1) Реалізувати два варіанти черги: Producer / Consumer (Point-to-Point) і Publish / Subscribe (Topic)

- producer-consumer

```
rmq is up-to-date
Recreating producer1 ... done
Recreating consumer1 ... done
Attaching to producer1, consumer1
consumer1            | Connected to rmq
producer1            | Connected to rmq
consumer1            | Consumed msg0
consumer1            | Consumed msg1
consumer1            | Consumed msg2
consumer1            | Consumed msg3
consumer1            | Consumed msg4
consumer1            | Consumed msg5
consumer1            | Consumed msg6
producer1            | Published msg0
consumer1            | Consumed msg7
consumer1            | Consumed msg8
consumer1            | Consumed msg9
producer1            | Published msg1
producer1            | Published msg2
producer1            | Published msg3
producer1            | Published msg4
producer1            | Published msg5
producer1            | Published msg6
producer1            | Published msg7
producer1            | Published msg8
producer1            | Published msg9
producer1 exited with code 0
```

- publisher-subscriber

```
rmq is up-to-date
Starting subscriber11 ... done
Starting subscriber12 ... done
Recreating publisher1 ... done
Attaching to subscriber12, subscriber11, publisher1
publisher1           | Connected to rmq
subscriber11         | Connected to rmq
publisher1           | Published msg0
subscriber11         | Consuming from amq.gen-rObngcX3ct23Fxuob9PtPA
subscriber12         | Connected to rmq
publisher1           | Published msg1
subscriber11         | Consumed msg0
subscriber12         | Consuming from amq.gen-ieYPAD_qJwT9IfbKHAHJsw
publisher1           | Published msg2
publisher1           | Published msg3
publisher1           | Published msg4
publisher1           | Published msg5
publisher1           | Published msg6
publisher1           | Published msg7
publisher1           | Published msg8
publisher1           | Published msg9
subscriber11         | Consumed msg1
subscriber11         | Consumed msg2
subscriber12         | Consumed msg0
subscriber12         | Consumed msg1
subscriber12         | Consumed msg2
subscriber12         | Consumed msg3
subscriber12         | Consumed msg4
subscriber12         | Consumed msg5
subscriber12         | Consumed msg6
subscriber12         | Consumed msg7
subscriber12         | Consumed msg8
subscriber12         | Consumed msg9
subscriber11         | Consumed msg3
subscriber11         | Consumed msg4
subscriber11         | Consumed msg5
subscriber11         | Consumed msg6
subscriber11         | Consumed msg7
subscriber11         | Consumed msg8
subscriber11         | Consumed msg9
publisher1 exited with code 0
```

2) Для окремої черги реалізувати наступну логіку - клієнт відправляє повідомлення в чергу, один з консьюмерів його вичитує, модифікує і кладе у відповідну чергу клієнту, який виконував відправку, клієнт вичитує відповідь і відображає його

- перевірити випадок коли консьюмер не працює і що повідомлння будуть збережені у черзі, а після включення консьюмера - всі опрацьовані і відповіді доставлені клієнту

```
rmq is up-to-date
Creating worker2 ... done
Creating client2 ... done
Attaching to worker2, client2
client2             | Connected to rmq
worker2             | Connected to rmq
client2             | Published msg0
worker2             | Consumed msg0
client2             | Consumed msg0_processed
worker2             | Published msg0
client2             | Published msg1
worker2             | Consumed msg1
worker2             | Published msg1
client2             | Consumed msg1_processed
client2             | Published msg2
worker2             | Consumed msg2
worker2             | Published msg2
client2             | Consumed msg2_processed
client2             | Published msg3
worker2             | Consumed msg3
worker2             | Published msg3
client2             | Consumed msg3_processed
client2             | Published msg4
worker2             | Consumed msg4
worker2             | Published msg4
client2             | Consumed msg4_processed
client2             | Published msg5
worker2             | Consumed msg5
worker2             | Published msg5
client2             | Consumed msg5_processed
client2             | Published msg6
worker2             | Consumed msg6
worker2             | Published msg6
client2             | Consumed msg6_processed
client2             | Published msg7
worker2             | Consumed msg7
worker2             | Published msg7
client2             | Consumed msg7_processed
client2             | Published msg8
worker2             | Consumed msg8
worker2             | Published msg8
client2             | Consumed msg8_processed
client2             | Published msg9
worker2             | Consumed msg9
worker2             | Published msg9
client2             | Consumed msg9_processed
```

