Összetettebb példa

Egy minőségbiztosító rendszer mérőgépének 3 állapotát küldjük egy 'qualityQueue' nevű üzenetsorra. Készítsen egy több komponensből álló alkalmazást, amely 2 kliensen keresztül kommunikál az üzenetsorral az alábbi módon:

- Az első komponenst, ami a mérőgépre helyezett érzékelőre kapcsolódik a '**qualityQueue**' üzenetsorra pont-pont csatlakozással véletlenszerűen GOOD, EXCELLENT és WRONG üzeneteket küld másodpercenként.
- **Készítsen egy második komponenst** amely a 'GOOD', 'EXCELLENT' és a 'WRONG' üzeneteket leolvassa a **qualityQueue** sorról és gyűjti. Minden 10 megkapott azonos üzenet után a '**qualityStatistics**' sorra küld egy üzenetet, amiben azt jelzi, hogy 10 (adott minőségű) üzenetet feldolgozott.
- **Készítsen egy harmadik komponenst**, ami a 'qualityStatistics' sorrol olvassa a statisztikát és a konzolba kiírja hogy pl. '10 'WRONG' messages has been processed'

flowchart TB MQ[("RabbitMQ Server\n(qualityQueue, qualityStatistics)")] Client1["Component 1\n(Sensor Data Sender)"] -->|sends GOOD/EXCELLENT/WRONG| MQ Client2["Component 2\n(Quality Message Consumer)"] -- collects messages --> MQ Client2 -->|sends batch of 10 messages| MQ Client3["Component 3\n(Statistics Consumer)"] -- receives and prints batches --> MQ subgraph Docker MQ end subgraph Components Client1 Client2 Client3 end classDef machine fill:#f9f,stroke:#333,stroke-width:2px; classDef clients fill:#ccf,stroke:#333,stroke-width:2px; class Docker machine; class Client1,Client2,Client3 clients;

A fenti feladatot a http://docker.iit.uni-miskolc.hu keretrendszerben oldjuk meg.

RabbitMQ indítása docker-ben

A feladat megoldásához több instance-t (konzolt) érdemes indítani. Az első konzol fogja a rabbitMQ szervert indítani. Adjunk hozzá egy konzolt (node 1) és futtassuk a következő parancsot:

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

A futtatás után a rabitMQ management konzol elérhető az 15672-es porton, a guest/guest megadásával. A bal oldali listában láthatjuk a node1 10.x.y.z belső IP címét, amit használhatunk a kliensekben és a feldolgozóban.

Hozzunk létre egy másik konzolt és indítsuk el az alábbi parancsot:

```
pip install pika
```

Ezzel telepítettük a pika modult, ami a rabbitMQ-hoz való csatlakozást biztosítja.

Hozzuk létre a quality_message_sender.py-t:

Használjuk a megfelelő IP-t a init*(self):* ben

```
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()
```

A második komponenshez indítsunk egy új konzolt:

A init*(self):* konstruktorban állítsuk be a rabbitMQ szerver IP címét

```
import pika

class QualityConsumer:
    def __init__(self):
        self.connection =
pika.BlockingConnection(pika.ConnectionParameters('localhost'))
        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()
```

Készítsük el a statisztika kiírását egy új konzolban:

```
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()
```

Feladat:

A 15672-es porton lépjük be a rabbitMQ management console-ra és vizsgáljuk meg a lehetőségeit.