

Project Presentation: Distributed System for Data Validation and Persistence

Daniela Cadena
Eladio Huamani

National University of Engineering
Faculty of Sciences

CC4P1 - Concurrent Programming and Distributed Systems



- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details
- 5 Docker and Docker Compose
- 6 Testing and Evaluation
- 7 Conclusions

- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details
- 5 Docker and Docker Compose
- 6 Testing and Evaluation
- 7 Conclusions

1. Introduction

- **Project Goal:** Implement a distributed system that simulates a real-world scenario of asynchronous data validation and persistence, using messaging middleware.
- **Key Technologies:**
 - Node.js (Client)
 - Java (Validation Service)
 - Python (Persistence Service)
 - RabbitMQ (Middleware)
 - PostgreSQL (Main DB)
 - MariaDB (Validation DB)
 - Docker & Docker Compose
- **Simulation:** A flow where user data is sent, validated (e.g., DNI), and then stored in a decoupled and robust manner.

- 1 Introduction
- 2 System Architecture**
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details
- 5 Docker and Docker Compose
- 6 Testing and Evaluation
- 7 Conclusions

2. System Architecture (Part 1)

- **Overview:**
 - A system composed of multiple independent services that collaborate through a message bus.
-
- **Main Components:**
 - **Client (Node.js):** Interface for data entry. Sends data to RabbitMQ.
 - **DNI Validator Service (Java):** Consumes from RabbitMQ, validates DNI against MariaDB, and resends if valid.
 - **Persistence Service (Python):** Consumes validated messages and stores them in PostgreSQL.

2. System Architecture (Part 2)

- **Databases:**
 - **DB1 (PostgreSQL):** Stores validated user information (ID, name, email, DNI, etc.).
 - **DB2 (MariaDB):** Contains reference records for DNI validation.

- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ**
- 4 Implementation Details
- 5 Docker and Docker Compose
- 6 Testing and Evaluation
- 7 Conclusions

3. Communication Flow: RabbitMQ (Part 1)

- What is RabbitMQ?
 - An open-source message broker that acts as an intermediary between producers and consumers.
 - Enables asynchronous and decoupled communication.
- Detailed Message Flow:
 - 1 **Client (Node.js)** sends user data to the `validate_dni` queue.
 - 2 **Validator Service (Java)** consumes from `validate_dni`.
Validates DNI against **DB2 (MariaDB)**.
 - 3 If DNI is valid, it publishes the message to `validate_save_user`.
 - 4 **Persistence Service (Python)** consumes from `validate_save_user`. Saves user to **DB1 (PostgreSQL)**.

3. Communication Flow: RabbitMQ (Part 2)

- **Key RabbitMQ Concepts Used:**
 - **Producer:** Node.js Client.
 - **Consumers:** Java and Python Services.
 - **Queues:** `validate_dni`, `validate_save_user` (durable).
 - **Persistent Messages:** To prevent message loss.
 - **Acknowledgements (ACKs):** Confirmation of processing.
- **Benefits of RabbitMQ:**
 - Decoupling, Scalability, Fault Tolerance, Asynchronous Processing.

Visual Suggestion

- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details**
- 5 Docker and Docker Compose
- 6 Testing and Evaluation
- 7 Conclusions

4. Implementation Details (Part 1)

- **Client (Node.js):**
 - Use of `inquirer` for interactive capture.
 - Connection to RabbitMQ (`amqplib`), sending to `validate_dni`.
 - Messages as persistent: `true`.
- **DNI Validator (Java):**
 - Connection to RabbitMQ (`com.rabbitmq.client`) with retries.
 - Consumption from `validate_dni`.
 - DNI extraction (regex), validation against MariaDB (JDBC).
 - Publishing to `validate_save_user` if successful.
 - Handling of `basicAck`.

4. Implementation Details (Part 2)

- **Consumer and Persistence (Python):**
 - Connection to RabbitMQ (pika) with `BlockingConnection`.
 - Consumption of messages from `validate_save_user`.
 - Data processing and saving to PostgreSQL (psycopg2).
 - **Concurrency:** Use of threads (`threading`) to process multiple messages and improve performance.
 - Handling of `basic_ack`.

