Informe de Resultados: Ingeniería de Software 3

Taller Mensajería Asíncrona – RabbitMQ

Integrantes:

Levir Heladio Hernandez Suarez

Cristian Alexander Malagon Ruiz

David Santiago Morantes Duarte

Luis Andres Gonzalez Corzo

Código: 2201776

Grupo: C1

Escuela de ingeniería de sistemas e informática

Universidad Industrial de Santander

22 de septiembre de 2024

Demostración del Despliegue de Contenedores

Enlace: Ver video

Proyecto: Ver repositorio

Estructura de Carpetas del Proyecto

Este taller está organizado en carpetas que contienen tanto consumidores como productores, implementados en *Python* con *Flask* y en *Java* con *Spring*. Además, incluye archivos *.yml* que corresponden a las infraestructuras a desplegar con *docker-compose*.

Carpeta "*consumer*":

- **ConsumerController.java:** API REST en Spring, accesible en http://localhost:8082/consumed/messages
- **ConsumerService.java:** Servicio consumidor de RabbitMQ.

Carpeta "consumer-py":

- **consumer.py:** Consumidor de RabbitMQ y API REST en Flask, accesible en http://localhost:8082/consumed/messages

Carpeta "producer":

- **ProducerController.java:** API REST, accesible en http://localhost:8081/publish/{message}
- **ProducerService.java:** Servicio productor de RabbitMQ.

Carpeta "producer-py":

- **producer.py:** Productor de RabbitMQ y API REST en Flask, accesible en http://localhost:8081/publish/{message}

Descripción de los Archivos Dockerfile y Docker-Compose

Cada una de estas carpetas incluye un *Dockerfile* que define las siguientes variables de entorno:

- **`BROKER_NAME`**`=broker-1`
- **`QUEUE_NAME`**`=cola1`

Los archivos *.yml* presentes en cada carpeta corresponden a cada uno de los casos de despliegue planteados en clase. Estos archivos crean un *bróker* utilizando la imagen de *RabbitMQ* y realizan un *healthcheck* para asegurar que *RabbitMQ* esté disponible en el *puerto 15672* y que se pueda crear una cola en el *bróker*.

Además, los contenedores de productores y consumidores están configurados con la etiqueta *depends_on*, lo que asegura que el *bróker* esté operativo antes de establecer las conexiones correspondientes.

Instrucciones para Levantar las Diferentes Infraestructuras

A continuación, se detallan los pasos para levantar cada infraestructura correspondiente a los diferentes casos:

- **CASO 1: [1 Broker, 1 Consumidor, 1 Productor en Java]**

Ejecutar: `docker-compose -f broker-consumer-producer-[java].yml up -d`

- **CASO 1: [1 Broker, 1 Consumidor, 1 Productor en Python]**

Ejecutar: `docker-compose -f broker-consumer-producer-[python].yml up -d`

- **CASO 2: [1 Broker, 2 Consumidores, 2 Produtores en Java]**

Ejecutar: `docker-compose -f broker-consumer[2]-producer[2]-[java].yml up -d`

- **CASO 2: [1 Broker, 2 Consumidores, 2 Produtores en Python]**

Ejecutar: `docker-compose -f broker-consumer[2]-producer[2]-[python].yml up -d`

- **CASO 3: [1 Broker, 2 Consumidores, 2 Produtores en Java y Python]**

Ejecutar: `docker-compose -f broker-consumer[2]-producer[2]-[java-python].yml up -d`

NOTAS IMPORTANTES:

- Asegurarse de utilizar una versión actualizada de Docker-Compose (mínimo v2).
- No levantar más de una infraestructura simultáneamente para evitar conflictos de nombres y puertos. Si desea probar otra configuración, ejecute primero el comando `down`.

Puertos Disponibles por Caso

CASO 1: [1 Broker, 1 Consumidor, 1 Productor]:

- **Bróker:** http://localhost:15672/
- **Productor:** http://localhost:8081/publish/{message}
- **Consumidor:** http://localhost:8082/consumed/messages

CASO 2-3: [1 Broker, 2 Consumidores, 2 Productores]:

- **Bróker:** http://localhost:15672/
- **Productor 1:** http://localhost:8081/publish/{message}
- **Productor 2:** http://localhost:8082/publish/{message}
- **Consumidor 1:** http://localhost:8083/consumed/messages
- **Consumidor 2:** http://localhost:8084/consumed/messages

NOTAS ADICIONALES:

- Es necesario recargar los *consumidores* para ver los nuevos mensajes recibidos.
- Cuando hay más de un consumidor, los mensajes se distribuyen siguiendo el *protocolo Round-Robin*, alternando su recepción.

CASO 4: [1 Broker, 2 Consumidores, 2 Productores y Distribuido]

- **Máquina Broker:**

```
docker run --name broker-1 -e BROKER_NAME=localhost -e
QUEUE_NAME=cola1 -p5672:5672 -p15672:15672 -d rabbitmq:management
```

```
docker exec broker-1 /bin/bash -c 'rabbitmqadmin -u guest -p guest
-H ${BROKER_NAME} -P 15672 declare queue name=${QUEUE_NAME}
durable=true'
```

- **Máquina Productor en Java:**

```
docker build -t java-producer .producer
```

```
docker run --name java-producer-1 -e BROKER_NAME=host -e
QUEUE_NAME=cola1 -p8081:8081 -d java-producer
```

- **Máquina Productor en Python:**

```
docker build -t python-producer .producer-py
```

```
docker run --name python-producer-1 -e BROKER_NAME=host -e
QUEUE_NAME=cola1 -p8082:8081 -d python-producer
```

- **Máquina Consumidor en Java:**

```
docker build -t java-consumer .consumer
```

```
docker run --name java-consumer-1 -e BROKER_NAME=host -e
QUEUE_NAME=cola1 -p8083:8082 -d java-consumer
```

- **Máquina Consumidor en Python:**

```
docker build -t python-consumer .consumer-py
```

```
docker run --name python-consumer-1 -e BROKER_NAME=host -e
QUEUE_NAME=cola1 -p8084:8082 -d python-consumer
```

NOTA FINAL:

La variable de entorno **`BROKER_NAME`** tanto en el *consumidor* como en el *productor* debe ser reemplazada por la *IP* de la *Máquina Broker*.