Assignment validated
Received message for validation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Ignoring the assignment as it's not in the 'corrected' state.
Rece
```

The script establishes a RabbitMQ connection, creates a queue (named "validation_queue") and an exchange (named "assignment_exchange"), binds the queue to the exchange, and configures a callback function (named "ta_callback}) to process incoming messages.

The message-consuming process is started via the `start_listening} method, and the teaching assistant waits for messages until the user interrupts it (CTRL+C).

4. module_cordinator.py -

A ModuleCoordinator class is defined in this script, and it is designed to listen for assignment messages in the RabbitMQ'results_queue'.

```python
import pika
import json

class ModuleCoordinator:
    def __init__(self):
        connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
        self.channel = connection.channel()
        self.channel.exchange_declare(exchange='assignment_exchange', exchange_type='fanout')
        self.channel.queue_declare(queue='results_queue')
        self.channel.queue_bind(exchange='assignment_exchange', queue='results_queue')
        self.channel.basic_consume(queue='results_queue', on_message_callback=self.coordinator_callback, auto_ack=True)

    def coordinator_callback(self, ch, method, properties, body):
        assignment = json.loads(body.decode('utf-8'))
        print(f"Received message for confirmation: {assignment}")

        if assignment['status'] == 'validated':
            assignment['status'] = 'confirmed'
            print(f"Assignment confirmed: {assignment}")
        else:
            print("Ignoring the assignment as it's not in the 'validated' state.")

    def start_listening(self):
        print('Module Coordinator is waiting for messages........')
        self.channel.start_consuming()

if __name__ == "__main__":
    demonstrator = ModuleCoordinator()
    demonstrator.start_listening()
```

```
(base) aakashsukre@Aakashs-MacBook-Air ex2 % python module_coordinator.py
Module Coordinator is waiting for messages........
```

It examines incoming messages, prints a confirmation message, and verifies assignments if they are in the "validated" condition.

