Universidad EAFIT
Ingeniería de Sistemas
ST0263 Tópicos Especiales en Telemática, 2021-2

Laboratorio #4

Título: Middleware Orientados a Mensajes (MOM)

Objetivo: Desplegar un middleware orientado a mensajes (MOM) con el fin de afianzar los conocimientos desarrollados en la sesión de clases.

Fecha de entrega: 10 de septiembre 2021

1. Introducción

Un middleware es considerado un elemento fundamental para el desarrollo y despliegue de los sistemas distribuidos. Básicamente se constituye en un intermediario en aplicaciones/objetos/componentes/servicios distribuidos y los cuales requieren comunicarse entre si. Considerando que la comunicación entre objetos/aplicaciones distribuidos puede ser sincrónica o asincrónica, un MOM permite una forma de comunicación asincrónica entre éstos aportando también para lograr un fuerte desacoplamiento entre los mismos.

Refs:

- Middleware - Wikipedia
- Message-oriented middleware - Wikipedia
- RabbitMQ - Wikipedia
- Messaging that just works — RabbitMQ

2. Descripción del MOM – RabbitMQ

RabbitMQ es middleware de código abierto que va a funcionar como un intermediario entre aplicaciones que pueden ser independientes entre si. De esta forma, RabbitMQ se constituye en una capa software que le permitirá la comunicación entre ellas. Una de las grandes ventajas que da el considerar este tipo de soluciones, es que la arquitectura final del sistema se convierte en una solución débilmente acoplada.

3. Instalación y ejecución del servidor MOM.

Algunos ejemplos de rabbitmq en: https://github.com/st0263eafit/st026320212.git

Para efectos de este laboratorio, vamos a instalar un servidor RabbitMQ utilizando contenedores. Particularmente Docker en Amazon.

Pasos.

1) Despliegue una instancia EC2. Cuando este creando el grupo de seguridad, permita el ingreso del tráfico al puerto 5672 y 15672 TCP.

2) Instale el software docker en la máquina para poder ejecutar una imagen del contenedor de RabbitMQ.

$ sudo yum install docker

3) Habilite el servicio de docker en la máquina a través del comando:

$ sudo systemctl start docker

4) Inicie el servicio de docker en la máquina a través del comando:

$ sudo systemctl start docker

5) Agregue el usuario al grupo de docker

$ sudo usermod -aG docker ec2-user

6) Ponga en ejecución el docker del servidor RabbitMQ con el siguiente comando:

```
$ sudo docker run -d --hostname my-rabbit -p 15672:15672 -p 5672:5672 --name rabbit-server -e RABBITMQ_DEFAULT_USER=user -e RABBITMQ_DEFAULT_PASS=password rabbitmq:3-management
```

7) Verifique que la imagen esta corriendo:

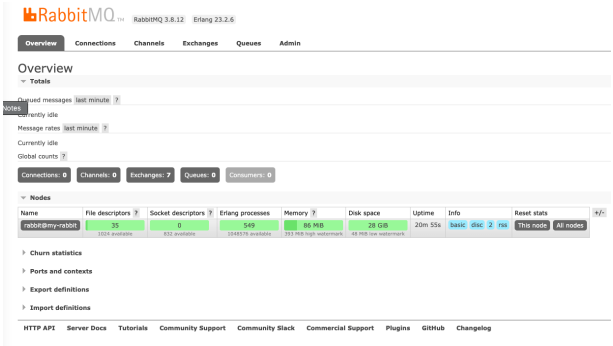
$sudo docker ps

Al ejecutar el comando, debe ver algo parecido a esto:

```
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                                                  NAMES
d004cd3b3c4d        rabbitmq:3-management   "docker-entrypoint.s…"   9 seconds ago       Up 7 seconds        4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671
/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp   rabbit-server
```

8) En un browser, digite la dirección IP pública (elástica) de su instancia en AWS en el puerto 15672 con el fin de poder ingresar a la consola de administración del servidor RabbitMQ. http://<dirIP:15672>.

Utilice el username: user y password: password.

