Assignment 04

BRAC UNIVERSITY

Abir Ahammed Bhuiyan
ID: 20101197
Dept. of Computer Science and Engineering

Section : 01
Subject : CSE484
Semester : Spring-2024
Date : 30.03.2024

What is RabbitMQ?

RabbitMQ is an open-source message broker software that facilitates communication between different systems by implementing the `Advanced Message Queuing Protocol (AMQP)`. It acts as a mediator for passing messages between applications, providing a reliable and scalable platform for asynchronous messaging. RabbitMQ enables decoupling of components, ensuring efficient communication in distributed systems.

Task 1: "Hello World!" - The simplest thing that does

Setup

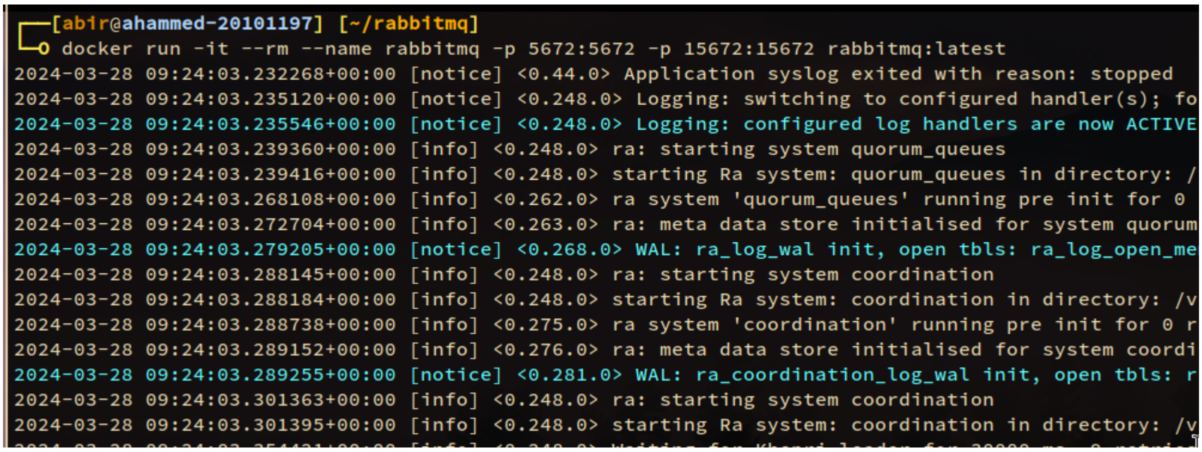
First we have to install `RabbitMQ` in our machine and make sure it is running on the standard port `5672`. We will be using docker container instead of installing `RabbitMQ` in our system.

```
docker pull rabbitmq
```

```
┌─[abir@ahammed-20101197] [~/rabbitmq]
└──o docker pull rabbitmq
Using default tag: latest
latest: Pulling from library/rabbitmq
bccd10f490ab: Pull complete
f1000ca5c91d: Pull complete
15fd8d52bc6c: Pull complete
d9b381d4c87a: Pull complete
497a9eb5f435: Pull complete
dd2bd0cede52: Pull complete
bed3197826ba: Pull complete
0c1d09ec487d: Pull complete
a89de7acba35: Pull complete
74d11b8768c5: Pull complete
b8a34a89caa8: Pull complete
Digest: sha256:3794b07f7c85e010f464badb564100f752224107977dd00d5e2e26e3b18f76a7
Status: Downloaded newer image for rabbitmq:latest
docker.io/library/rabbitmq:latest
┌─[abir@ahammed-20101197] [~/rabbitmq]
└──o
```