```
Received message for confirmation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Assignment confirmed: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'confirmed'}
Received message for confirmation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Assignment confirmed: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'confirmed'}
Received message for confirmation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Assignment confirmed: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'confirmed'}
Received message for confirmation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Assignment confirmed: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'confirmed'}
Received message for confirmation: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'validated'}
Assignment confirmed: {'StudentID': 23200858, 'Answer': 'Solution', 'status': 'confirmed'}
```

After establishing a RabbitMQ connection, the class creates a queue and an exchange and begins to wait for messages until they are interrupted.

Summary:

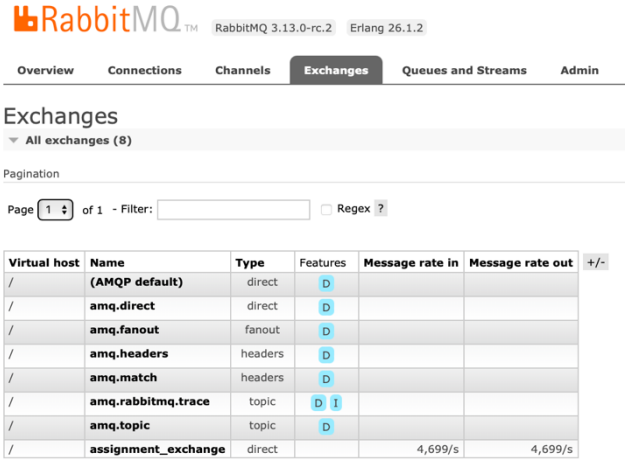
- Configured RabbitMQ as a message queue system.
- examined distributed system communication between the coordinator of the module, the student, the demonstrator, and the teaching assistant.
- Used the asynchronous publish-subscribe technique for communication.
- Exhibited the division of responsibilities in system design.
- Learned about RabbitMQ routing, queue binding, and debugging.
- Effective communication within decentralized systems is essential.
- Asynchronous processing for scalability and fault tolerance.
- Clear system design with clearly defined roles.

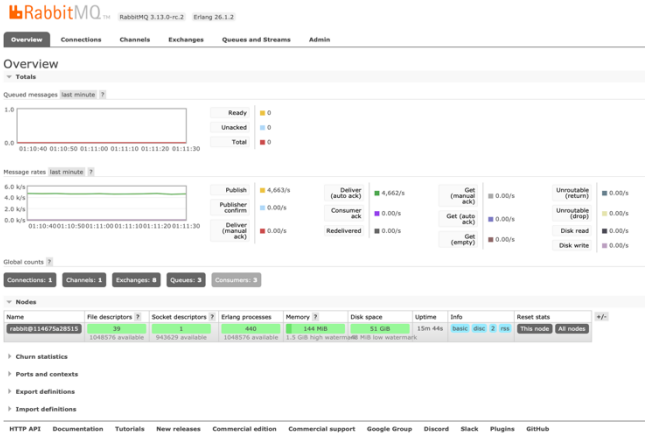
- In conclusion, combining message queues and containerization, Exercise Two offers useful guidance for creating scalable, fault-tolerant, and effective distributed systems.

Output Observation:

1. Open Docker and run RabbitMQ and open http://localhost:15672/#/queues

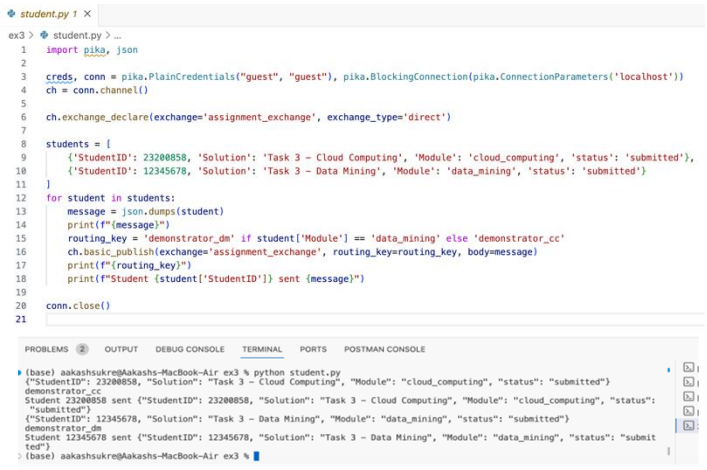
2. Check output in the RabbitMQ localhost in queues where you can see 3 section. Here we can see the outputs in the RabbitMQ software to check the response.





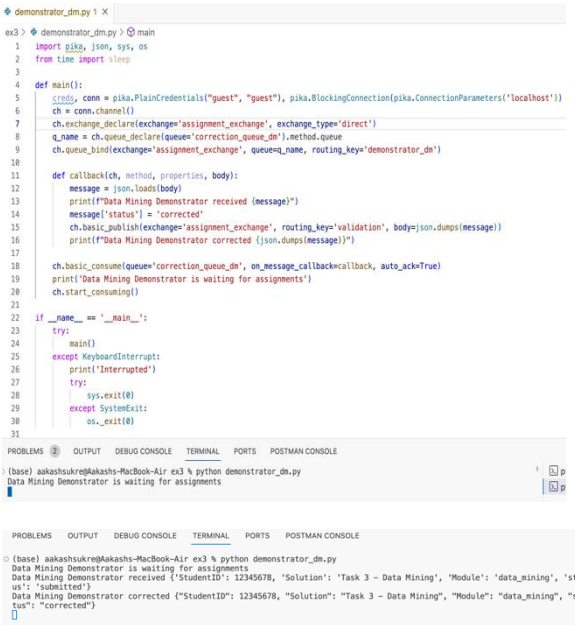
EXERCISE 3 : optimizes the assignment correction system from Exercise 1

1. **Student.py –**



- The module ('data_mining' or 'cloud_computing') is initialized in the Student file.
- The class constructor creates a RabbitMQ channel and connection.
- The defination declares a queue unique to the module and creates a direct exchange called "assignment_exchange."
- The submit_assignment function creates an assignment dictionary and uses RabbitMQ to publish it to the relevant module queue.
- Additionally, a message confirming the submission's success is printed by the procedure.
- After the assignment is turned in, the RabbitMQ connection is terminated.
- It creates an instance of the Student class for the Cloud Computing and Data Mining modules.
- The submit_assignment method is used to turn in assignments for both modules.

2. demonstrator_dm.py -



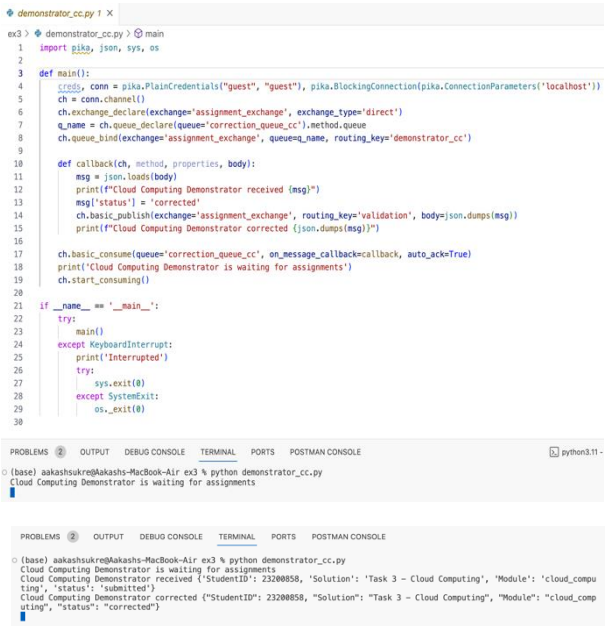
```
import pika, json, sys, os
from time import sleep

def main():
    creds, conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
    q_name = ch.queue_declare(queue='correction_queue_dm').method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_dm')

    def callback(ch, method, properties, body):
        message = json.loads(body)
        print(f"Data Mining Demonstrator received {message}")
        message['status'] = 'corrected'
        ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(message))
        print(f"Data Mining Demonstrator corrected {json.dumps(message)}")

    ch.basic_consume(queue='correction_queue_dm', on_message_callback=callback, auto_ack=True)
    print('Data Mining Demonstrator is waiting for assignments')
    ch.start_consuming()

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

```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python demonstrator_dm.py
Data Mining Demonstrator is waiting for assignments
```



```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python demonstrator_dm.py
Data Mining Demonstrator is waiting for assignments
Data Mining Demonstrator received {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'submitted'}
Data Mining Demonstrator corrected {"StudentID": 12345678, "Solution": "Task 3 - Data Mining", "Module": "data_mining", "status": "corrected"}
```