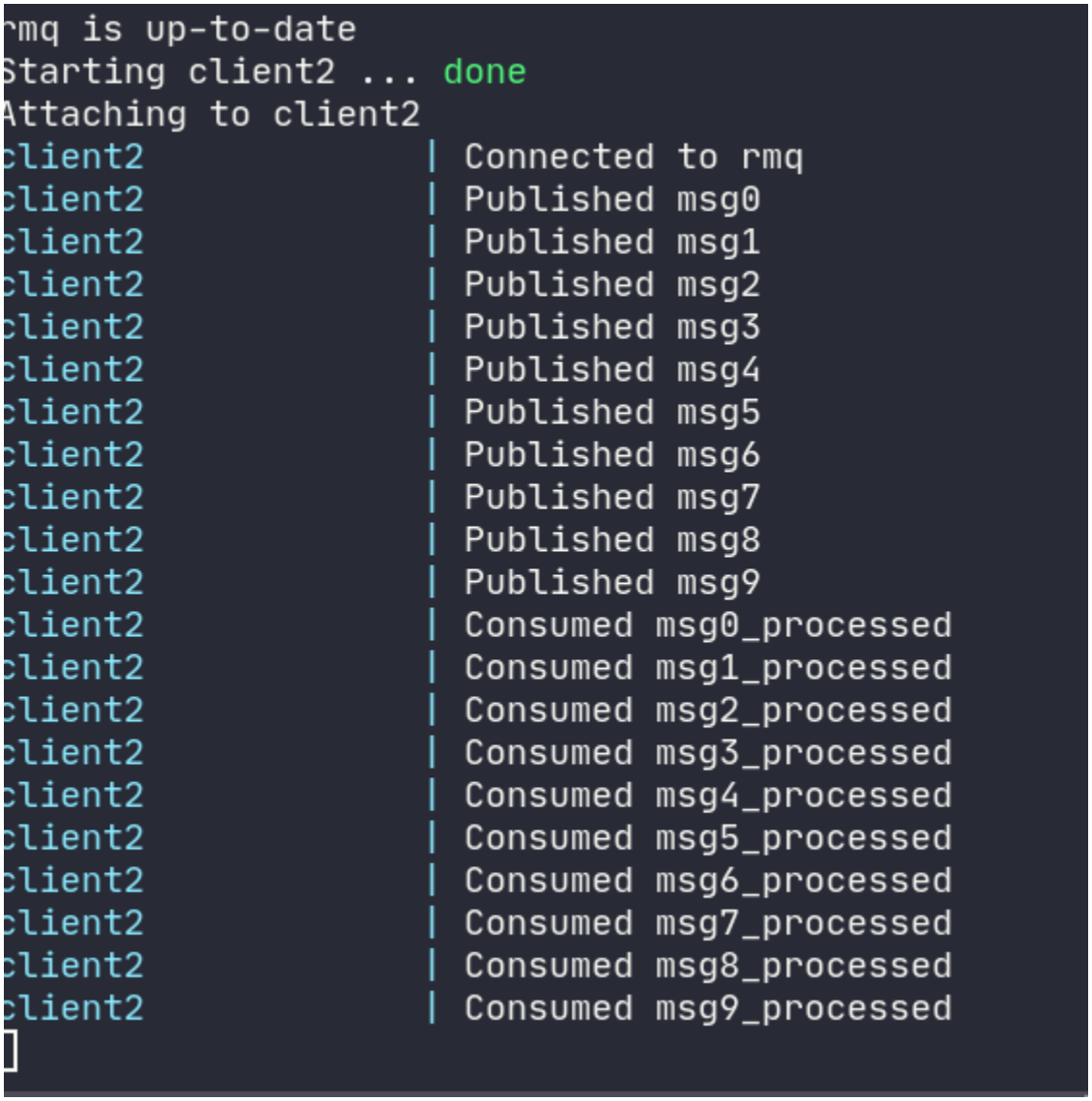
Starting client without worker:

```
rmq is up-to-date
Starting client2 ... done
Attaching to client2
client2              | Connected to rmq
client2              | Published msg0
client2              | Published msg1
client2              | Published msg2
client2              | Published msg3
client2              | Published msg4
client2              | Published msg5
client2              | Published msg6
client2              | Published msg7
client2              | Published msg8
client2              | Published msg9
```

