Advanced Message Queuing Protocol (AMQP)

Desarrollo de Sistemas en Red

Los servicios deben ser capaces de comunicarse entre sí. Se intercambia información y se transmiten datos.

AMPQ

¿Qué es?

- es un protocolo de mensajería de estándar abierto que permite la **comunicación asíncrona** entre aplicaciones y sistemas distribuidos, facilitando el intercambio seguro y eficiente de mensajes a través de redes.

Principales características

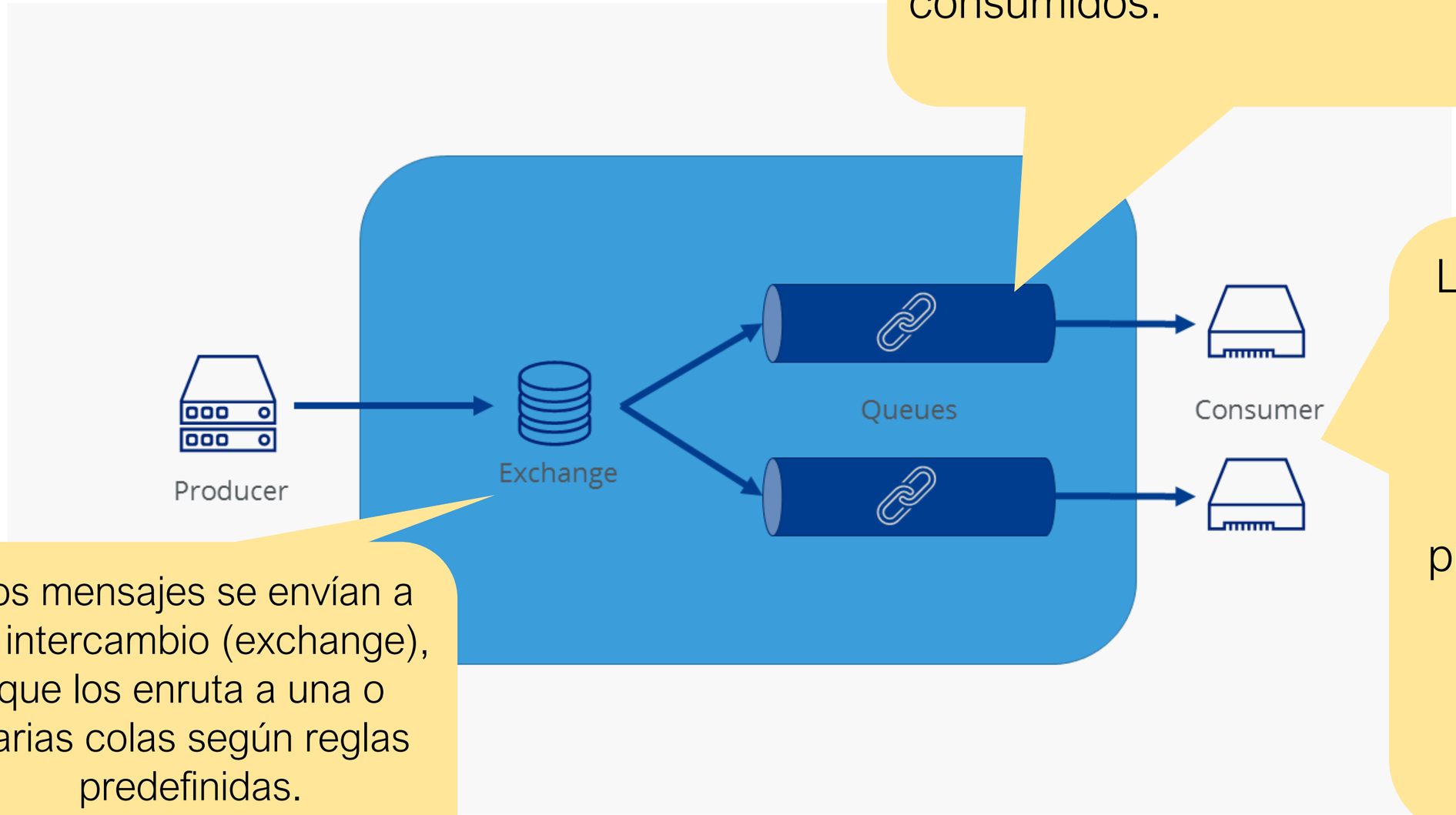
- AMQP es un protocolo a nivel de aplicación, lo que significa que define cómo se estructuran y transmiten los mensajes entre sistemas.

- Permite la creación de colas de mensajes, enrutamiento flexible (punto-a-punto y publicación-suscripción), entrega garantizada y seguridad.

- Es independiente del lenguaje de programación y de la plataforma, lo que facilita la interoperabilidad entre diferentes sistemas

Funcionamiento básico

Las colas actúan como buffers temporales, almacenando los mensajes hasta que estén listos para ser consumidos.