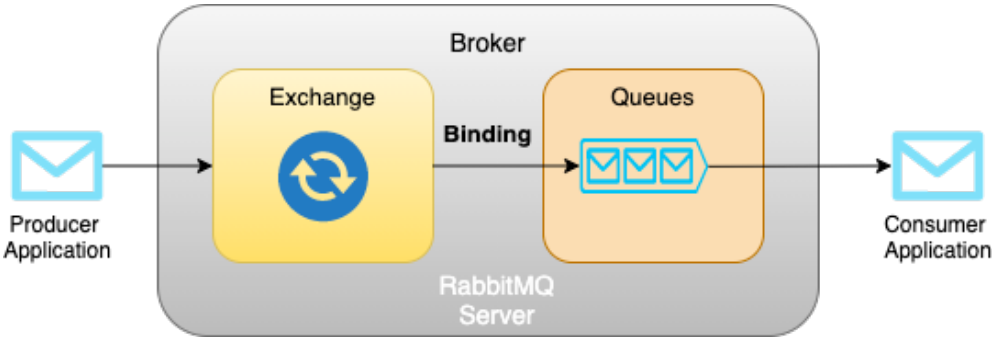


4. Prueba del MOM

En este punto se procederá a configurar los parámetros necesarios para intercambiar los mensajes entre dos aplicaciones: productor y consumidor. De esta forma se configura en el servidor RabbitMQ:

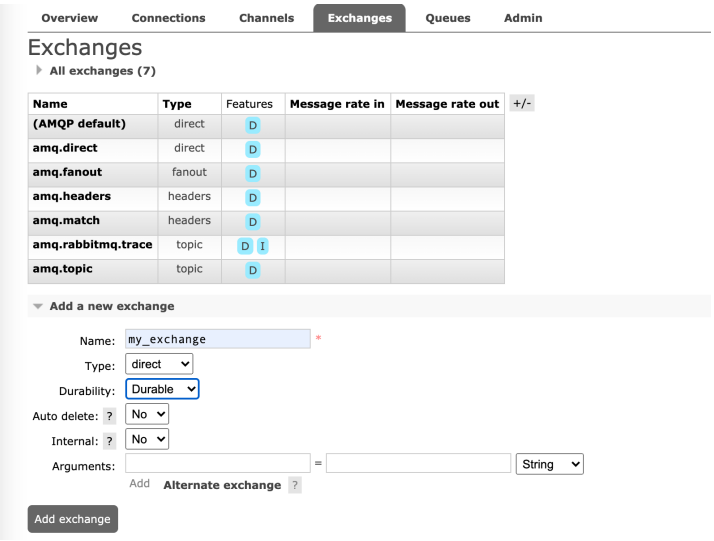
1. **Exchange**: Es el sitio donde la aplicación enviará/publicaran los mensajes. Básicamente son los responsables por el enrutamiento de los mensajes a la cola seleccionada. Para esto utiliza algunos atributos con el fin de ubicar el mensaje en la cola adecuada.
2. **Cola**: Es la estructura donde se colocaran los mensajes para ser retirados por la aplicación consumidor.

En una vista de alto nivel, esta es la arquitectura en términos generales a desplegar:

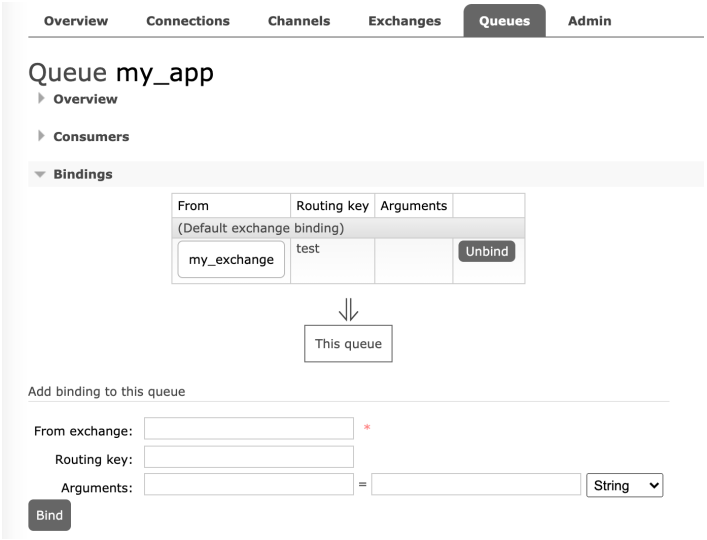


Ahora, realizaremos el conjunto de pasos para crear el exchange y la cola.