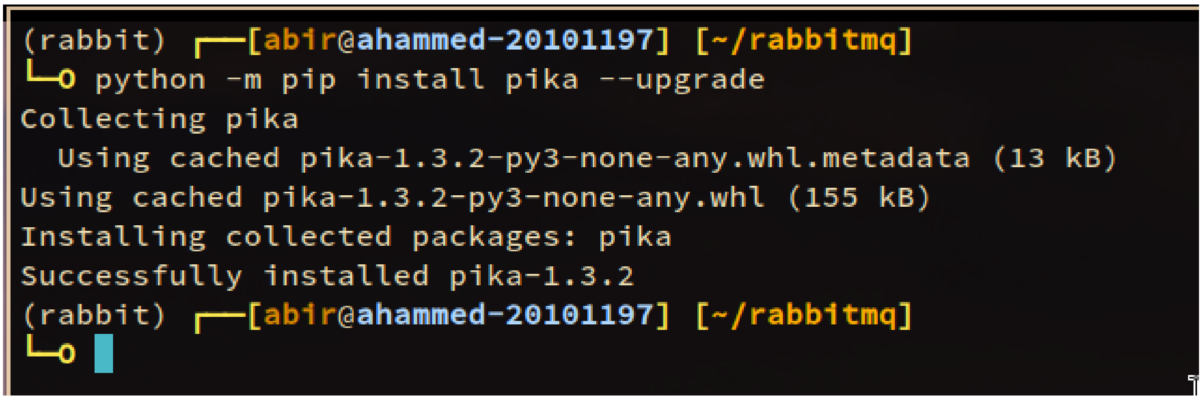
After pulling `Rabbitmq` docker image we should run this image in detached mode (making sure the port 5672 is mapped)

```
docker run -it --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:latest
```



We will be using `python` language along with `pika` client for this experiment, so we should install this package using `pip`.

```
python -m pip install pika --upgrade
```



Our test bed is ready, now let's write some code.

Code

Let's write `send.py` or from where the message will be transmitted. The pseudocode and also the code for publishing is given below,

1. make connection
2. create a channel
3. declare a queue
4. publish the message
5. close the connection

```python
#!/usr/bin/env python
import pika, sys

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
message = ' '.join(sys.argv[1:]) or "Hello World!"
channel.basic_publish(exchange='',
                      routing_key='hello',
                      body=message)
print(" [x] Sent {message}")
```

Let's write `receive.py` or the program that will recieve the message. The pseudocode and also the code for consuming is given below,

1. declare a main method
2. in the main method create a connection
3. in the main method create a channel
4. in the main method define a queue
5. in the main method define a callback method
6. in the main consume the message using the callback method
7. at the end handle exception for breaking out of the code

```python
#!/usr/bin/env python
import pika, sys, os

def main():
    connection = 
pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='hello')

    def callback(ch, method, properties, body):
        print(f" [x] Received {body}")

    channel.basic_consume(queue='hello', on_message_callback=callback, 
auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

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

Run

Let's run the code to test our theory.

In the left terminal we are running the sender program and in the right terminal we are running the receiving program. After running the `recieve.py` we run `send.py` containing message. From the image we can see we have sent `Hello World Human!` and it was received by the receiver program.

Task 2: "Work queues"- Distributing tasks among workers

Theory

The main idea behind Work Queues (aka: *Task Queues*) is to avoid doing a resource-intensive task immediately and having to wait for it to complete. Instead we schedule the task to be done later. We encapsulate a *task* as a message and send it to the queue. A worker process running in the background will pop the tasks and eventually execute the job. When we run many workers the tasks will be shared between them. One of the advantages of using a Task Queue is the ability to easily parallelise work.

In the previous tutorial our task was straight forward but for this tutorial we have to mimic real world complex task to do that we will be using `time.sleep()` to make it seem the workers are performing some heavy tasks. We'll take the number of dots in the string as its complexity; every dot will account for one second of "work". For example, a fake task described by `Hello...` will take three seconds.

Code

The code for `new_task.py` is,

