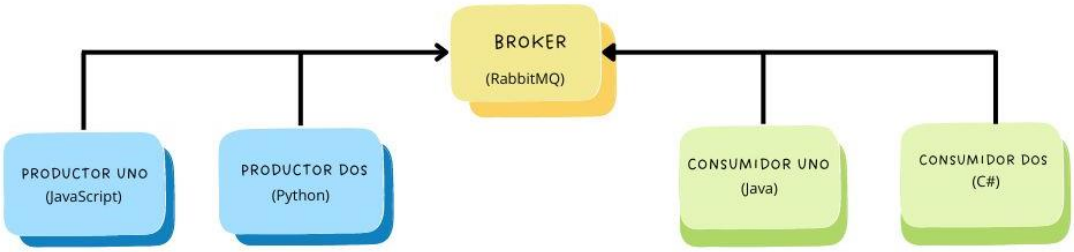
Solución taller 1 – Ingeniería de software III

Integrantes:

- Juan Camilo Lozada Garavito – 2205560
- Daniel Camilo Barrera Pérez – 2205562
- Jerson Julian Cañon Castillo – 2205633
- Santiago González Flores - 2200165

Versión 3



Configuración para broker:

```yaml
broker:
  image: rabbitmq:3.13-management
  ports:
    - "5672:5672"
    - "15672:15672"
  restart: unless-stopped
```

Configuración para productor 1:

Código para enviar mensajes en JavaScript

```
#!/usr/bin/env node
var amqp = require('amqplib/callback_api');

const RABBITMQ_HOST = process.env.RABBITMQ_HOST || 'localhost';
const RABBITMQ_PORT = process.env.RABBITMQ_PORT || '5672';
const MENSAJE = process.env.MENSAJE || 'Hola mundo!';

console.log(`RabbitMQ Host: ${RABBITMQ_HOST}`);
console.log(`RabbitMQ Port: ${RABBITMQ_PORT}`);
console.log(`Mensaje: ${MENSAJE}`);

amqp.connect(`amqp://${RABBITMQ_HOST}:${RABBITMQ_PORT}`, function(error0, connection) {
    if (error0) {
        throw error0;
    }
    connection.createChannel(function(error1, channel) {
        if (error1) {
            throw error1;
        }

        var queue = 'hello';

        channel.assertQueue(queue, {
            durable: false
        });

        setInterval(() => {
            channel.sendToQueue(queue, Buffer.from(MENSAJE));
            console.log(" [x] ENVIADO: %s", MENSAJE);
        }, 3500);

    });
});
```

Dockerfile correspondiente para utilizar en el docker compose

```
FROM node:18.15.0

WORKDIR /app
COPY package*.json .
RUN npm install
COPY producers/send.js .


CMD [ "node", "send.js" ]
```

Configuración para productor 2:

Código para enviar mensajes en Python

```python
import pika
import time
import os

RABBITMQ_HOST = os.environ.get('RABBITMQ_HOST') if os.environ.get('RABBITMQ_HOST') != None else 'localhost'
RABBITMQ_PORT = int(os.environ.get('RABBITMQ_PORT')) if os.environ.get('RABBITMQ_PORT') != None else 5672
MENSAJE = os.environ.get('MENSAJE') if os.environ.get('MENSAJE') != None else 'Hola Mundo'

print("RABBITMQ_HOST: ", RABBITMQ_HOST)
print("RABBITMQ_PORT: ", RABBITMQ_PORT)
print("MENSAJE: ", MENSAJE)

connection = pika.BlockingConnection(pika.ConnectionParameters(host = RABBITMQ_HOST, port = RABBITMQ_PORT))
channel = connection.channel()
channel.queue_declare(queue='hello')

while True:
    channel.basic_publish(exchange='',
                          routing_key='hello',
                          body=MENSAJE)
    print(" [x] ENVIADO: " + MENSAJE, flush=True)
    time.sleep(4)
```

Dockerfile correspondiente para utilizar en el docker compose

```dockerfile
FROM python:3

WORKDIR /app
COPY producers/send.py .
RUN pip install pika

CMD [ "python", "send.py" ]
```

Configuración para consumidor 1:

Código para recibir mensaje con Java

```java
import java.nio.charset.StandardCharsets;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.DeliverCallback;

public class Receiver {

    private final static String QUEUE_NAME = "hello";


    public static void main(String[] argv) throws Exception {

        String value_name = System.getenv("RABBITMQ_HOST");
        String value_host = System.getenv("RABBITMQ_PORT");
        String RABBITMQ_HOST = value_name != null ? value_name : "localhost";
        String RABBITMQ_PORT = value_host != null ? value_host : "5672";

        System.out.println("RABBITMQ_HOST: " + RABBITMQ_HOST);
        System.out.println("RABBITMQ_PORT: " + RABBITMQ_PORT);

        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(RABBITMQ_HOST);
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        channel.queueDeclare(QUEUE_NAME, false, false, false, null);
        System.out.println(" [*] Esperando mensajes en JAVA.... ");

        DeliverCallback deliverCallback = (consumerTag, delivery) -> {
            String message = new String(delivery.getBody(), StandardCharsets.UTF_8);
            System.out.println(" [x] MENSAJE RECIBIDO EN JAVA: '" + message + "'");
        };
        channel.basicConsume(QUEUE_NAME, true, deliverCallback, consumerTag -> { });
    }

}
```