Starting worker:

```
rmq is up-to-date
Starting worker2 ... done
Attaching to worker2
worker2              | Connected to rmq
worker2              | Consumed msg0
worker2              | Published msg0
worker2              | Consumed msg1
worker2              | Published msg1
worker2              | Consumed msg2
worker2              | Published msg2
worker2              | Consumed msg3
worker2              | Published msg3
worker2              | Consumed msg4
worker2              | Published msg4
worker2              | Consumed msg5
worker2              | Published msg5
worker2              | Consumed msg6
worker2              | Published msg6
worker2              | Consumed msg7
worker2              | Published msg7
worker2              | Consumed msg8
worker2              | Published msg8
worker2              | Consumed msg9
worker2              | Published msg9
```

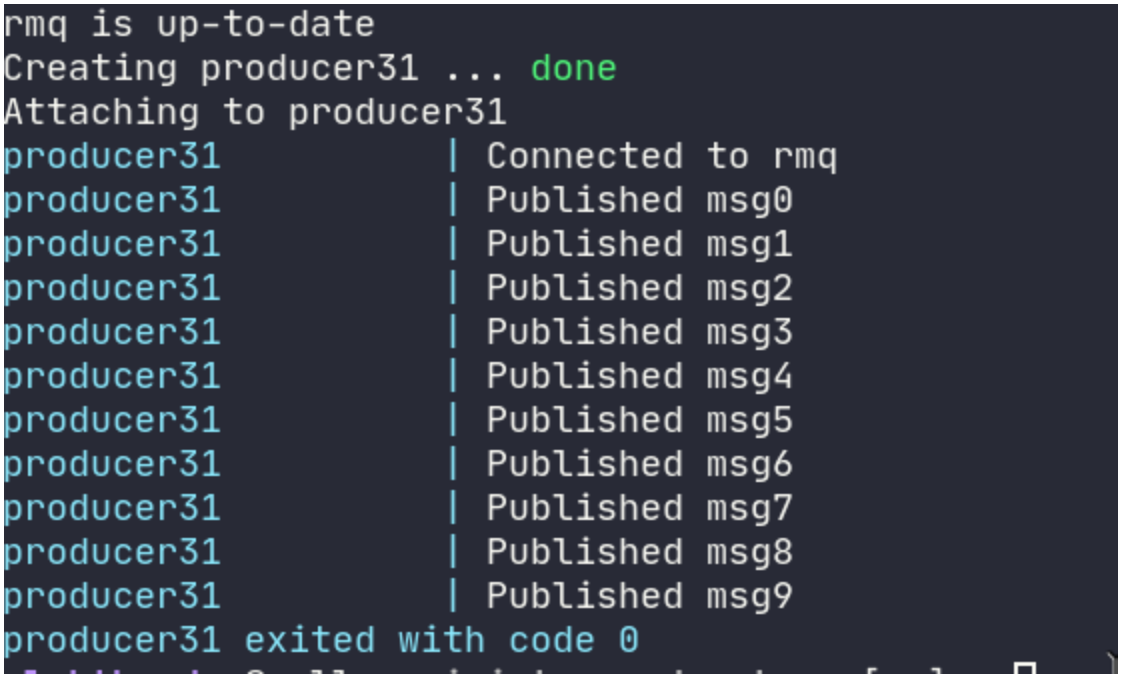
Client received processed messages

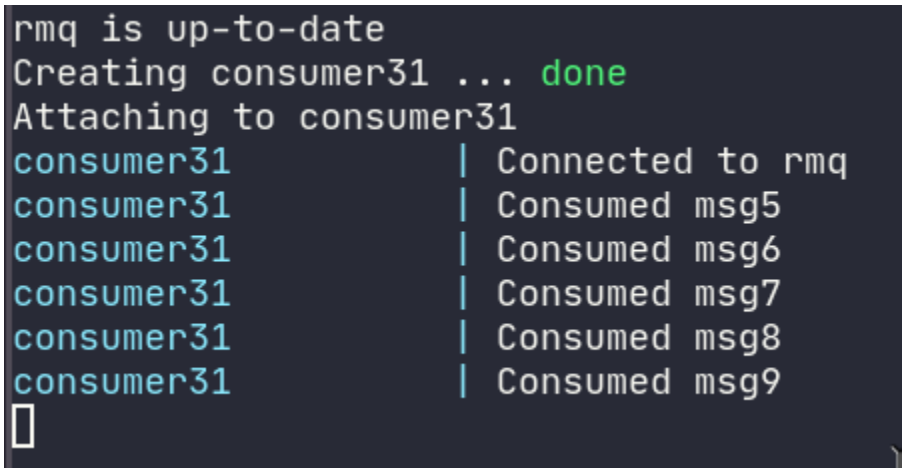


