Arquitecturas de Mensajería en Sistemas Distribuidos

Infraestructura: Docker + RabbitMQ | Sistemas Distribuidos

Integrantes:

● Barroso, Gonzalo ● Carreño, Hugo ● Velasco, Benjamin

Profesor:

● Carlos Emmanuel Absch Guillaumin

CONTENIDO DE LA PRESENTACIÓN

1. INTRODUCCIÓN

El Problema

Los sistemas distribuidos modernos necesitan:

✗ Desacoplar componentes
✗ Tolerar fallos
✗ Escalar horizontalmente

La comunicación síncrona (HTTP) introduce acoplamiento temporal y dependencia directa entre componentes.

La Solución: Mensajería Asíncrona

✓ Los sistemas operan de forma independiente

✓ El broker actúa como intermediario

✓ Los mensajes persisten aunque el consumidor no esté activo

✓ Permite múltiples consumidores y balanceo de carga

2. ¿QUÉ ES RABBITMQ?

RabbitMQ es un broker de mensajería open-source que implementa el protocolo AMQP (Advanced Message Queuing Protocol). Permite la comunicación indirecta entre productores y consumidores mediante colas y exchanges.

Producer

Envía mensajes al Exchange. No conoce a los consumidores.

Exchange

Recibe mensajes y los enruta a las colas según reglas (direct, fanout, topic).

Queue

Almacena mensajes hasta que son consumidos. Puede ser durable.

Consumer

Lee y procesa mensajes. Confirma con ACK al terminar.

EMPRESAS QUE USAN RABBITMQ

Adoptado por grandes compañías para escalar sus sistemas de mensajería

Instagram

Redes Sociales

Notificaciones y procesamiento de actividad de usuarios a escala masiva.

Reddit

Comunidad / Media

Cola de tareas para procesamiento de votos, posts y moderación de contenido.

Mozilla

Open Source / Web

Pipeline de builds y distribución de mensajes entre servicios internos.

VMware

Cloud / Software

Infraestructura de mensajería en productos de virtualización y cloud.

NASA

Aeroespacial / Gov

Transmisión de datos de telemetría entre sistemas de misión crítica.

Zalando

E-Commerce

Comunicación entre microservicios en su plataforma de moda europea.

3. IMPLEMENTACIÓN — Parte A: Infraestructura Docker

docker-compose.yml

```yaml
version: '3.8'
services:
  rabbitmq:
    image: rabbitmq:3-management
    ports:
      - "5672:5672"    # AMQP
      - "15672:15672"  # Admin UI
    environment:
      RABBITMQ_DEFAULT_USER: admin
      RABBITMQ_DEFAULT_PASS: admin
    volumes:
      - rabbit_data:/var/lib/rabbitmq
```

1 **Levantar el broker**

`docker-compose up -d`

2 **Verificar el estado**

`docker ps → contenedor activo`

3 **Admin Panel**

`http://localhost:15672 (admin/admin)`

4 **Resultado**

`Cola visible en Management UI ✓`

3. IMPLEMENTACIÓN — Partes B y C: Productor y Consumidor

producer.py

```
import pika

conn = pika.BlockingConnection(
  pika.ConnectionParameters('localhost'))
channel = conn.channel()

channel.queue_declare(
  queue='tareas', durable=True)

for i in range(10):
  channel.basic_publish(
    exchange='',
    routing_key='tareas',
    body=f'Tarea #{i+1}',
    properties=pika.BasicProperties(
      delivery_mode=2)  # persistente
  )
```

consumer.py

```
import pika, time

channel.basic_qos(prefetch_count=1)

def callback(ch, method, props, body):
  print(f'Procesando: {body}')
  time.sleep(2)  # simula carga
  ch.basic_ack(
    delivery_tag=method.delivery_tag)

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

channel.start_consuming()
# Ejecutar N instancias en paralelo
```

4. PRUEBAS REALIZADAS

P1

Envío y Recepción Básica

✓ 10 mensajes enviados y recibidos correctamente por 1 consumidor.

P2

Distribución entre Consumidores

✓ Con 3 consumidores, RabbitMQ distribuyó los mensajes en round-robin, cada uno procesó ~3-4 tareas.

P3

Reentrega sin ACK

✓ Al detener un consumidor antes del ACK, el mensaje fue reenviado a otro consumidor disponible.

P4

Persistencia tras Reinicio

✓ Los mensajes en cola sobrevivieron al reinicio del broker gracias a durable=True y delivery_mode=2.

5. PREGUNTAS TEÓRICAS

Q1 ¿Diferencia entre comunicación síncrona y asíncrona?

→ Síncrona (HTTP): emisor espera respuesta → acoplamiento temporal. Asíncrona: emisor continúa sin esperar → mayor independencia.

Q2 ¿Rol de RabbitMQ en sistemas distribuidos?

→ Actúa como broker intermediario: desacopla productores y consumidores, gestiona colas y enrutado de mensajes.

Q3 ¿Qué problema resuelven los acknowledgements?

→ Garantizan que el mensaje se procesó exitosamente. Sin ACK, RabbitMQ reencola el mensaje para otro consumidor.

5. PREGUNTAS TEÓRICAS (cont.)

Q4 ¿Qué ocurre si un consumidor falla durante el procesamiento?

→ Sin ACK el mensaje permanece 'unacked'. Al detectar la desconexión, RabbitMQ lo reencola para otro consumidor disponible.

Q5 ¿Ventajas de múltiples consumidores?

→ Paralelismo real: varias tareas en simultáneo. Balanceo de carga automático (round-robin). Mayor throughput y resiliencia ante fallos individuales.

Q6 Explique el concepto de persistencia de mensajes.

→ durable=True persiste la cola en disco. delivery_mode=2 persiste cada mensaje. Ambos combinados sobreviven reinicios del broker.

6. CONCLUSIONES

Desacoplamiento real

Productores y consumidores pueden evolucionar de forma independiente sin afectarse mutuamente.

Alta disponibilidad

La combinación de múltiples consumidores + persistencia garantiza tolerancia a fallos efectiva.

Docker simplifica el despliegue

Levantar el broker con un solo comando permite replicar el entorno en cualquier máquina.

Observabilidad integrada

La Management UI de RabbitMQ permite monitorear colas, tasas de mensajes y conexiones en tiempo real.

¡Gracias!

¿Preguntas?