LAB – Message Queue Systems

Hands on experience with RabbitMQ

References:

- RabbitMQ documentation https://www.rabbitmq.com/tutorials/

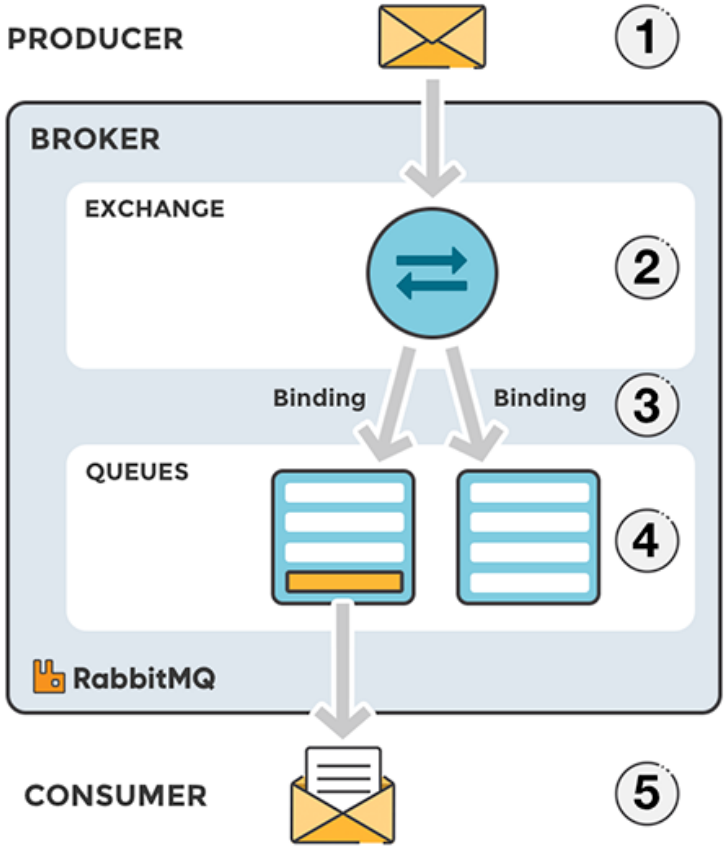
RabbitMQ

- RabbitMQ is an open-source message-broker software that implements the Advanced Message queuing Protocol (AMQP)
- Different libraries available for different languages implement the functionalities required to interface with RabbitMQ, among them we have **PIKA**, a python library
- The basic architecture includes a Producer application and a Consumer application, the producer enqueues messages, the consumer dequeues them
- Broker functionalities implemented by RabbitMQ take care of routing the messages between them