3. demonstrator_cc.py



```
import pika, json, sys, os

def main():
    creds, conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
    q_name = ch.queue_declare(queue='correction_queue_cc').method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_cc')

    def callback(ch, method, properties, body):
        msg = json.loads(body)
        print(f"Cloud Computing Demonstrator received {msg}")
        msg['status'] = 'corrected'
        ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(msg))
        print(f"Cloud Computing Demonstrator corrected {json.dumps(msg)}")

    ch.basic_consume(queue='correction_queue_cc', on_message_callback=callback, auto_ack=True)
    print('Cloud Computing Demonstrator is waiting for assignments')
    ch.start_consuming()

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

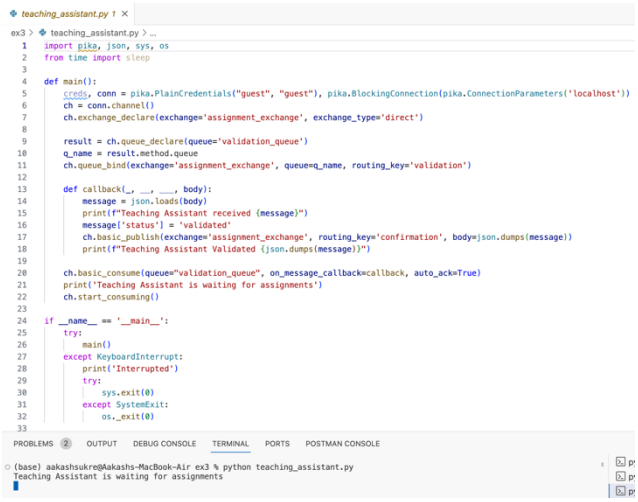
```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python demonstrator_cc.py
Cloud Computing Demonstrator is waiting for assignments
```



```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python demonstrator_cc.py
Cloud Computing Demonstrator is waiting for assignments
Cloud Computing Demonstrator received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'submitted'}
Cloud Computing Demonstrator corrected {"StudentID": 23200858, "Solution": "Task 3 - Cloud Computing", "Module": "cloud_computing", "status": "corrected"}
```

- In the above code which explains a system, demonstrators are represented by the CloudComputingDemonstrator and DataMiningDemonstrator defination.
- They connect to RabbitMQ, define dedicated queues for their modules, start with a 'corrected' state, and wait endlessly for assignments to be sent in.
- When an assignment is received, it is published to the 'validation' queue via the 'assignment_exchange' and its status is changed to 'corrected'.
- Within the entire system, these defination help with assignment correction for the Data Mining and Cloud Computing modules.

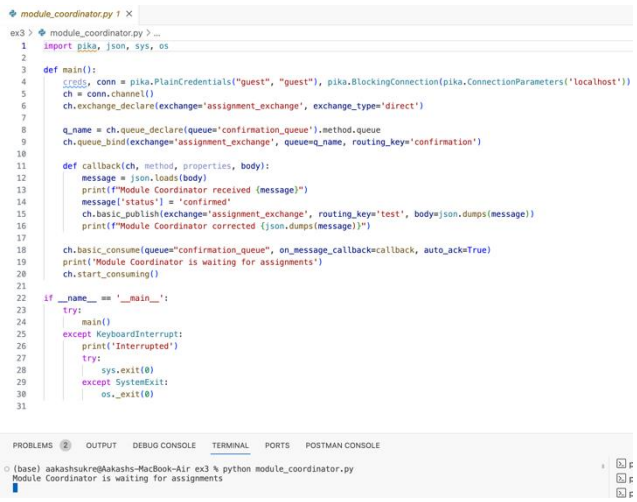
3. teaching_assitance.py -



```python
import pika, json, sys, os
from time import sleep

def main():
    creds, conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')

    result = ch.queue_declare(queue='validation_queue')
    q_name = result.method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='validation')

    def callback(_, __, ___, body):
        message = json.loads(body)
        print(f"Teaching Assistant received {message}")
        message['status'] = 'validated'
        ch.basic_publish(exchange='assignment_exchange', routing_key='confirmation', body=json.dumps(message))
        print(f"Teaching Assistant Validated {json.dumps(message)}")

    ch.basic_consume(queue="validation_queue", on_message_callback=callback, auto_ack=True)
    print('Teaching Assistant is waiting for assignments')
    ch.start_consuming()

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

```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python teaching_assistant.py
Teaching Assistant is waiting for assignments
```

4. module_cordinator.py -



```python
import pika, json, sys, os

def main():
    creds, conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')

    q_name = ch.queue_declare(queue='confirmation_queue').method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='confirmation')

    def callback(ch, method, properties, body):
        message = json.loads(body)
        print(f"Module Coordinator received {message}")
        message['status'] = 'confirmed'
        ch.basic_publish(exchange='assignment_exchange', routing_key='test', body=json.dumps(message))
        print(f"Module Coordinator corrected {json.dumps(message)}")

    ch.basic_consume(queue="confirmation_queue", on_message_callback=callback, auto_ack=True)
    print('Module Coordinator is waiting for assignments')
    ch.start_consuming()

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

```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python module_coordinator.py
Module Coordinator is waiting for assignments
```

```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python module_coordinator.py
Module Coordinator is waiting for assignments
Module Coordinator received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'validated'}
Module Coordinator corrected {"StudentID": 23200858, "Solution": "Task 3 - Cloud Computing", "Module": "cloud_computing", "status": "confirmed"}
Module Coordinator received {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'validated'}
Module Coordinator corrected {"StudentID": 12345678, "Solution": "Task 3 - Data Mining", "Module": "data_mining", "status": "confirmed"}
```