Los mensajes se envían a un intercambio (exchange), que los enruta a una o varias colas según reglas predefinidas.

Los consumidores (receptores) extraen los mensajes de las colas para procesarlos, lo que permite una comunicación asíncrona y desacoplada.

Usos comunes

- AMQP es ampliamente utilizado en arquitecturas de microservicios, sistemas distribuidos y aplicaciones en la nube, donde se requiere una comunicación confiable y escalable entre componentes.

- Herramientas populares como RabbitMQ y Apache ActiveMQ implementan AMQP para gestionar la mensajería entre aplicaciones.

Ejemplo de uso RabbitMQ

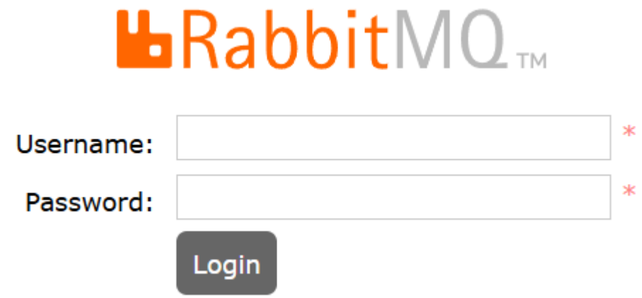
Generaremos un contenedor de Docker con el servicio de RabbitMQ

```
docker run -d --name rabbit -p 5672:5672 -p 15672:15672 rabbitmq:3-management
```