RabbitMQ Message Flow

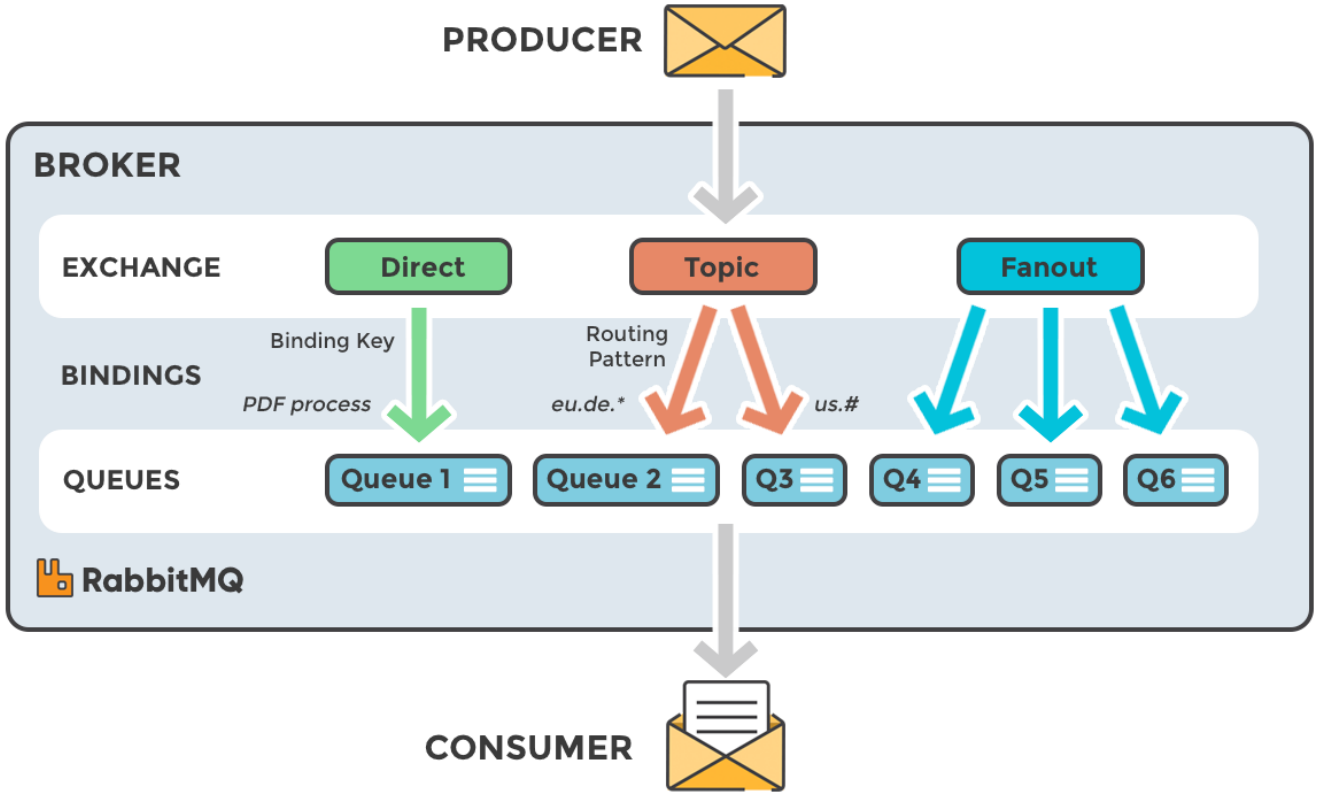
- RabbitMQ internally is composed of a set of queues (one for each consumer) and a set of exchanges that take care of routing the messages. Routing occurs according to a set of rules (binding)

1. The producer publishes a message to an *exchange*. When creating an exchange, the type must be specified. This topic will be covered later
2. The exchange receives the message and is now responsible for routing the message. The exchange takes different message attributes into account, such as the *routing key*, depending on the exchange type.
3. *Bindings* must be created from the exchange to queues. In this case, there are two bindings to two different queues from the exchange. The exchange routes the message into the queues depending on message attributes.
4. The messages stay in the queue until they are handled by a consumer
5. The consumer handles the message.



Exchange types

- Each queue has associated a **routing key**
- Three different exchange types are available:
  - **Direct**: The message is routed to the queues whose binding key exactly matches the routing key of the message.
  - **Fanout**: A fanout exchange routes messages to all the queues bound to it.
  - **Topic**: The topic exchange performs a wildcard match between the routing key and the routing pattern specified in the binding.