```
(base) aakashsukre@Aakashs-MacBook-Air ex3 % python teaching_assistant.py
Teaching Assistant is waiting for assignments
Teaching Assistant received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'corrected'}
Teaching Assistant Validated {"StudentID": 23200858, "Solution": "Task 3 - Cloud Computing", "Module": "cloud_computing", "status": "validated"}
Teaching Assistant received {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'corrected'}
Teaching Assistant Validated {"StudentID": 12345678, "Solution": "Task 3 - Data Mining", "Module": "data_mining", "status": "validated"}
```

- The TeachingAssistant and ModuleCordinator python scripts creates a RabbitMQ connection and channel for communication.
- The def establishes a validation-specific queue (called "validation_queue") and (‘confirmation_queue’) and creates a direct exchange called "assignment_exchange."
- The 'validation_queue' and ‘confirmation_queue’ are the place to continuously listen for incoming assignments.
- The callback method is initiated upon receiving a new assignment with the status 'corrected' and ‘validated’. JSON-formatted messages.
- Next, using the 'assignment_exchange', the assignment is republished to the 'confirmation' routing key with its status updated to 'validated'.
- The above both script stays active and prepared to validate new assignments by starting the message consumption process with the start_consuming method.
- In the event of an interruption, exception handling guarantees a proper exit using os._exit(0) or sys.exit(0).

Teaching Assistance :

Function:	Provides as the system's representation of a teaching assistant.
Responsibility:	The checks corrected assignments and listens for assignments in the validation_queue.
Status Update:	'assignment_exchange' modifies the assignment status to 'verified' and passes it to the 'confirmation' routing key.

Module Cordinator :

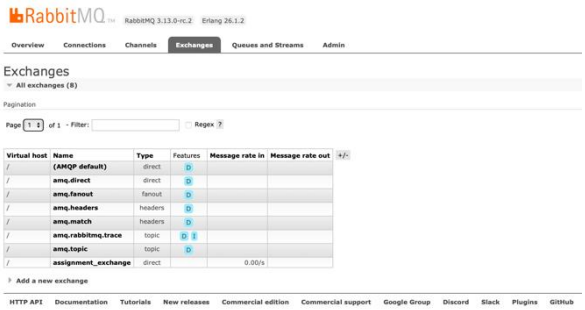