3. **Crear el exchange:** En el menú, seleccione la opción de Exchanges. Para efectos de este laboratorio vamos a crear un exchange que se denomine “my_exchange” del tipo “direct”. Deje el resto de los parámetros por defecto y de click en “add exchange”.



4. **Crear la cola.** En el menú, seleccione la opción de “Queues”. Para efectos de este laboratorio, crearemos una cola denominada “my_app”.
5. **Asociar el exchange y la cola.** Para esto, da click en la cola que creamos “my_app”. En la sección “Bindings”, digite el exchange que vamos a asociar, en este caso “my_exchange”. Igualmente, digite el routing key, “test” y de click en “Bind”.



En este punto ya el servidor RabbitMQ esta listo para recibir mensajes de una aplicación para colocarlos en la cola y que sean tomado por una aplicación.

6. Aplicación Productor.

Con el fin de poner en marcha las funcionalidades del servidor MOM RabbitMQ desplegado, se va a ejecutar una primera y sencilla aplicación cliente la cual envía un mensaje al servidor. Para poder ejecutar la aplicación, requiere el paquete pika, el cual debe ser instalado para la versión de Python que este ejecutando.

A continuación puede observar el código fuente ejemplo para la aplicación Publisher. Tenga en cuenta que dónde aparece la dirección IP coloque la dirección IP del servidor RabbitMQ.

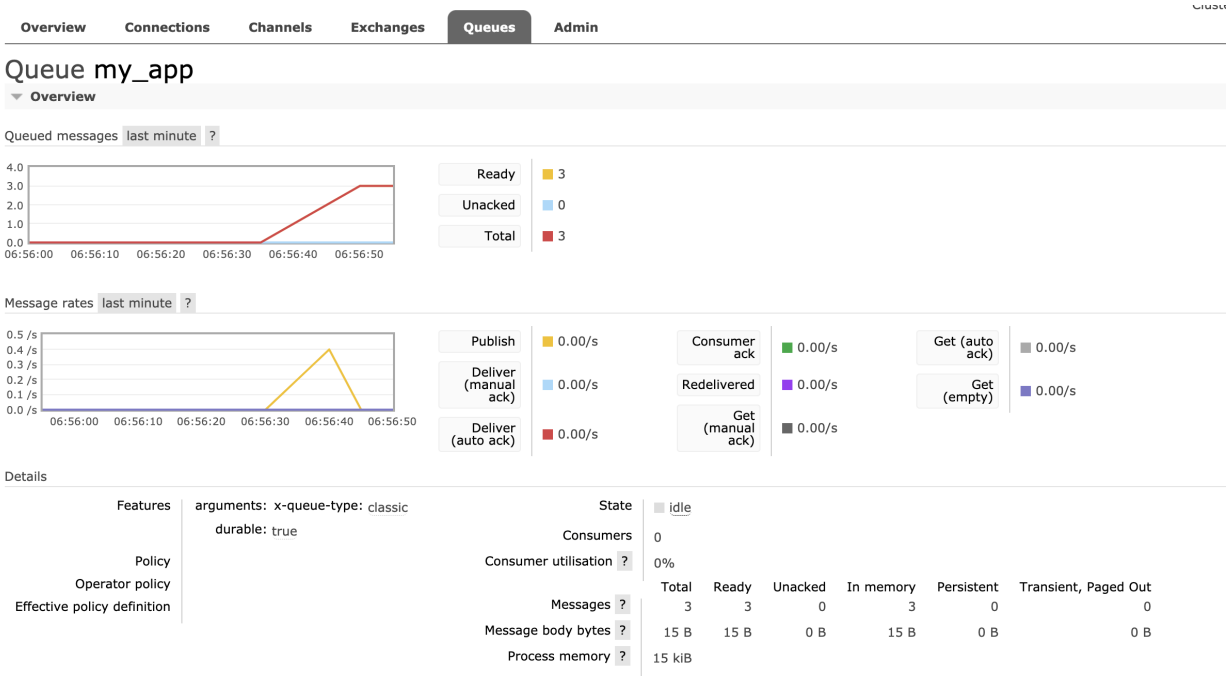
```
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('dirIP', 5672, '/',
pika.PlainCredentials('user', 'password')))
channel = connection.channel()

channel.basic_publish(exchange='my_exchange', routing_key='test', body='Test!')
print("Runnning Producer Application...")
print(" [x] Sent 'Hello World...!'")

connection.close()
```

