Quality measurement system example

We send states to a message queue named 'qualityQueue', which stores simple status messages of a quality assurance system. Create a multi-component application that communicates with the message queue through two clients in the following way:

1. The first client connects to the sensor placed on the measuring machine and randomly sends GOOD, EXCELLENT, and WRONG messages to the 'qualityQueue' message queue every second.
2. Create a component that reads and collects the 'GOOD', 'EXCELLENT', and 'WRONG' messages from the qualityQueue queue. After receiving 10 identical messages, it sends a message to the 'qualityStatistics' queue indicating that it has processed 10 messages of a certain quality.
3. Create a second client that reads the statistics from the 'qualityStatistics' queue and prints to the console, for example, '10 'WRONG' messages has been processed'.

flowchart TB MQ[("RabbitMQ Server\n(qualityQueue, qualityStatistics)")] Component1["Component 1\n(Sensor Data Sender)"] -->|sends GOOD/EXCELLENT/WRONG| MQ Component2["Component 2\n(Quality Message Consumer)"] -- collects messages --> MQ Component2 -->|sends batch of 10 messages| MQ Component3["Component 3\n(Statistics Consumer)"] -- receives and prints batches --> MQ subgraph Docker MQ end subgraph Components Component1 Component2 Component3 end classDef machine fill:#f9f,stroke:#333,stroke-width:2px; classDef clients fill:#ccf,stroke:#333,stroke-width:2px; class Docker machine; class Component1,Component2,Component3 clients;

Let's try to solve the task with http://docker.iit.uni-miskolc.hu framework.

Starting RabbitMQ in Docker

To solve the task, it is recommended to start multiple instances (terminals). The first terminal will start the RabbitMQ server. Open a new terminal (node 1) and run the following command:

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

After running the command above, the RabbitMQ management console will be accessible on port 15672, using the credentials guest/guest. In the left-side menu, you can find the internal IP address (**10.x.y.z**) of node1, which can be used in the clients and processors.

Create another terminal and execute the following command:

```
pip install pika
```

This command installs the pika module, which provides the connection to RabbitMQ.

Create the *quality_message_sender.py*:

Use the appropriate IP address in the *init(self):* method.

```
import pika
```

```
import random
import time

class QualitySender:
    def __init__(self):
        self.connection =
pika.BlockingConnection(pika.ConnectionParameters('10.x.y.z'))
        self.channel = self.connection.channel()
        self.channel.queue_declare(queue='qualityQueue')

    def start_sending(self):
        qualities = ['GOOD', 'EXCELLENT', 'WRONG']
        while True:
            quality = random.choice(qualities)
            self.channel.basic_publish(exchange='',
routing_key='qualityQueue', body=quality)
            print(f'Sent quality: {quality}')
            time.sleep(1)

    def close_connection(self):
        self.connection.close()

if __name__ == '__main__':
    sender = QualitySender()
    try:
        sender.start_sending()
    except KeyboardInterrupt:
        sender.close_connection()
```

Let's create the *quality_message_consumer.py* file:

(do not forget to create it in an other terminal, and run *pip install pika* and set the proper IP in *pika.ConnectionParameters()*)

```
import pika

class QualityConsumer:
    def __init__(self):
        self.connection =
pika.BlockingConnection(pika.ConnectionParameters('10.x.y.z'))
        self.channel = self.connection.channel()
        self.channel.queue_declare(queue='qualityQueue')
        self.channel.queue_declare(queue='qualityStatistics')
        self.message_count = {'GOOD': 0, 'EXCELLENT': 0, 'WRONG': 0}

    def start_consuming(self):
        def callback(ch, method, properties, body):
            quality = body.decode()
```

```
            self.message_count[quality] += 1
            print(f'Received quality: {quality}')
            if self.is_batch_completed():
                self.send_statistics()
                self.reset_message_count()

        self.channel.basic_consume(queue='qualityQueue',
on_message_callback=callback, auto_ack=True)
        self.channel.start_consuming()

    def send_statistics(self):
        for quality, count in self.message_count.items():
            if count > 0:
                message = f'{count} {quality} messages has been processed'
                self.channel.basic_publish(exchange='',
routing_key='qualityStatistics', body=message)
                print(f'Sent statistics: {message}')

    def reset_message_count(self):
        for quality in self.message_count:
            self.message_count[quality] = 0

    def is_batch_completed(self):
        return sum(self.message_count.values()) >= 10

    def close_connection(self):
        self.connection.close()

if __name__ == '__main__':
    consumer = QualityConsumer()
    try:
        consumer.start_consuming()
    except KeyboardInterrupt:
        consumer.close_connection()
```

The third components prints the statistics. Let's create an other instance (terminal) and create statistics_consumer.py

<sxh python> import pika

RabbitMQ settings connection = pika.BlockingConnection(pika.ConnectionParameters('10.x.y.z')) channel = connection.channel()

channel.queue_declare(queue='qualityStatistics')

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

```
  message = body.decode()
  print(f'{message}')
  ch.basic_ack(delivery_tag=method.delivery_tag)
```

channel.basic_consume(queue='qualityStatistics', on_message_callback=callback)

print('Waiting for quality statistics...') channel.start_consuming() </shx>