Diagrama de Arquitectura y Protocolo del Sistema Distribuido

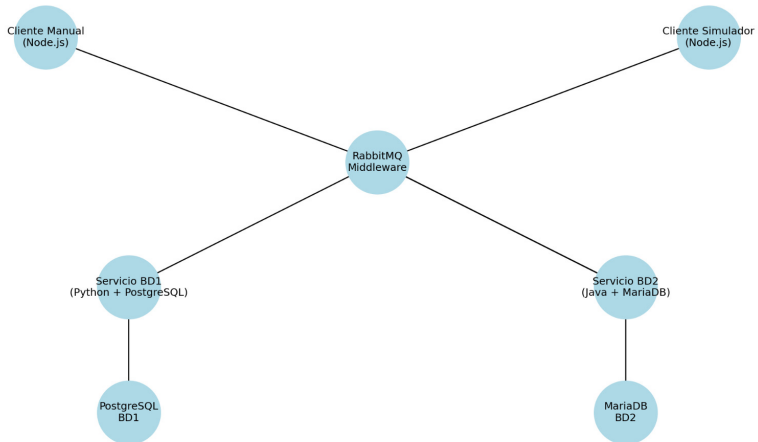


Figura 1: Architechture Diagram

- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details
- 5 Docker and Docker Compose**
- 6 Testing and Evaluation
- 7 Conclusions

5. Docker and Docker Compose (Part 1)

- **Why Docker?**
 - Consistent Environments.
 - Service Isolation.
 - Simple Scalability.
 - Simplified Deployment (`docker-compose up`).
- **Orchestration with Docker Compose**
(`docker-compose.yml`):
 - Service definition: `rabbitmq`, `bd1` (PostgreSQL), `bd2` (MariaDB), `client-simulator` (Node.js), `service_bd1` (Python), `service_bd2` (Java).
 - Configuration of internal Docker networks.

5. Docker and Docker Compose (Part 2)

- **The Role of Dockerfiles:**
 - **client-simulator (Node.js):** Base node:18-alpine, npm install.
 - **service_bd2 (Java):** Base openjdk:17, compiles, RabbitMQ retries.
 - **service_bd1 (Python):** Base python:3.11-slim, installs pika, psycopg2.
 - Lightweight and specific images.
- **Lifecycle with docker-compose up:**
 - 1 Start rabbitmq, bd1, bd2.
 - 2 SQL scripts to initialize DBs.
 - 3 Java/Python services wait for RabbitMQ and listen to queues.
 - 4 Interactive Node.js for data input.

5. Docker and Docker Compose (Part 3)

- **Robustness Thanks to Docker:**
 - Retry logic in services to handle race conditions at startup.
 - Volume mounting for SQL scripts.
 - Secure and fluid communication over the Docker network.

```

daniela@fedora: ~/main $ cd 72
~/../..../middleware-config $ docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED          STATUS          PORTS
d922e359699c   rabbitmq:4-management              "docker-entrypoint.s..." 6 seconds ago    Up 5 seconds    4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, [::]:5672->5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672->15672/tcp, [::]:15672->15672/tcp
a4019b01ae38   postgres:15                         "docker-entrypoint.s..." 6 seconds ago    Up 5 seconds    0.0.0.0:5432->5432/tcp, [::]:5432->5432/tcp
1151c2935bf7   mariadb:10.11                      "docker-entrypoint.s..." 6 seconds ago    Up 5 seconds    0.0.0.0:3306->3306/tcp, [::]:3306->3306/tcp

```

Figura 2: Containers

- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details
- 5 Docker and Docker Compose
- 6 Testing and Evaluation**
- 7 Conclusions

6. Testing and Evaluation

- **Performance Evaluation:**

- Stress test script (1000 random entries) to evaluate performance and resilience.

- **Data Validation:**

- Queries to DB1 (PostgreSQL) and DB2 (MariaDB) to verify correct insertion and validation logic.

- **Manual Entry:**

- System supports interactive user registration from the client terminal.

