Exercise 9

Event publishing with RabbitMQ

Prior Knowledge

Previous exercises

Objectives

Understanding queuing and pub-sub models

Software Requirements

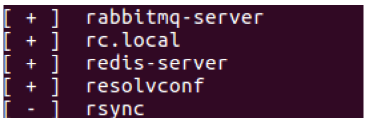
- Java Development Kit 8
- RabbitMQ
- Python

Overview

We will update the Purchase service to publish every new order to a RabbitMQ server. We will use a Python client to subscribe to this.

Steps

1. Create a directory for this work and clone the git repository:
```
mkdir ~/ex9
cd ~/ex9
git clone https://github.com/pzfreo/PSBComplete.git
```

2. Make sure redis and rabbitmq-server are both running locally:
```
service --status-all
```

```
[ + ]  rabbitmq-server
[ + ]  rc.local
[ + ]  redis-server
[ + ]  resolvconf
[ - ]  rsync
```

3. Take a look at the code. We didn't previously look at a class:
`src/main/java/org/freo/purchase/Publish.java`

4. Take a look now. This basically connects to a RabbitMQ server and publishes to a queue named "purchase".

5. Look at the Purchase class
`src/main/java/org/freo/purchase/Purchase.java`

6. In this version there are two commented out lines:
```
        Publish publisher = new Publish();
```
and later
```
        publisher.publish(input);
```

Uncomment these.

7. Rebuild the project

   ```
   gradle build
   ```

8. RabbitMQ should be installed and running. Check with:
   ```
   service --status-all
   ```

9. As part of the build, the tests will have created Orders and these should already have published messages to Rabbit. Check and see. In a terminal window type:

   ```
   sudo rabbitmqctl list_queues
   ```

   You should see:

   ```
   Listing queues ...
   purchase    6
   ```

10. Let's create a simple Python client to subscribe to the queue, pick up the messages and parse the JSON.

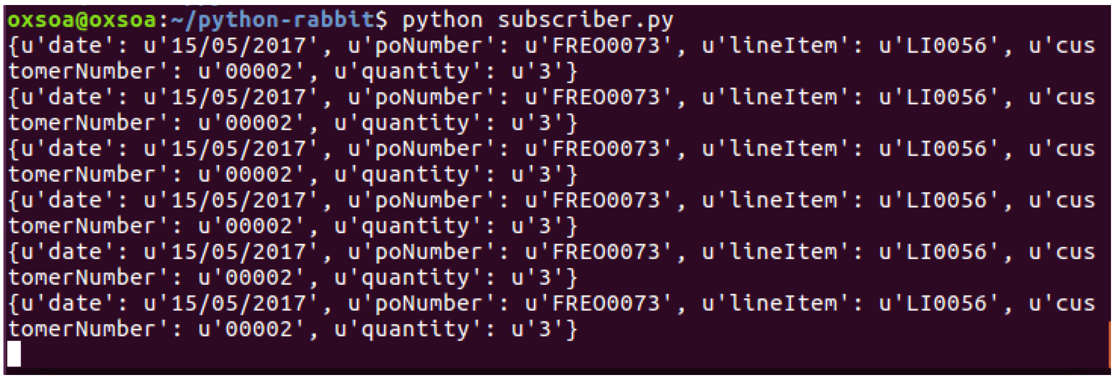
11. In ~/ex9, make a directory called **python-rabbit** and in there create a new file **subscriber.py**

12. Use the following Python code (here: https://freo.me/soa-rabbit-py)

    ```
    import pika
    import json

    connection =
    pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    channel = connection.channel()
    channel.queue_declare(queue='purchase')

    def callback(ch, method, properties, body):
        result = json.loads(body)
        print(result)

    channel.basic_consume(callback,
                          queue='purchase',
                          no_ack=True)

    channel.start_consuming()
    ```

13. Run the code:
    ```
    python subscriber.py
    ```

You should see:



```
oxsoa@oxsoa:~/python-rabbit$ python subscriber.py
{u'date': u'15/05/2017', u'poNumber': u'FRE00073', u'lineItem': u'LI0056', u'cus
tomerNumber': u'00002', u'quantity': u'3'}
{u'date': u'15/05/2017', u'poNumber': u'FRE00073', u'lineItem': u'LI0056', u'cus
tomerNumber': u'00002', u'quantity': u'3'}
{u'date': u'15/05/2017', u'poNumber': u'FRE00073', u'lineItem': u'LI0056', u'cus
tomerNumber': u'00002', u'quantity': u'3'}
{u'date': u'15/05/2017', u'poNumber': u'FRE00073', u'lineItem': u'LI0056', u'cus
tomerNumber': u'00002', u'quantity': u'3'}
{u'date': u'15/05/2017', u'poNumber': u'FRE00073', u'lineItem': u'LI0056', u'cus
tomerNumber': u'00002', u'quantity': u'3'}
{u'date': u'15/05/2017', u'poNumber': u'FRE00073', u'lineItem': u'LI0056', u'cus
tomerNumber': u'00002', u'quantity': u'3'}
```

14. Note that even though your Purchase service is not running, the messages remain queued up. In other words, as long as the Rabbit server is running, the two ends of the system don't need to both be up at the same time.

15. Run the service and use ARC to generate some new orders and see them also distributed to your Python system.

16. That's all