Deploy RabbitMQ in a container

```
docker run -d --hostname my-rabbit --name some-rabbit
rabbitmq:3
```

```
root@HAJJVX8OPD7M5QO:~# docker run -d --hostname my-rabbit --name some-rabbit rabbitmq:3
Unable to find image 'rabbitmq:3' locally
3: Pulling from library/rabbitmq
5c939e3a4d10: Already exists
c63719cdbe7a: Already exists
19a861ea6baf: Already exists
651c9d2d6c4f: Already exists
da31881b2e3b: Pull complete
df67acc10503: Pull complete
31f8b0bc70f4: Pull complete
a1cd9cbfba9d: Pull complete
9bae18855d32: Pull complete
e094f487f477: Pull complete
Digest: sha256:8d8caded7222302a3d5cdcd5d1d37680a46b6a26bf
Status: Downloaded newer image for rabbitmq:3
368a5202e52e680c38611bcc0125e83689357dcdffd2579e630f522cd
```

```
root@HAJJVX8OPD7M5QO:~# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
368a5202e52e        rabbitmq:3          "docker-entrypoint.s…"   7 seconds ago       Up 5 seconds        4369/tcp, 5671-5672/tcp,
bit
```

```
docker network inspect bridge
```

```
"Containers": {
    "0f0219d595593e8bd2fa42b785cbc8bb8b172d4faa98c581a6cdd8787e21577e": {
        "Name": "intelligent_lalande",
        "EndpointID": "5449aa1eb5509dead0be0aa9f7bf0ee356f787a7b73a6aa082d1de70be9c8787",
        "MacAddress": "02:42:ac:11:00:03",
        "IPv4Address": "172.17.0.3/16",
        "IPv6Address": ""
    },
    "368a5202e52e680c38611bcc0125e83689357dcdffd2579e630f522cd18dace8": {
        "Name": "some-rabbit",
        "EndpointID": "f127237732cd897d2367cc77ca85012fc97d68e69b8afaa8076356f7eda74289",
        "MacAddress": "02:42:ac:11:00:05",
        "IPv4Address": "172.17.0.5/16",
        "IPv6Address": ""
    },
    "acba68e0d30dc74a5af0562625eb8346e49410ce004debc647960366956cbba3": {
        "Name": "bold_roentgen",
        "EndpointID": "86006663e0d6e0be92f82e3914825f4d7fa81fce1233f43a799cbfc36d15aaf6",
        "MacAddress": "02:42:ac:11:00:02",
        "IPv4Address": "172.17.0.2/16",
        "IPv6Address": ""
    },
    "b637d91f2d185aae897aa4f97a6c77382206719d30c542e5d70dc8f41957a7eb": {
        "Name": "mysql",
        "EndpointID": "5549b9e06189f3d7ec22b6095afc8d7c1603ec3be54dc02d0b3b463fe0e0b90d",
        "MacAddress": "02:42:ac:11:00:04",
        "IPv4Address": "172.17.0.4/16",
        "IPv6Address": ""
    }
```

Default username and password: guest/guest

Create a RabbitMQ producer

- Dockerfile

```
FROM python:3-alpine
RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app
# Install the PIKA library
RUN pip3 install pika
# This variable forces pika to print something out
ENV PYTHONUNBUFFERED=1
COPY producer.py /usr/src/app
ENTRYPOINT ["python3"]
CMD ["producer.py"]
```

Producer code

```
import pika

# Connect to RabbitMQ
connection =
pika.BlockingConnection(pika.ConnectionParameters('172.17.0.5'))
channel = connection.channel()

# Create a queue
channel.queue_declare(queue='hello’)

# Send the message
channel.basic_publish(exchange='', routing_key='hello', body='Hello World!’)

# Close the channel
connection.close()
```

Default exchange type is used (direct)

Default exchange will be used

Consumer code

```
# Connect to RabbitMQ
connection = pika.BlockingConnection(pika.ConnectionParameters(host='172.17.0.5'))
channel = connection.channel()

# Connect to a queue
channel.queue_declare(queue='hello’)

# Define a callback invoked every time a message is received
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)

# Subscribe to the queue and assign the callback
channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)

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

Build and run

- Build

```
docker build -t rabbitmq-consumer .
docker build -t rabbitmq-producer .
```

- Run

```
docker run -d --name consumer rabbitmq-consumer
docker run -d --name producer rabbitmq-producer
```

```
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker build -t rabbitmq-producer .
Sending build context to Docker daemon  3.072kB
Step 1/8 : FROM python:3-alpine
 ---> a0ee0c90a0db
Step 2/8 : RUN mkdir -p /usr/src/app
 ---> Using cache
 ---> 7352b763891b
Step 3/8 : WORKDIR /usr/src/app
 ---> Using cache
 ---> 704a7edded43
Step 4/8 : RUN pip3 install pika
 ---> Running in 203fa43d3f38
Collecting pika
  Downloading pika-1.1.0-py2.py3-none-any.whl (148 kB)
Installing collected packages: pika
Successfully installed pika-1.1.0
Removing intermediate container 203fa43d3f38
 ---> 7496bf981332
Step 5/8 : ENV PYTHONUNBUFFERED=1
 ---> Running in 91de1633ce1d
Removing intermediate container 91de1633ce1d
 ---> b950e5a0bd81
Step 6/8 : COPY producer.py /usr/src/app
 ---> 4c08da705911
Step 7/8 : ENTRYPOINT ["python3"]
 ---> Running in 0e4df6a27a3b
Removing intermediate container 0e4df6a27a3b
 ---> 00c772ad0da9
Step 8/8 : CMD ["producer.py"]
 ---> Running in 5e089df8af6f
Removing intermediate container 5e089df8af6f
 ---> 2d840450091f
Successfully built 2d840450091f
Successfully tagged rabbitmq-producer:latest
```

Check Log

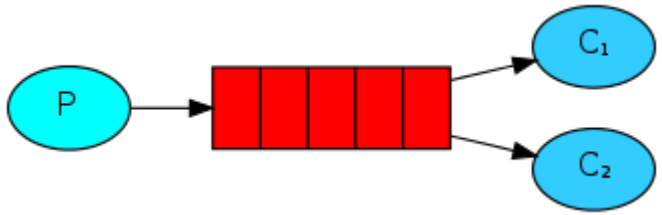
```
docker logs consumer
docker logs producer
docker logs rabbitqm
```