3) Налаштувати максимальна довжина черги *(Maxlength)* (

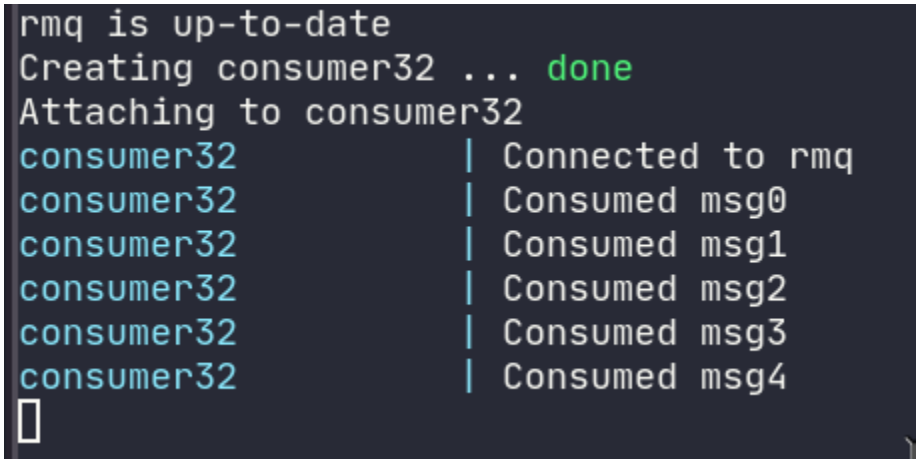
- подивитись що відбувається з новими повідомленнями коли черга заповнена
- мають бути два варіанти: затираються старі, не додаються нові

- Drop-head (default)





- reject-publish



4) Налаштувати збереження черги повідомлень (Message Persistence - *Durable* queue)

producing messages:

```
rmq is up-to-date
Creating producer4 ... done
Attaching to producer4
producer4           | Connected to rmq
producer4           | Published msg0
producer4           | Published msg1
producer4           | Published msg2
producer4           | Published msg3
producer4           | Published msg4
producer4           | Published msg5
producer4           | Published msg6
producer4           | Published msg7
producer4           | Published msg8
producer4           | Published msg9
producer4 exited with code 0
```

Stopping rabbitmq server:

```
olekthunder@mellon miniature-adventure [mq] ➜ docker-compose stop rmq
Stopping rmq ... done
```