7. Verificación de Mensajes en el Servidor MOM

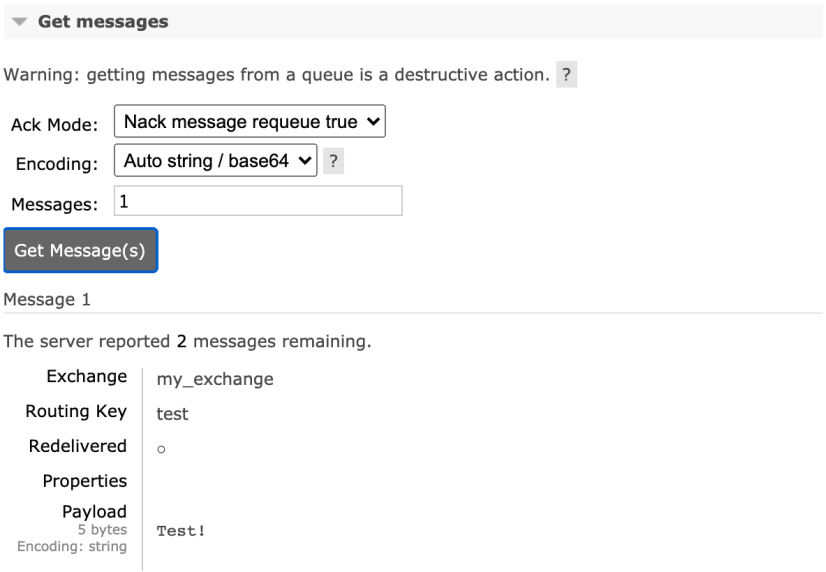
En este punto, vamos a verificar el funcionamiento de todo el ecosistema desarrollado y desplegado. Para esto, ejecute la aplicación productor y envíe tres mensajes al servidor RabbitMQ. Una vez ejecute la aplicación debe observar en consola que el mensaje ha sido enviado.

```
Runnning Producer Application...
 [x] Sent 'Hello World...!'
```

Ahora verifiquemos en el servidor que los mensajes si estén en la cola destinada para estos efectos. En el menú de Queues, seleccione la cola “my_app”. Ahí podrá observar un reporte parecido al que se observa en la figura siguiente. Como puede notar, debe haber tres mensajes en la cola, producto de las tres veces que ejecuto la aplicación.



En esa misma vista de administración, en la sección de “Get Message(s)” puede dar click y observar detalles del mensaje.



8. Aplicación Consumidor

En esta sección, puede observar el código fuente para la aplicación consumidor.

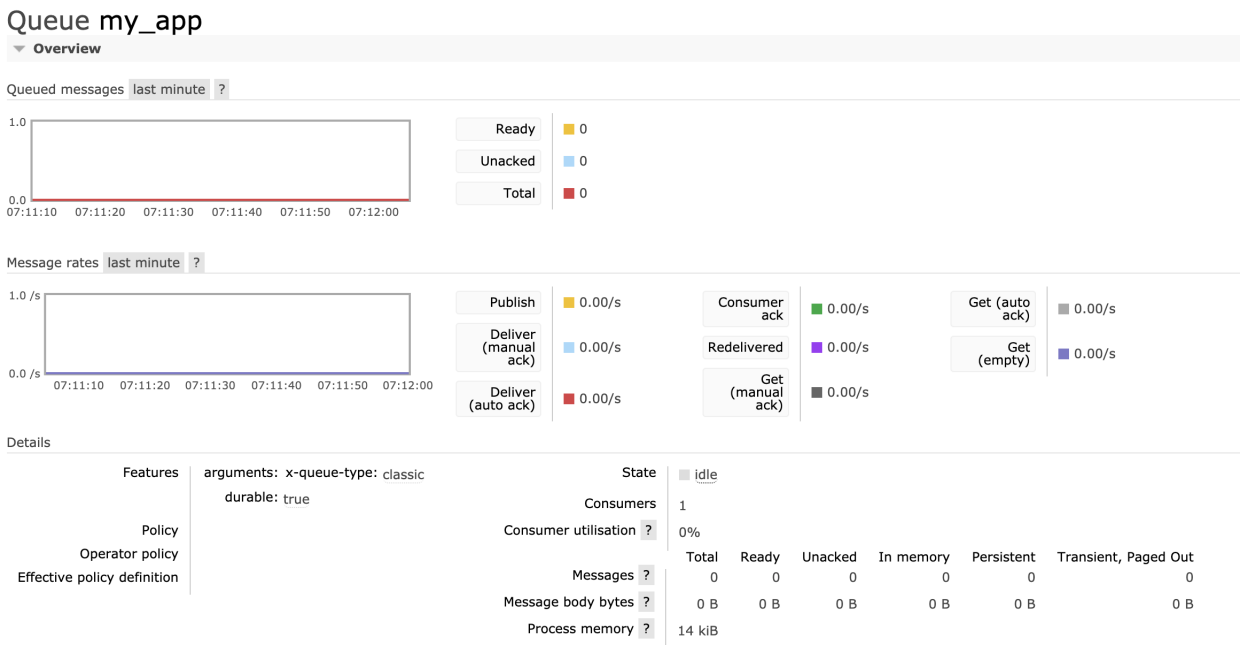
```
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('dirIP', 5672, '/',
pika.PlainCredentials("user", "password")))
```

```
channel = connection.channel()

def callback(ch, method, properties, body):
    print(f'{body} is received')

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

Una vez ejecute el código de la aplicación consumidor, podrá ver lo siguiente en la consola:

```
b'Test!' is received
b'Test!' is received
b'Test!' is received
```

Esta salida nos indica que los tres mensajes fueron retirados de la cola. Ahora verifique nuevamente el estado de la cola en el servidor RabbitMQ y debe visualizar que la cola ya no tiene mensajes disponibles tal como se puede observar en la siguiente figura.



9. Reto

Para probar este middleware realizará una aplicación sencilla la cual elegirá entre una de estas dos:

- **App1-canales:** tendrá dos (2) módulos principales: Proveedor de anuncios (AdFuente), es decir, es desde donde se generan los mensajes y un Cliente de anuncios (AdCliente), el cual recibe los mensajes enviados a un Canal por un AdFuente.

  La aplicación debe gestionar:

1. Canales y Mensajes: *Canales* a través de los cuales fluyen los *Mensajes* originados en un *AdFuente* (pueden haber varias fuentes en un mismo Canal) hacia uno o más AdCliente
2. Envíos: son los mensajes que son transmitidos por un AdFuente hacia un Canal y que le debe llegar a TODOS los AdCliente. Tenga en cuenta la situación cuando los clientes están o no en línea, que supuesto realizar al respecto.

Los canales son temáticos (deportes, tecnología, noticias, culinaria, bolsa, etc) y deben ser gestionados en el sistema (crear, modificar, borrar, etc un canal).

Los anuncios son recibidos por los clientes en modo PUSH y PULL. Por anuncios PUSH se entiende la característica de recibir en un cliente mensajes sin haber sido solicitada explícitamente por el cliente. Por anuncios PULL se entiende cuando el cliente explícitamente recupera mensajes de un canal específico. Debe tener en cuenta que criterio utilizaría para borrar mensajes de un canal.a cola.

- **App2-colas:** Simulación de un gestor de tareas para procesamiento distribuido: Tenemos un conjunto de clientes que tienen cada uno tareas para procesar (task_i) y tenemos un conjunto de servidores que ejecutan esas tareas (task_i). Los clientes (publishers) envían a una cola las tareas específicas a procesar, estas tareas son almacenadas en una cola del MOM. Los servidores (subscribers) van tarea por tarea a la cola a retirar una task_i específica y ejecutarla. Una vez un servidor termina una tarea específica notificará por otro medio al cliente específico. En síntesis:

    1. Cola: Almacenamiento en el MOM donde se depositan las tareas.
    2. Mensaje: Identificador de la tarea, el username y el email para notificación
    3. Cliente: Publisher que envía tareas a la cola.
    4. Server: Subscribe quien extrae uno a uno tareas de la cola.

10. Para efectos de este laboratorio consulte los siguientes aspectos:

    1. ¿Cuál es el protocolo que utiliza la aplicación cliente para comunicarse con el servidor RabbitMQ? Consulte y revise el protocolo que utiliza el cliente para comunicarse con el servidor. Para efectos de síntesis, realice un mapa mental de las características del protocolo.