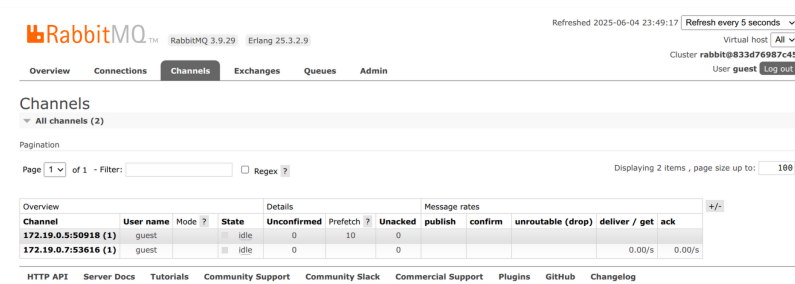


Figura 3: Channels

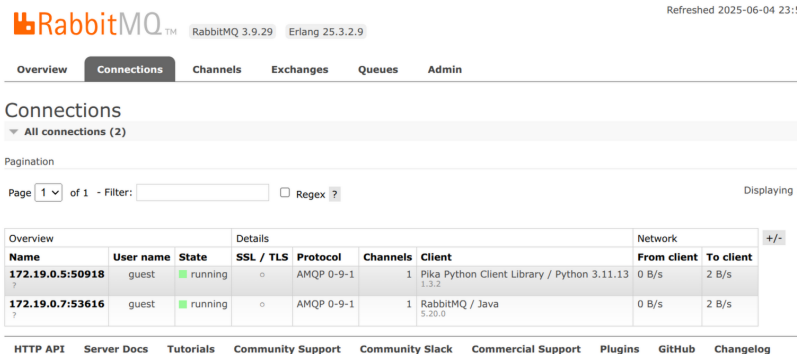


Figura 4: Connections services

```
✓ cliente-simulador          Built          0.0s
✓ servicio_bd1               Built          0.0s
✓ servicio_bd2               Built          0.0s
✓ Network middleware-config_default Created        0.2s
✓ Container middleware-config-rabbitmq-1 Started        0.9s
✓ Container middleware-config-bd2-1 Started          0.6s
✓ Container middleware-config-bd1-1 Started          0.7s
✓ Container cliente-simulador Started          1.4s
✓ Container middleware-config-servicio_bd2-1 Started      1.4s
✓ Container middleware-config-servicio_bd1-1 Started      1.3s
✓ Container middleware-config-cliente-1 Started          1.5s
daniela@fedora ~$ docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
2659db64d009   middleware-config-servicio_bd2      "java -jar app.jar"      8 seconds ago Up 7 seconds  middleware-config-servicio_bd2-1
e775806c6c16   middleware-config-servicio_bd1      "python consumer_bd1..." 8 seconds ago Up 7 seconds  middleware-config-servicio_bd1-1
d4d240073537   rabbitmq:3.9-management            "docker-entrypoint.s..." 8 seconds ago Up 8 seconds  4369/tcp, 5671/tcp, 0.0.0.0:5672→5672/tcp, [::]:5672→5672/tcp, 15671/tcp, 15691-15692/tcp, 25672/tcp, 0.0.0.0:15672→15672/tcp, [::]:15672→15672/tcp  middleware-config-rabbitmq-1
16c08d1d54cd   mariadb:10.11                      "docker-entrypoint.s..." 8 seconds ago Up 8 seconds  0.0.0.0:3306→3306/tcp, [::]:3306→3306/tcp  middleware-config-bd2-1
56a6dae04bc0   postgres:15                         "docker-entrypoint.s..." 8 seconds ago Up 8 seconds  0.0.0.0:5432→5432/tcp, [::]:5432→5432/tcp  middleware-config-bd1-1
```