Function:	In the system, this entity represents a Module Coordinator.
Responsibility:	Gives attention to assignments in the confirmation_queue and verifies assignments that have been validated.
Status Update:	'confirmed' is the new assignment status, which is forwarded to the 'confirmation' routing key via the 'assignment_exchange'.

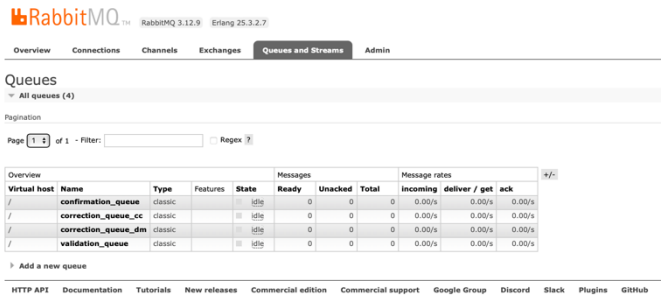
Summary:

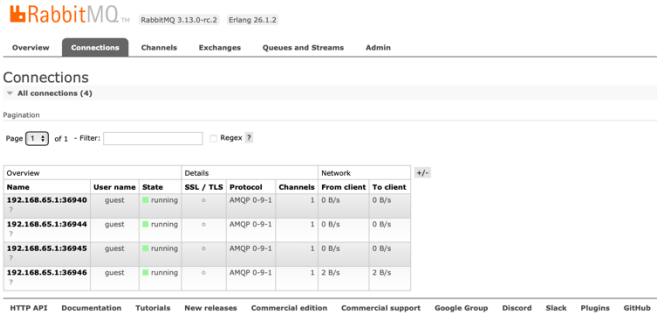
- Its main objective is to distribute tasks to demonstrators in an effective manner accor -ding to their modules.
- Students, teaching assistants, coordinators of the modules, and demonstrators of cloud computing and data mining are the entities participating. Assignments submitted by students are forwarded to demonstrators and queues that are appropriate.
- In order to minimize resource waste, the system makes sure assignments are only handled by appropriate groups.
- The aim is to construct this optimized scenario in Python using Pika and RabbitMQ. The goal is to present a simplified and more effective assignment correction procedure.

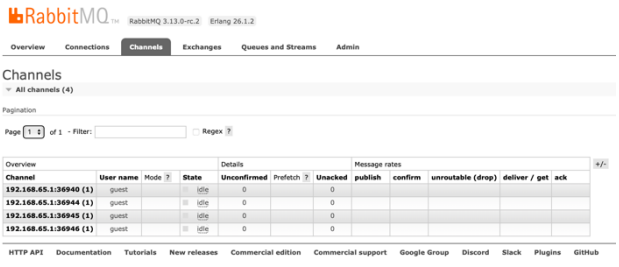
Output Observation:

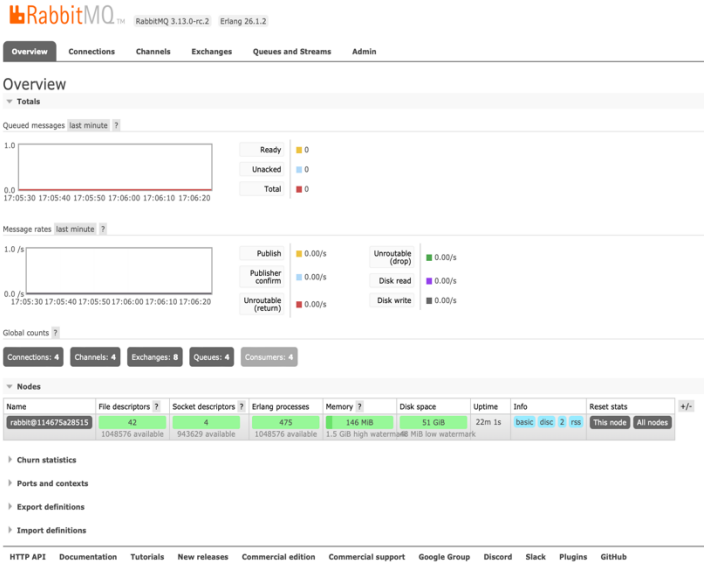
Here, we can see every tab from the RabbitMQ program, providing us with information about how the exercise 3 processes link with each other conveniently and efficiently optimize the system.

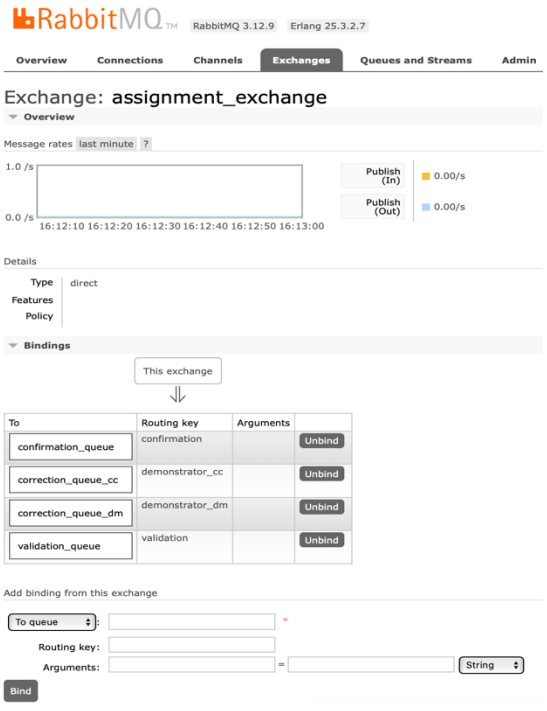






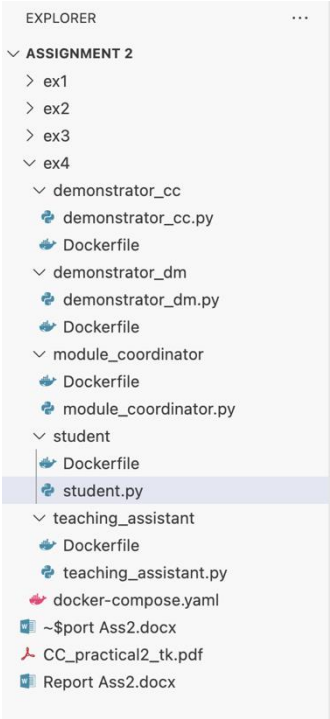






EXERCISE 4 :

Steps:



1. Create 5 folders and add those 5 files in that files. In ex4 folder.

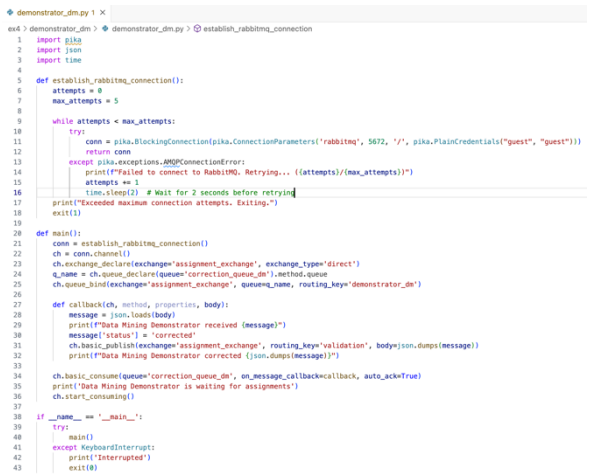
   Student,
   demonstrator_cc,
   demonstrator_dm,
   module_cordinator ,
   teaching_assistance,

2. add 5 Docker files in each folder respectively Dockerfile

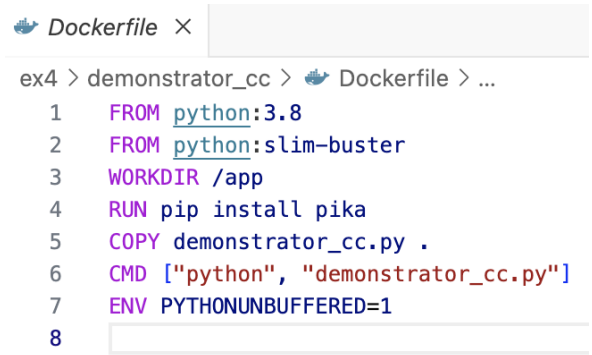
3. Create docker-compose.yml file

Here we can see the below files and their code respectively.

1. **demonstrator_dm.py Docker file**

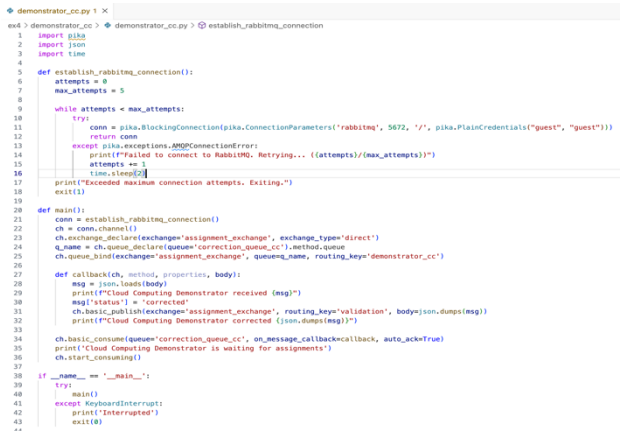