Para probar que funciona accede a: http://<host_ip>:15672

user: guest

pass: guest



https://www.rabbitmq.com/tutorials/amqp-concepts

Dependencias para el ejemplo

--- para el consumidor ---

```
pip install pika
```

--- para el productor ---

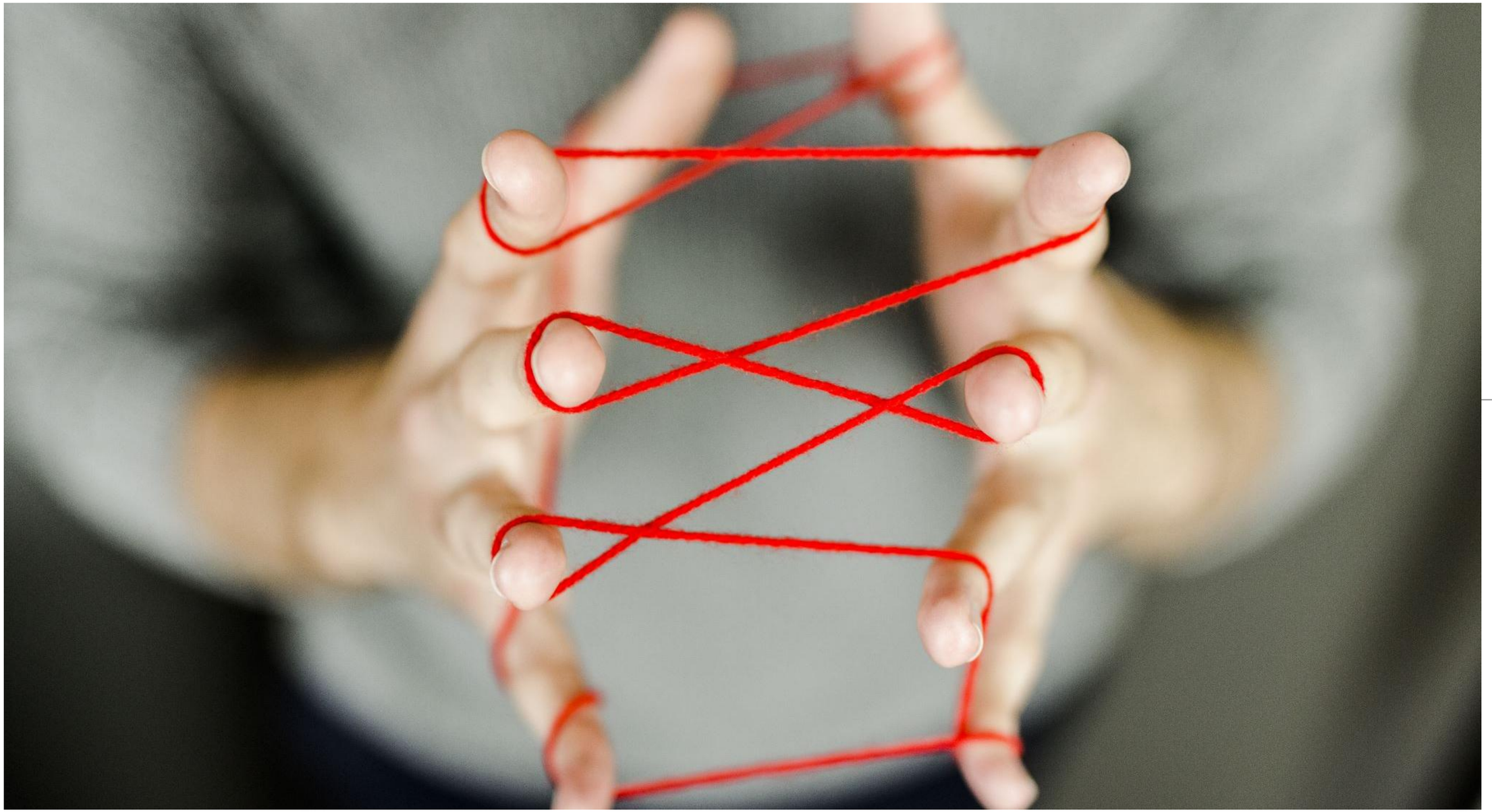
```
npm install amqplib
```

producer.js

```
const amqp = require('amqplib');


const publishOrder = async (order) => {
  const connection = await amqp.connect('amqp://guest:guest@192.168.33.20');
  const channel = await connection.createChannel();


  const exchange = 'orders';
  const routingKey = 'order.created';


  await channel.assertExchange(exchange, 'topic', { durable: true });
  channel.publish(exchange, routingKey, Buffer.from(JSON.stringify(order)));


  console.log("[x] Enviado:", order);


  setTimeout(() => {
    connection.close();
  }, 500);
}


publishOrder({
  id: 'A001',
  customer: 'Mario López',
  total: 200
});
```

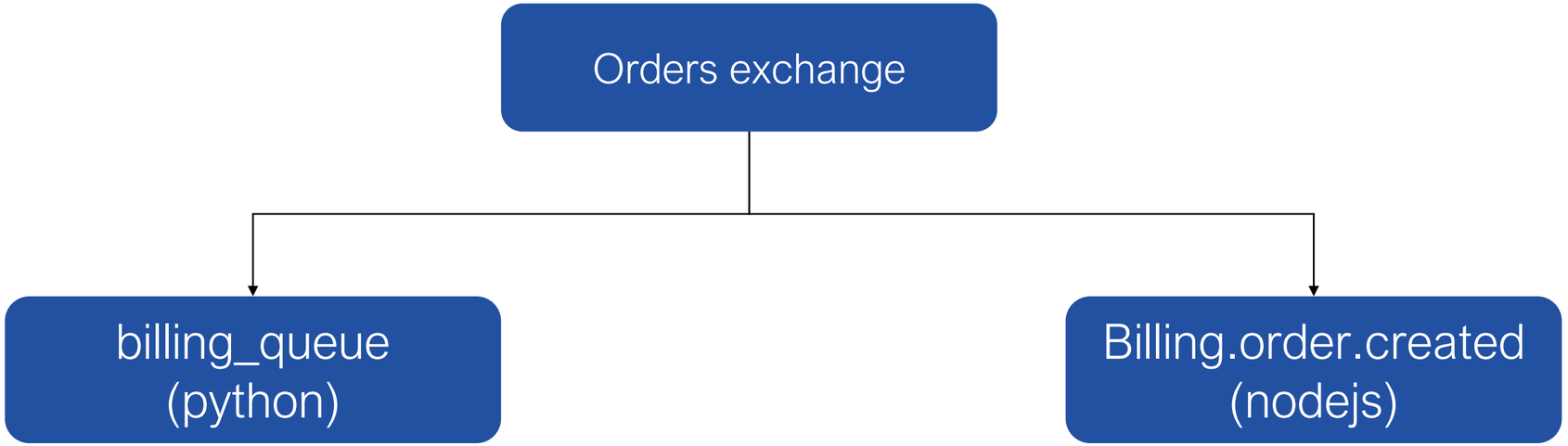
consumer.py

```python
import pika, json

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='192.168.33.20',credentials=pika.PlainCredentials("guest",
"guest"))
)
channel = connection.channel()

exchange = "orders"
routing_key = "order.created"
queue = "billing_queue"

channel.exchange_declare(exchange=exchange, exchange_type="topic", durable=True)
channel.queue_declare(queue=queue, durable=True)
channel.queue_bind(queue=queue, exchange=exchange, routing_key=routing_key)

def callback(ch, method, properties, body):
    order = json.loads(body.decode())
    print("[>] Procesando orden:", order)
    ch.basic_ack(delivery_tag=method.delivery_tag)

channel.basic_consume(queue=queue, on_message_callback=callback)

print("Esperando órdenes para facturación...")
channel.start_consuming()
```



Ejemplo del instructor

En el ejemplo, vimos un esquema de eventos **pub-sub**



Así se ven típicamente los microservicios

- billing-service → factura
- notification-service → manda correo
- Analytics-service → guarda en data lake

Cada servicio tiene su propia cola

¿Cuándo NO recibirían los dos el mismo mensaje?