Figura 5: Up Containers


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
cliente-simulador | Enviado usuario 963: 60000962
cliente-simulador | Enviado usuario 964: 60000963
cliente-simulador | Enviado usuario 965: 60000964
cliente-simulador | Enviado usuario 966: 60000965
cliente-simulador | Enviado usuario 967: 60000966
cliente-simulador | Enviado usuario 968: 60000967
cliente-simulador | Enviado usuario 969: 60000968
cliente-simulador | Enviado usuario 970: 60000969
cliente-simulador | Enviado usuario 971: 60000970
cliente-simulador | Enviado usuario 972: 60000971
cliente-simulador | Enviado usuario 973: 60000972
cliente-simulador | Enviado usuario 974: 60000973
cliente-simulador | Enviado usuario 975: 60000974
cliente-simulador | Enviado usuario 976: 60000975
cliente-simulador | Enviado usuario 977: 60000976
cliente-simulador | Enviado usuario 978: 60000977
cliente-simulador | Enviado usuario 979: 60000978
cliente-simulador | Enviado usuario 980: 60000979
cliente-simulador | Enviado usuario 981: 60000980
cliente-simulador | Enviado usuario 982: 60000981
cliente-simulador | Enviado usuario 983: 60000982
cliente-simulador | Enviado usuario 984: 60000983
cliente-simulador | Enviado usuario 985: 60000984
cliente-simulador | Enviado usuario 986: 60000985
cliente-simulador | Enviado usuario 987: 60000986
cliente-simulador | Enviado usuario 988: 60000987
cliente-simulador | Enviado usuario 989: 60000988
cliente-simulador | Enviado usuario 990: 60000989
cliente-simulador | Enviado usuario 991: 60000990
cliente-simulador | Enviado usuario 992: 60000991
cliente-simulador | Enviado usuario 993: 60000992
cliente-simulador | Enviado usuario 994: 60000993
cliente-simulador | Enviado usuario 995: 60000994
cliente-simulador | Enviado usuario 996: 60000995
cliente-simulador | Enviado usuario 997: 60000996
cliente-simulador | Enviado usuario 998: 60000997
cliente-simulador | Enviado usuario 999: 60000998
cliente-simulador | Enviado usuario 1000: 60000999
cliente-simulador | Terminé el envío masivo.
cliente-simulador exited with code 0
```

Figura 6: Send data