Starting rabbitmq server and consumer:

```
Starting rmq ... done
Creating consumer4 ... done
Attaching to consumer4
consumer4           | Failed to connect, retrying...
consumer4           | Failed to connect, retrying...
consumer4           | Failed to connect, retrying...
consumer4           | Connected to rmq
consumer4           | Consumed msg0
consumer4           | Consumed msg1
consumer4           | Consumed msg2
consumer4           | Consumed msg3
consumer4           | Consumed msg4
consumer4           | Consumed msg5
consumer4           | Consumed msg6
consumer4           | Consumed msg7
consumer4           | Consumed msg8
consumer4           | Consumed msg9
```

5) Налаштувати максимальний час перебування повідомлення в черзі *(MessageTTL)*

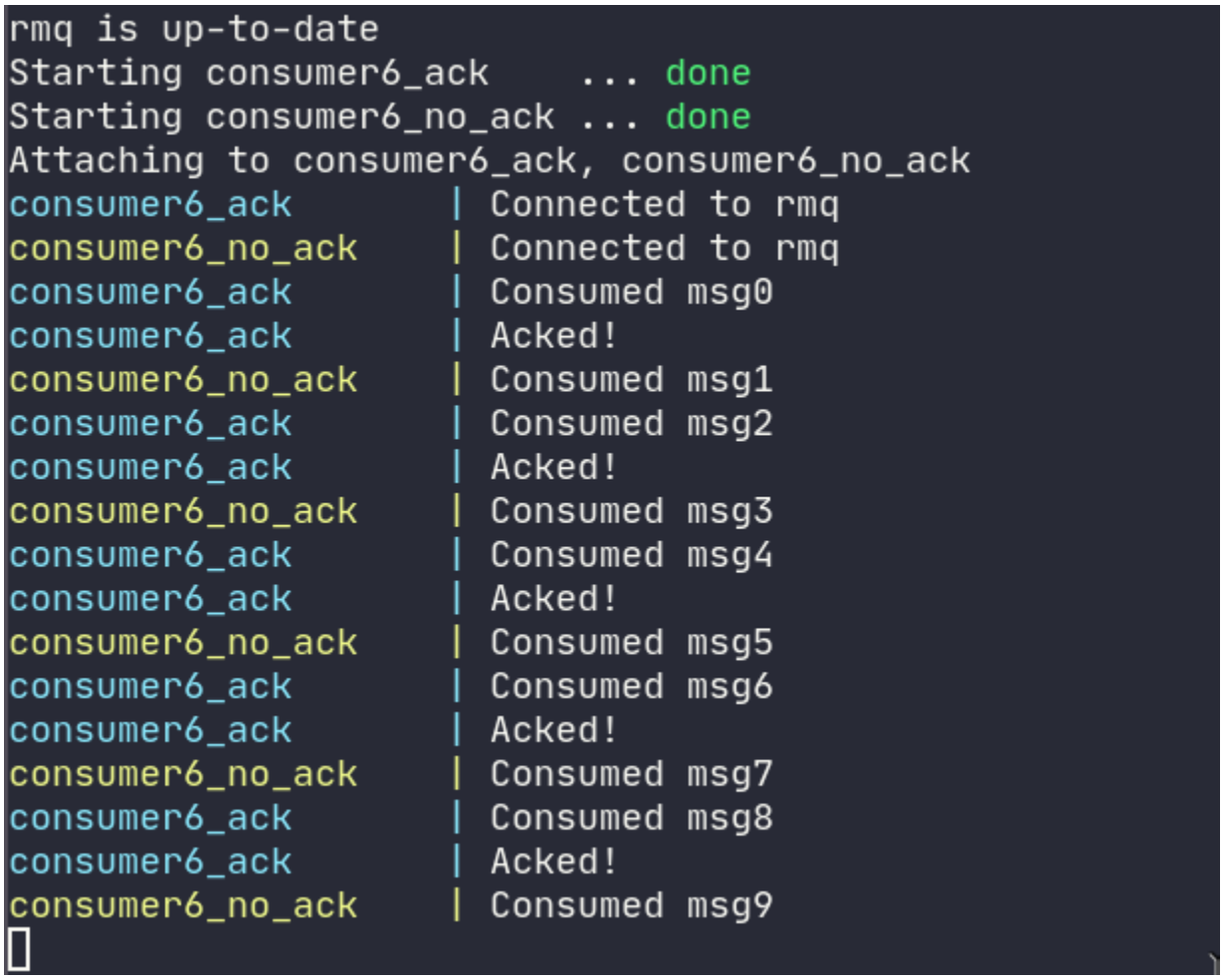
TTL is 5 seconds, consumer sleeps for 10 seconds, producer produces 10 messages with 1 second delay

```
rmq is up-to-date
Creating producer5 ... done
Creating consumer5 ... done
Attaching to consumer5, producer5
consumer5           | Connected to rmq
producer5           | Connected to rmq
producer5           | Published msg0
producer5           | Sleeping for 1 second...
consumer5           | Sleeping for 10 seconds...
producer5           | Published msg1
producer5           | Sleeping for 1 second...
producer5           | Published msg2
producer5           | Sleeping for 1 second...
producer5           | Published msg3
producer5           | Sleeping for 1 second...
producer5           | Published msg4
producer5           | Sleeping for 1 second...
producer5           | Published msg5
producer5           | Sleeping for 1 second...
producer5           | Published msg6
producer5           | Sleeping for 1 second...
producer5           | Published msg7
producer5           | Sleeping for 1 second...
producer5           | Published msg8
producer5           | Sleeping for 1 second...
producer5           | Published msg9
producer5           | Sleeping for 1 second...
consumer5           | Consumed msg5
consumer5           | Consumed msg6
consumer5           | Consumed msg7
consumer5           | Consumed msg8
consumer5           | Consumed msg9
producer5 exited with code 0
```