Cuando ambos se suscriben a **la misma cola**.

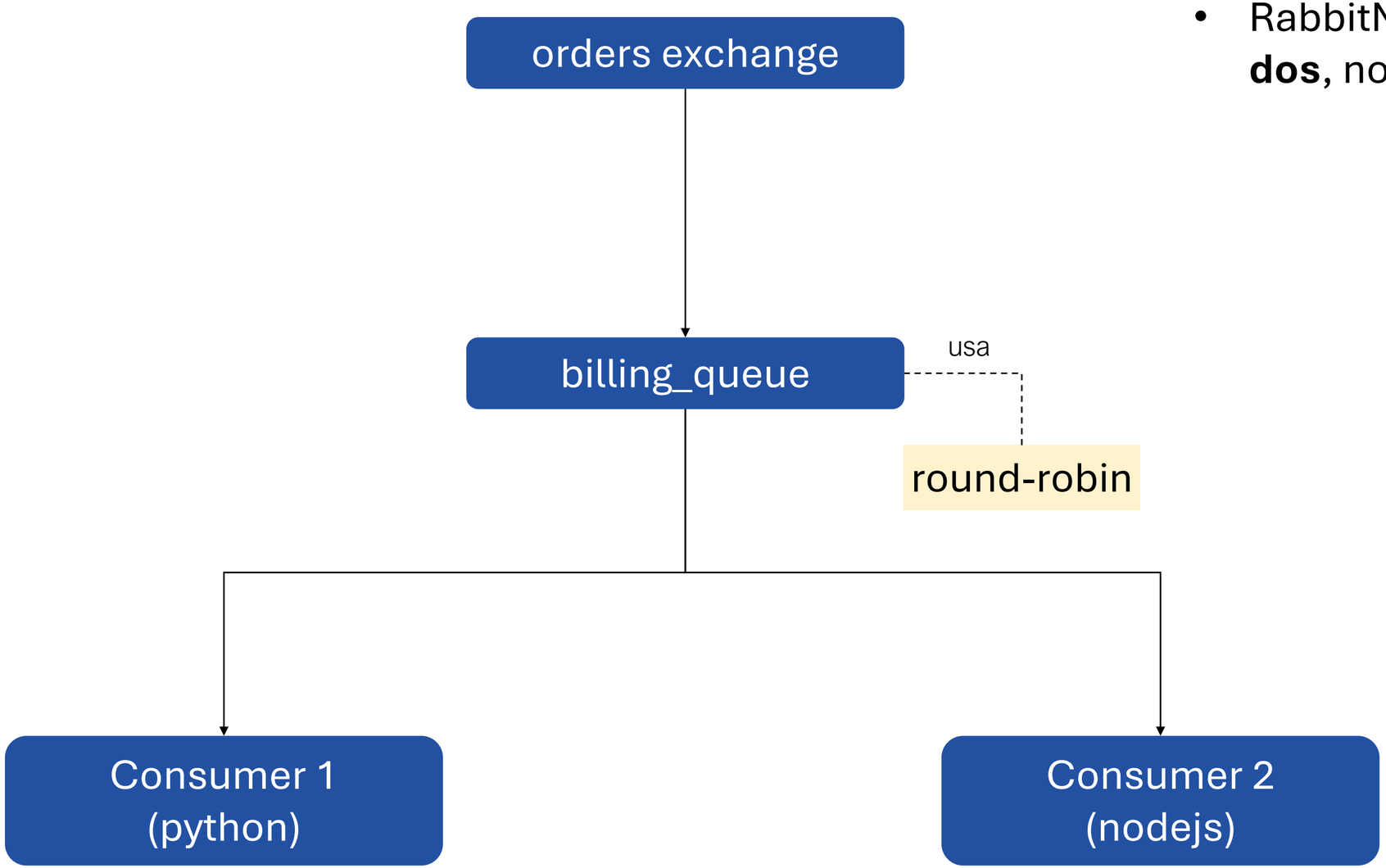
--- En el consumer.py ---

```
queue = "billing_queue"
```

--- En el consumer.js ---

```
const queue = 'billing_queue'
```

- Llega un mensaje a billing_queue.
- RabbitMQ se lo entrega a **uno de los dos**, no a ambos.



Esquema de eventos **competing consumers**

Aspectos de diseño

- ¿En qué momento abrir el *channel* y encolar el mensaje?
  - Conexión y channel: **una vez al inicio del microservicio**
- ¿Publicar el mensaje en el controller?
  - Tecnicamente **puedes**, pero es mejor separarlo un poco. Usa la capa de servicios