```
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)

message = ' '.join(sys.argv[1:]) or "Hello World!"
channel.basic_publish(
```

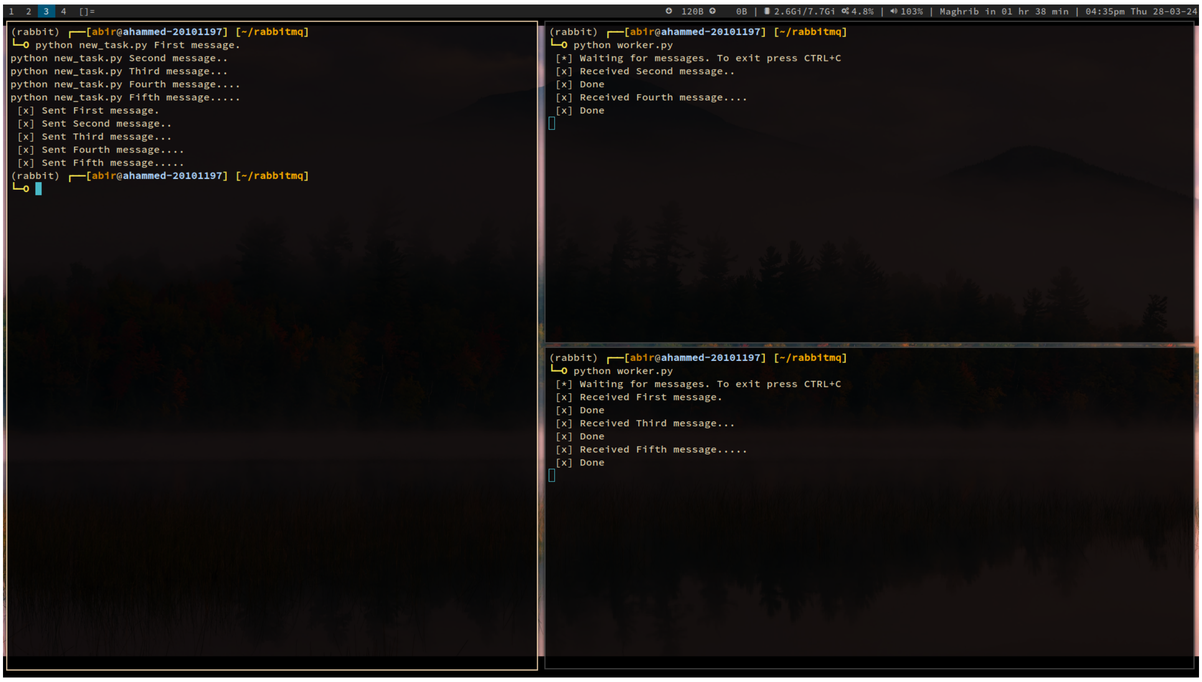
```
    exchange='',
    routing_key='task_queue',
    body=message,
    properties=pika.BasicProperties(
        delivery_mode=pika.DeliveryMode.Persistent
    ))
print(f" [x] Sent {message}")
connection.close()
```

The code for `worker.py` is,

```
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(f" [x] Received {body.decode()}")
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()
```

Run

We will create three terminals or three shells among them one will be running `new_task.py` or send works and the two workers will perform the tasks parallelly.


1 2 3 4 []=
120B 0B | 2.6Gi/7.7Gi 4.8% | 103% | Maghrib in 01 hr 38 min | 04:35pm Thu 28-03-24
(rabbit) ┌─[abir@ahammed-20101197] [~/rabbitmq]
└─o python new_task.py First message.
python new_task.py Second message..
python new_task.py Third message...
python new_task.py Fourth message....
python new_task.py Fifth message.....
[x] Sent First message.
[x] Sent Second message..
[x] Sent Third message...
[x] Sent Fourth message....
[x] Sent Fifth message.....
(rabbit) ┌─[abir@ahammed-20101197] [~/rabbitmq]
└─o
(rabbit) ┌─[abir@ahammed-20101197] [~/rabbitmq]
└─o python worker.py
[*] Waiting for messages. To exit press CTRL+C
[x] Received Second message..
[x] Done
[x] Received Fourth message....
[x] Done
(rabbit) ┌─[abir@ahammed-20101197] [~/rabbitmq]
└─o python worker.py
[*] Waiting for messages. To exit press CTRL+C
[x] Received First message.
[x] Done
[x] Received Third message...
[x] Done
[x] Received Fifth message.....
[x] Done