```python
import pika
import json
import time

def establish_rabbitmq_connection():
    attempts = 0
    max_attempts = 5

    while attempts < max_attempts:
        try:
            conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials("guest", "guest")))
            return conn
        except pika.exceptions.AMQPConnectionError:
            print(f"Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})")
            attempts += 1
            time.sleep(2)  # Wait for 2 seconds before retrying
    print("Exceeded maximum connection attempts. Exiting.")
    exit(1)

def main():
    conn = establish_rabbitmq_connection()
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
    q_name = ch.queue_declare(queue='correction_queue_dm').method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_dm')

    def callback(ch, method, properties, body):
        message = json.loads(body)
        print(f"Data Mining Demonstrator received {message}")
        message['status'] = 'corrected'
        ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(message))
        print(f"Data Mining Demonstrator corrected {json.dumps(message)}")

    ch.basic_consume(queue='correction_queue_dm', on_message_callback=callback, auto_ack=True)
    print('Data Mining Demonstrator is waiting for assignments')
    ch.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        exit(0)
```



```
Dockerfile
ex4 > demonstrator_cc > Dockerfile > ...
1   FROM python:3.8
2   FROM python:slim-buster
3   WORKDIR /app
4   RUN pip install pika
5   COPY demonstrator_cc.py .
6   CMD ["python", "demonstrator_cc.py"]
7   ENV PYTHONUNBUFFERED=1
8
```

2. demonstrator_cc.py Docker file



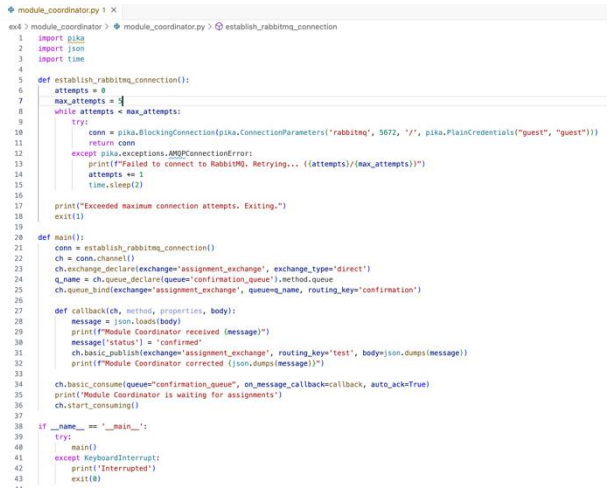
```python
import pika
import json
import time

def establish_rabbitmq_connection():
    attempts = 0
    max_attempts = 5

    while attempts < max_attempts:
        try:
            conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials("guest", "guest")))
            return conn
        except pika.exceptions.AMQPConnectionError:
            print(f"Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})")
            attempts += 1
            time.sleep(2)
    print("Exceeded maximum connection attempts. Exiting.")
    exit(1)

def main():
    conn = establish_rabbitmq_connection()
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
    q_name = ch.queue_declare(queue='correction_queue_cc').method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_cc')

    def callback(ch, method, properties, body):
        msg = json.loads(body)
        print(f"Cloud Computing Demonstrator received {msg}")
        msg['status'] = 'corrected'
        ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(msg))
        print(f"Cloud Computing Demonstrator corrected {json.dumps(msg)}")

    ch.basic_consume(queue='correction_queue_cc', on_message_callback=callback, auto_ack=True)
    print('Cloud Computing Demonstrator is waiting for assignments')
    ch.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        exit(0)
```

Dockerfile

ex4 > demonstrator_cc > Dockerfile > ...

```
FROM python:3.8
FROM python:slim-buster
WORKDIR /app
RUN pip install pika
COPY demonstrator_cc.py .
CMD ["python", "demonstrator_cc.py"]
ENV PYTHONUNBUFFERED=1
```

3. module_coordinator.py and Docker file



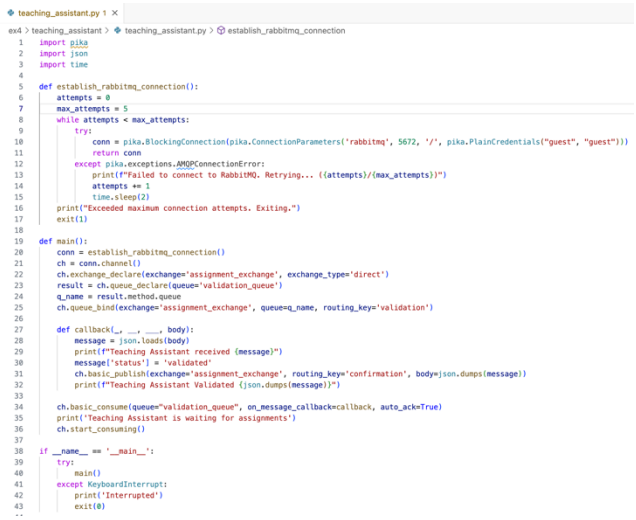
```python
import pika
import json
import time

def establish_rabbitmq_connection():
    attempts = 0
    max_attempts = 5
    while attempts < max_attempts:
        try:
            conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials("guest", "guest")))
            return conn
        except pika.exceptions.AMQPConnectionError:
            print(f"Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})")
            attempts += 1
            time.sleep(2)

    print("Exceeded maximum connection attempts. Exiting.")
    exit(1)

def main():
    conn = establish_rabbitmq_connection()
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
    q_name = ch.queue_declare(queue='confirmation_queue').method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='confirmation')

    def callback(ch, method, properties, body):
        message = json.loads(body)
        print(f"Module Coordinator received {message}")
        message['status'] = 'confirmed'
        ch.basic_publish(exchange='assignment_exchange', routing_key='test', body=json.dumps(message))
        print(f"Module Coordinator corrected {json.dumps(message)}")

    ch.basic_consume(queue="confirmation_queue", on_message_callback=callback, auto_ack=True)
    print('Module Coordinator is waiting for assignments')
    ch.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        exit(0)
```

Dockerfile

ex4 > module_coordinator > Dockerfile > ...