```
root@HAJJVX8OPD7M5QO:~/rabbitmq-consumer# docker logs consumer
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
root@HAJJVX8OPD7M5QO:~/rabbitmq-consumer# docker logs producer
 [x] Sent 'Hello World!'
```

```
2020-02-19 17:47:07.657 [info] <0.597.0> started TCP listener on [::]:5672
2020-02-19 17:47:07.657 [info] <0.267.0> Running boot step cluster_name defined by app rabbit
2020-02-19 17:47:07.657 [info] <0.267.0> Running boot step direct_client defined by app rabbit
2020-02-19 17:47:07.792 [info] <0.8.0> Server startup complete; 0 plugins started.
 completed with 0 plugins.
2020-02-20 12:18:58.209 [info] <0.2650.2> accepting AMQP connection <0.2650.2> (172.17.0.6:34150 -> 172.17.0.5:5672)
2020-02-20 12:18:58.227 [info] <0.2650.2> connection <0.2650.2> (172.17.0.6:34150 -> 172.17.0.5:5672): user 'guest' authenticated and granted access t
o vhost '/'
2020-02-20 12:18:58.232 [info] <0.2650.2> closing AMQP connection <0.2650.2> (172.17.0.6:34150 -> 172.17.0.5:5672, vhost: '/', user: 'guest')
2020-02-20 12:19:56.159 [info] <0.2728.2> accepting AMQP connection <0.2728.2> (172.17.0.6:34154 -> 172.17.0.5:5672)
2020-02-20 12:19:56.161 [info] <0.2728.2> connection <0.2728.2> (172.17.0.6:34154 -> 172.17.0.5:5672): user 'guest' authenticated and granted access t
o vhost '/'
2020-02-20 12:20:30.676 [info] <0.2780.2> accepting AMQP connection <0.2780.2> (172.17.0.7:49086 -> 172.17.0.5:5672)
2020-02-20 12:20:30.678 [info] <0.2780.2> connection <0.2780.2> (172.17.0.7:49086 -> 172.17.0.5:5672): user 'guest' authenticated and granted access t
o vhost '/'
2020-02-20 12:20:30.682 [info] <0.2780.2> closing AMQP connection <0.2780.2> (172.17.0.7:49086 -> 172.17.0.5:5672, vhost: '/', user: 'guest')
```

Work Queues

- Work Queues (or Task Queues) can be created to dispatch *work-intensive tasks among different workers*
- One or more tasks producers encapsulate the task into a message and send the message to the message queue system
- A worker (from a pool of workers) eventually becomes available and receives and processes the task



Deploy and test more consumers

- <u>Deploy more than one consumer</u>
- Restart several times the producer to produce more messages

```
docker restart producer
```

- Check that some messages are dispatched to one consumer, others to another
- The consumer can add the following to avoid to receive a message when it is busy serving another one

```
channel.basic_qos(prefetch_count=1)
```

```
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker logs consumer
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker logs consumer2
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
```

Message routing

- Complex message routing can be performed by creating multiple exchanges and bindings
- A binding is a relationship between an exchange and a queue

```
channel.queue_bind(exchange=exchange_name, queue=queue_name)
```

- This can be simply read as: the queue is interested in messages from this exchange
- The message routing behavior for a binding depends on the type of exchange
- For instance, direct exchange is simple - a message goes to the queues whose binding key (the name of the queue) exactly matches the routing key of the message

Exchange example - Producer