Dockerfile para utilizar en el archivo Docker compose

```dockerfile
FROM openjdk:22

WORKDIR /app
COPY consumers/Receiver_JAVA .

CMD [ "java", "-cp" ,".:amqp-client-5.21.0.jar:slf4j-api-1.7.36.jar:slf4j-simple-1.7.36.jar", "Receiver" ]
```

Configuración para consumidor 2:

Código para recibir el mensaje en C#

```
using System.Text;
using RabbitMQ.Client;
using RabbitMQ.Client.Events;

var value_host = Environment.GetEnvironmentVariable("RABBITMQ_HOST");
var value_port = Environment.GetEnvironmentVariable("RABBITMQ_PORT");
var RABBITMQ_HOST = value_host != null ? value_host : "localhost";
var RABBITMQ_PORT = value_port != null ?  Int32.Parse(value_port): 5672;

Console.WriteLine($"RABBITMQ__HOST: {RABBITMQ_HOST}");
Console.WriteLine($"RABBITMQ__HOST: {RABBITMQ_PORT}");

var factory = new ConnectionFactory { HostName = RABBITMQ_HOST, Port = RABBITMQ_PORT};
using var connection = factory.CreateConnection();
using var channel = connection.CreateModel();

channel.QueueDeclare(queue: "hello",
                     durable: false,
                     exclusive: false,
                     autoDelete: false,
                     arguments: null);

Console.WriteLine(" [*] ESPERANDO MENSAJES DESDE .NET");

var consumer = new EventingBasicConsumer(channel);

consumer.Received += (model, ea) =>
{
    var body = ea.Body.ToArray();
    var message = Encoding.UTF8.GetString(body);
    Console.WriteLine($" [x] MENSAJE RECIBIDO DESDE .NET ->  {message}");
};
channel.BasicConsume(queue: "hello",
                     autoAck: true,
                     consumer: consumer);

await Task.Delay(-1);
```

Dockerfile correspondiente para utilizar en el docker compose

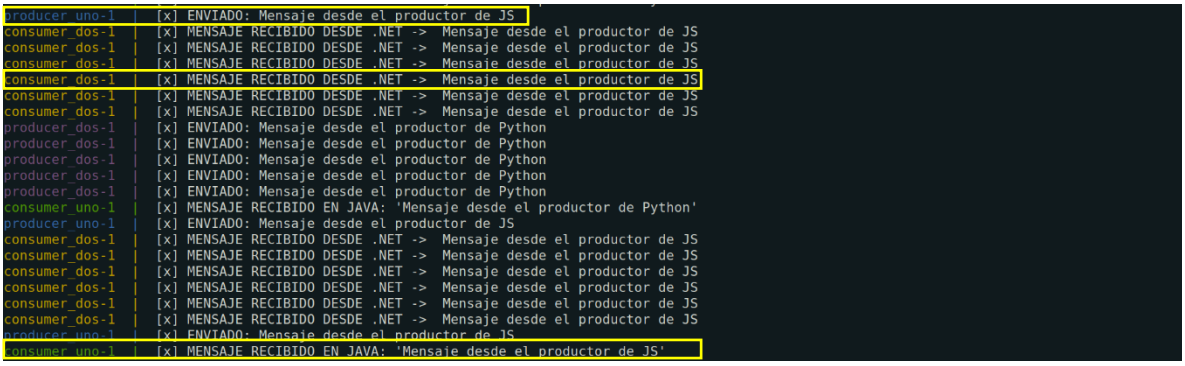
```
FROM mcr.microsoft.com/dotnet/sdk:8.0

WORKDIR /app
COPY consumers/Receiver_NET .
RUN dotnet add package RabbitMQ.Client
CMD [ "dotnet", "run" ]
```

Docker compose:

```yaml
services:
  broker:
    image: rabbitmq:3.13-management
    ports:
      - "5672:5672"
      - "15672:15672"
    restart: unless-stopped
  producer_uno:
    build:
      context: .
      dockerfile: Dockerfile_producer_js
    depends_on:
      - broker
    restart: unless-stopped
    environment:
      - RABBITMQ_HOST=broker
      - MENSAJE=Mensaje desde el productor de JS
  producer_dos:
    build:
      context: .
      dockerfile: Dockerfile_producer_py
    depends_on:
      - broker
    restart: unless-stopped
    environment:
      - RABBITMQ_HOST=broker
      - MENSAJE=Mensaje desde el productor de Python
  consumer_uno:
    build:
      context: .
      dockerfile: Dockerfile_consumer_java
    depends_on:
      - broker
    restart: unless-stopped
    environment:
      - RABBITMQ_HOST=broker
  consumer_dos:
    build:
      context: .
      dockerfile: Dockerfile_consumer_net
    depends_on:
      - broker
    restart: unless-stopped
    environment:
      - RABBITMQ_HOST=broker
```

Ejecución del proyecto

- En la siguiente imagen, podemos visualizar como el producer_uno envía el mensaje y como los consumidores (consumer_uno y consumidor 2) lo reciben



- En la siguiente imagen, podemos visualizar como el producer_dos envía el mensaje y como los consumidores (consumer_uno y consumidor 2) lo reciben

Anexos

Enlace del video: Hacer clic

Enlace del repositorio en GitHub con las 3 versiones del taller: Hacer clic