```
FROM python:3.8
FROM python:slim-buster
WORKDIR /app
RUN pip install pika
COPY module_coordinator.py .
CMD ["python", "module_coordinator.py"]
ENV PYTHONUNBUFFERED=1
```

4. teaching_assitant.py and Docker file



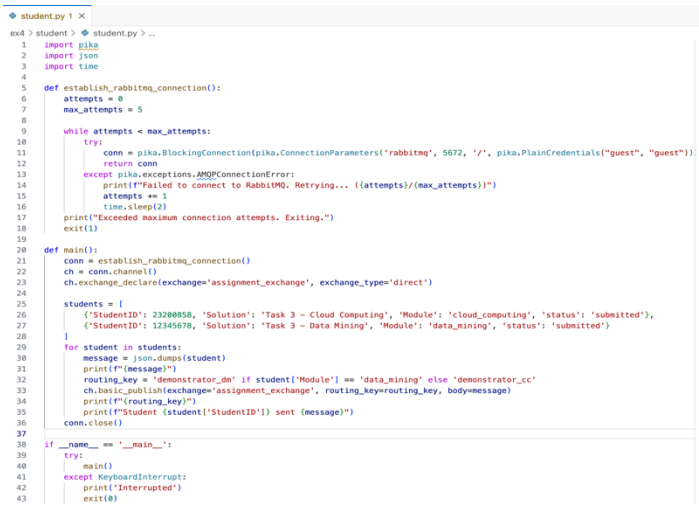
```python
import pika
import json
import time

def establish_rabbitmq_connection():
    attempts = 0
    max_attempts = 5
    while attempts < max_attempts:
        try:
            conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials("guest", "guest")))
            return conn
        except pika.exceptions.AMQPConnectionError:
            print(f"Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})")
            attempts += 1
            time.sleep(2)
    print("Exceeded maximum connection attempts. Exiting.")
    exit(1)

def main():
    conn = establish_rabbitmq_connection()
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
    result = ch.queue_declare(queue='validation_queue')
    q_name = result.method.queue
    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='validation')

    def callback(_, __, ___, body):
        message = json.loads(body)
        print(f"Teaching Assistant received {message}")
        message['status'] = 'validated'
        ch.basic_publish(exchange='assignment_exchange', routing_key='confirmation', body=json.dumps(message))
        print(f"Teaching Assistant Validated {json.dumps(message)}")

    ch.basic_consume(queue="validation_queue", on_message_callback=callback, auto_ack=True)
    print('Teaching Assistant is waiting for assignments')
    ch.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        exit(0)
```

Dockerfile

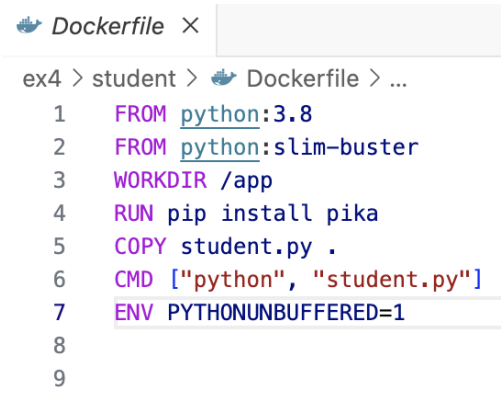
ex4 > teaching_assistant > Dockerfile > ...

```
FROM python:3.8
FROM python:slim-buster
WORKDIR /app
RUN pip install pika
COPY teaching_assistant.py .
CMD ["python", "teaching_assistant.py"]
ENV PYTHONUNBUFFERED=1
```

5. student.py and Docker file



```python
import pika
import json
import time

def establish_rabbitmq_connection():
    attempts = 0
    max_attempts = 5

    while attempts < max_attempts:
        try:
            conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials("guest", "guest")))
            return conn
        except pika.exceptions.AMQPConnectionError:
            print(f"Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})")
            attempts += 1
            time.sleep(2)
    print("Exceeded maximum connection attempts. Exiting.")
    exit(1)

def main():
    conn = establish_rabbitmq_connection()
    ch = conn.channel()
    ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')

    students = [
        {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'submitted'},
        {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'submitted'}
    ]
    for student in students:
        message = json.dumps(student)
        print(f"{message}")
        routing_key = 'demonstrator_dm' if student['Module'] == 'data_mining' else 'demonstrator_cc'
        ch.basic_publish(exchange='assignment_exchange', routing_key=routing_key, body=message)
        print(f"{routing_key}")
        print(f"Student {student['StudentID']} sent {message}")
    conn.close()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        exit(0)
```



Dockerfile

ex4 > student > Dockerfile > ...