```
# Message producer
…
# There is no need for queue declaration (I use the exchange)
channel.exchange_declare(exchange='exchange_name',
exchange_type='direct')


channel.basic_publish(exchange='exchange_name',
routing_key='name1', body='hello1')
channel.basic_publish(exchange='exchange_name',
routing_key='name2', body='hello2')


connection.close()
```

Exchange example - Consumer

```
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker logs producer
 [x] Sent 'Hello World!'
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker logs consumer
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello1'
 [x] Received b'Hello2'
```

```
# Message consumer
…
channel.exchange_declare(exchange='exchange_name', exchange_type='direct')

# I let the system to create the queue name
result = channel.queue_declare(queue='', exclusive=True)
queue_name = result.method.queue

# Bind the queue to one or more keys/exchanges (it can be done at runtime)
channel.queue_bind(exchange='exchange_name', queue=queue_name, routing_key='name1')
channel.queue_bind(exchange='exchange_name', queue=queue_name, routing_key='name2')

channel.basic_consume(
    queue=queue_name, on_message_callback=callback, auto_ack=True)
channel.start_consuming()
```

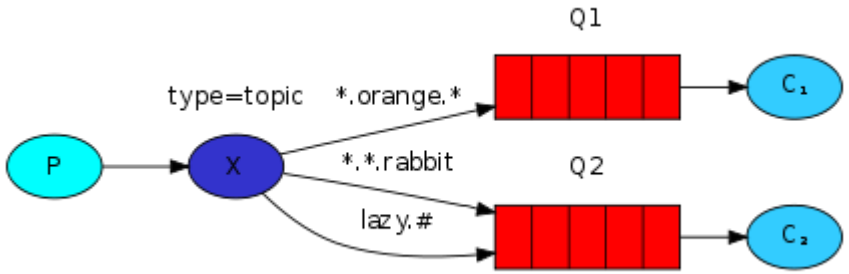
Message broadcast

- In the previous example messages were dispatched to the different consumers registered to the same queue in a round robin fashion (***direct*** message exchange type)
- If we want the messages to be delivered to all the subscribers, we need to change the exchange type to ***fanout***

channel.exchange_declare(exchange='logs', exchange_type='fanout')

Message Exchange by Topic

- The topic exchange type allows to route messages based not on a routing key but on a list of words (topics) delimited by dots
- Topic definitions can exploit wildcards, like * to substitute for exactly one word or # to substitute for zero or more words, through them bindings key that receive messages from different producers can be created
- In the example, if we consider a routing key with the following pattern, <celerity>.<colour>.<species>, Q1 will receive all the messages regarding oranges animals, while Q2 all messages about rabbits or lazy animals



Producer - Topic

```
channel.exchange_declare(exchange=‘topics', exchange_type='topic’)

routing_key1 = ‘lazy.green.rabbit’
routing_key2 = ‘fast.yellow.zebra’

message = 'Hello World!'
channel.basic_publish(exchange=‘topics', routing_key=routing_key1,
body=message)
channel.basic_publish(exchange=‘topics', routing_key=routing_key2,
body=message)

connection.close()
```

Consumer - Topic

```
channel.exchange_declare(exchange='topics', exchange_type='topic')

result = channel.queue_declare('', exclusive=True)
queue_name = result.method.queue

channel.queue_bind(exchange='topics', queue=queue_name,
routing_key='*.*.rabbit')
channel.queue_bind(exchange='topics', queue=queue_name,
routing_key='lazy.#')
```

Only the first message is received!

```
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker run -d --name producer rabbitmq-producer
f3e03502c86f2fd673c41f2b7ef5823b9a5b6c065578c89f85009f6dd72119f6
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker logs producer
 [x] Sent 2 messages!'
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker logs consumer
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello1'
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer#
```

Check RabbitMQ status

- Retrieve the list of queues

  `rabbitmqctl list_queues`

- Get message stats

  `rabbitmqctl list_queues name messages_ready messages_unacknowledged`

- Run the command on the container

  `docker exec CONTAINER_ID COMMAND`

```
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker exec 368a5202e52e rabbitmqctl list_queues
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name    messages
amq.gen--owsdjyAEMk7Yd-QQhxs-g  0
hello   0
root@HAJJVX8OPD7M5QO:~/rabbitmq-producer# docker exec 368a5202e52e rabbitmqctl list_queues name messages_ready messages_unacknowledged
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name    messages_ready  messages_unacknowledged
amq.gen--owsdjyAEMk7Yd-QQhxs-g  0       0
hello   0       0
```