6) Для окремої черги Producer / Consumer налаштувати *Message* Acknowledgment, який забезпечує гарантовану доставку повідомлень. Показати випадок, коли Consumer бере з черги повідомлення на обробку, але не може його обробити і падає (тобто не повертає Ack чи повертає негативний Ack). Показати, чи буде при цьому дане необроблене повідомлення взято на обробку іншим Consumer або виявиться втраченим.

- Для цього мають бути відпралені в чергу 10 повідолень - msg1 - msg10
- Паралельно запускаються Consumer 1 та Consumer 2
- Consumer 1 - відправляє Ack, Consumer 2 - не повертає Ack
- Consumer 1 та Consumer 2 мають вичитати по 5 повідомлень кожен
- Consumer 2 (який не повертав Ack) відключаєте
- Після того всі повідомлення які були доставлнні на Consumer 2 має бути автоматично вичитані Consumer 1

Starting two consumers, waiting for all messages to be consumed:



After consumer without ack killed:

```
rmq is up-to-date
Starting consumer6_ack    ... done
Starting consumer6_no_ack ... done
Attaching to consumer6_ack, consumer6_no_ack
consumer6_ack       | Connected to rmq
consumer6_no_ack    | Connected to rmq
consumer6_ack       | Consumed msg0
consumer6_ack       | Acked!
consumer6_no_ack    | Consumed msg1
consumer6_ack       | Consumed msg2
consumer6_ack       | Acked!
consumer6_no_ack    | Consumed msg3
consumer6_ack       | Consumed msg4
consumer6_ack       | Acked!
consumer6_no_ack    | Consumed msg5
consumer6_ack       | Consumed msg6
consumer6_ack       | Acked!
consumer6_no_ack    | Consumed msg7
consumer6_ack       | Consumed msg8
consumer6_ack       | Acked!
consumer6_no_ack    | Consumed msg9
consumer6_ack       | Consumed msg1
consumer6_ack       | Acked!
consumer6_ack       | Consumed msg3
consumer6_ack       | Acked!
consumer6_ack       | Consumed msg5
consumer6_ack       | Acked!
consumer6_ack       | Consumed msg7
consumer6_ack       | Acked!
consumer6_ack       | Consumed msg9
consumer6_ack       | Acked!
consumer6_no_ack exited with code 2
```