```
1  FROM python:3.8
2  FROM python:slim-buster
3  WORKDIR /app
4  RUN pip install pika
5  COPY student.py .
6  CMD ["python", "student.py"]
7  ENV PYTHONUNBUFFERED=1
8
9
```

Python Script file:

The above all python script files contains below description of the code :

1. establish_connection function:
- This attempts to establish a connection to RabbitMQ.
- This retries up to five times, pausing every two seconds in between.
- Prints a message if the connection fails after each attempt.

2. main function:
- Calls establish_connection to get a RabbitMQ connection.
- Creates a channel, declares an exchange, and sets up a queue and binding.
- Establishes a queue and binding, declares an exchange, and creates a channel.
Callback is invoked by consuming messages from the 'correction_queue_cc' queue.
- Prints messages when assignments are received and corrected.
- Starts reading messages and biding wait for assignments.

3. if __name__ == '__main__':
- When the script is run, it carries out the main function.

Docker File:

The above all Docker files contains below description of the code :

By using Docker's default bridge network, communication between containers in a Dockerized environment can be easily established by substituting 'rabbitmq' for 'localhost' in the code. Especially when containers are connected within the same network, it offers a reliable and portable method of referencing the RabbitMQ server.

1. Base Image (FROM): Indicates where the image begins.
2. The Working Directory (WORKDIR): establishes the directory for all commands that follow.
3. Dependencies Installation (RUN): Uses commands to set up dependencies or install programs.
4. File Copy (COPY): Transfers data from the local computer to the virtual machine.
5. Command (CMD): Specifies the initial command that a container will execute upon start-up.

1. Docker-compose.yml

docker-compose.yaml

ex4 > docker-compose.yaml

```yaml
version: '3'
networks:
  rabbitmq_go_net:
    driver: bridge

services:
  rabbitmq:
    image: "rabbitmq:3.13-rc-management"
    container_name: rabbitmq_container
    ports:
      - "5672:5672"
      - "15672:15672"
    environment:
      RABBITMQ_DEFAULT_USER: guest
      RABBITMQ_DEFAULT_PASS: guest
    networks:
      - rabbitmq_go_net

  demonstrator_cc:
    build: demonstrator_cc
    command: python ./demonstrator_cc.py
    restart: unless-stopped
    depends_on:
      - rabbitmq
    networks:
      - rabbitmq_go_net

  demonstrator_dm:
    build: demonstrator_dm
    command: python ./demonstrator_dm.py
    restart: unless-stopped
    depends_on:
      - rabbitmq
    networks:
      - rabbitmq_go_net

  teaching_assistant:
    build: teaching_assistant
    restart: unless-stopped
    command: python ./teaching_assistant.py
    depends_on:
      - rabbitmq
    networks:
      - rabbitmq_go_net

  module_coordinator:
    build: module_coordinator
    restart: unless-stopped
    command: python ./module_coordinator.py
    depends_on:
      - rabbitmq
    networks:
      - rabbitmq_go_net

  student:
    build: student
    depends_on:
      - rabbitmq
      - demonstrator_cc
      - demonstrator_dm
      - teaching_assistant
      - module_coordinator
    stdin_open: true
    tty: true
    networks:
      - rabbitmq_go_net
```

Here is the explanation of the above compose file:

The official RabbitMQ image is used to create a RabbitMQ service with default credentials and open messaging ports.

The above file involves five additional services that represent different entities: teaching assistant, demonstrator_dm, demonstrator_cc, module_coordinator, and student. Every service depends on the RabbitMQ service, builds from its own directory, and gives an instruction to run a Python script at startup.

To guarantee appropriate management and service availability, dependencies and restart policies are established on each service, and they are all connected to a unique bridge network called rabbitmq_go_net.

Summary:

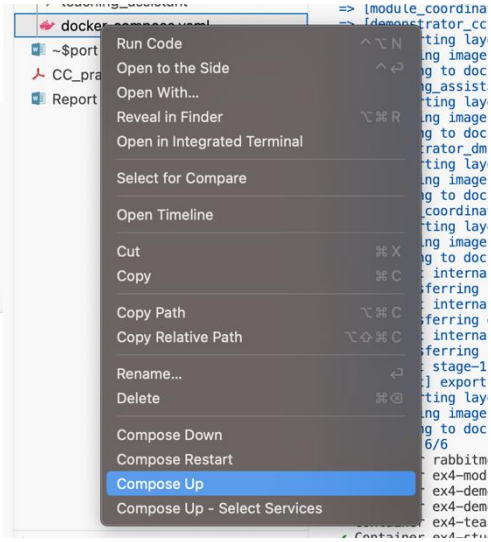
This above method makes it possible to create a reproducible and controlled environment for teaching purposes, showcasing the concepts of automation, containerization, and microservices communication via Docker and RabbitMQ.

Output Observation:

Here, we can see information about how the exercise 4 processes link with each other conveniently and efficiently optimize the system via Docker and RabbitMQ

1. Build and compose UP

```
=> [student stage-1 4/4] COPY student.py .
=> [student] exporting to image
=> => exporting layers
=> => writing image sha256:1f4b1742f93de20bcfe299dd6c52006088e186c1a39e5776602855708a22bb1d
=> => naming to docker.io/library/ex4-student
[+] Running 6/6
 ✔ Container rabbitmq_container          Started
 ✔ Container ex4-module_coordinator-1    Started
 ✔ Container ex4-demonstrator_dm-1       Started
 ✔ Container ex4-demonstrator_cc-1       Started
 ✔ Container ex4-teaching_assistant-1    Started
 ✔ Container ex4-student-1               Started
 * Terminal will be reused by tasks, press any key to close it.
```



2. Docker Execution Picture