```
~/../..../sistema-distribuido-middleware > docker exec -i postgres psql -U user -d bd1 -c "SELECT * FROM usuarios;"
```

id	nombre	correo	clave	dni	telefono	amigos
1	Maye	usuario24@correo.com	L07L1KtqX8syZrT	60000024	900000024	{}
2	Alphonso	usuario57@correo.com	SzEUmu5Djc21X3w	60000057	900000057	{}
3	Yasmin	usuario100@correo.com	uZNoSyurJZTBu0U	60000100	900000100	{2}
4	Madyson	usuario26@correo.com	wVGXdZ7Kd4IWNEA	60000026	900000026	{}
5	Wyatt	usuario108@correo.com	PGkFVYoahYBPpDh	60000108	900000108	{3,4}
6	Benedict	usuario111@correo.com	wmlUXYLE7aPb9eOf	60000111	900000111	{}
7	Abbie	usuario78@correo.com	VkdNggNvimz4grg	60000078	900000078	{}
8	Emanuel	usuario5@correo.com	fShi8N2ViUg1LLW	60000005	900000005	{6}
9	Monica	usuario31@correo.com	yUENegWWhnJ3uD0	60000031	900000031	{}
10	Frieda	usuario30@correo.com	K4q8NBd4LQzb4FB	60000030	900000030	{}
11	Ezequiel	usuario118@correo.com	QVN5wWx_dnc15r2	60000118	900000118	{2}
12	Josiane	usuario35@correo.com	WpYzhg4he5nvHoQ	60000035	900000035	{}
13	Caleb	usuario28@correo.com	Vb_ADr47xctkcyy	60000028	900000028	{}
14	Alexandrine	usuario1@correo.com	40b2u8G5s1uZmJwm	60000010	900000010	{12}
15	Rocio	usuario133@correo.com	vSapCL0qf2cXBtp	60000133	900000133	{}
16	Roberto	usuario0@correo.com	notEzERY1ZMJP8W	60000000	900000000	{4}
17	Randi	usuario2@correo.com	PkZEZnsuva06IaQ	60000002	900000002	{7}
18	Maegan	usuario130@correo.com	w0b0H7_DXxIpJ3v	60000130	900000130	{}
19	Alexzander	usuario14@correo.com	p03PgYfY1ZiRrw	60000014	900000014	{}
20	Nyah	usuario2@correo.com	Sfzj4_MJSKH1_o8	60000020	900000020	{11,18}
21	Sonia	usuario37@correo.com	mIQ7RMLBZf7rKP	60000037	900000037	{16}
22	Adolphus	usuario33@correo.com	3ZJMjwAsqhFh3Lw	60000033	900000033	{9,11}
23	Garnet	usuario117@correo.com	mg0KbJA6KKv8vf	60000117	900000117	{6}
24	Reggie	usuario149@correo.com	iZW_xcph9q_YTND	60000149	900000149	{20}
25	Lenna	usuario34@correo.com	T4eZwustnmjOtJ	60000034	900000034	{13,20}
26	Laurianne	usuario135@correo.com	777JqQsYKY5RuTL	60000135	900000135	{8,15}
27	Willie	usuario91@correo.com	JXuBxmQhBJK1DEH	60000091	900000091	{16}
28	Virgie	usuario22@correo.com	gPwH2EhyJC9Z_ig	60000022	900000022	{}
29	Ines	usuario12@correo.com	7YYIcGwr_2JzltW	60000012	900000012	{9}
30	Nickolas	usuario23@correo.com	nsMh7piil3bLjI4	60000023	900000023	{}
31	Hubert	usuario25@correo.com	gsGzJ9DIGiy_gdt	60000025	900000025	{11,23}

Figura 7: active users

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
Guardado usuario 60000761 (amigos válidos=0)
Guardado usuario 60000323 (amigos válidos=1)
Guardado usuario 60000770 (amigos válidos=4)
Guardado usuario 60000406 (amigos válidos=2)
Guardado usuario 60000784 (amigos válidos=1)
Guardado usuario 60000790 (amigos válidos=2)
Guardado usuario 60000799 (amigos válidos=2)
Guardado usuario 60000811 (amigos válidos=2)
Guardado usuario 60000333 (amigos válidos=3)
Guardado usuario 60000869 (amigos válidos=3)
Guardado usuario 60000576 (amigos válidos=1)
Guardado usuario 60000826 (amigos válidos=4)
Guardado usuario 60000341 (amigos válidos=3)
Guardado usuario 60000471 (amigos válidos=3)
Guardado usuario 60000593 (amigos válidos=4)
Guardado usuario 60000843 (amigos válidos=0)
Guardado usuario 60000598 (amigos válidos=3)
Guardado usuario 60000478 (amigos válidos=2)
Guardado usuario 60000481 (amigos válidos=3)
Guardado usuario 60000482 (amigos válidos=2)
Guardado usuario 60000248 (amigos válidos=4)
Guardado usuario 60000602 (amigos válidos=1)
Guardado usuario 60000535 (amigos válidos=2)
Guardado usuario 60000607 (amigos válidos=2)
Guardado usuario 60000492 (amigos válidos=4)
Guardado usuario 60000617 (amigos válidos=2)
Guardado usuario 60000877 (amigos válidos=1)
Guardado usuario 60000889 (amigos válidos=0)
Guardado usuario 60000901 (amigos válidos=4)
Guardado usuario 60000912 (amigos válidos=4)
daniela@fedora → service-bd1-python 3.13) C:\main s 723 -10
22:15:11 ✓ took 93ms

service-bd2 ✓ DNI válido → reenviado: 60000933
service-bd2 ✗ DNI no existe: 60000934
service-bd2 ✗ DNI no existe: 60000935
service-bd2 ✗ DNI no existe: 60000936
service-bd2 ✓ DNI válido → reenviado: 60000937
service-bd2 ✓ DNI válido → reenviado: 60000939
service-bd2 ✓ DNI válido → reenviado: 60000940
service-bd2 ✗ DNI no existe: 60000941
service-bd2 ✗ DNI no existe: 60000942
service-bd2 ✗ DNI no existe: 60000943
service-bd2 ✓ DNI válido → reenviado: 60000944
service-bd2 ✓ DNI válido → reenviado: 60000945
service-bd2 ✓ DNI válido → reenviado: 60000946
service-bd2 ✗ DNI no existe: 60000947
service-bd2 ✓ DNI válido → reenviado: 60000949
service-bd2 ✗ DNI no existe: 60000950
service-bd2 ✗ DNI no existe: 60000951
service-bd2 ✓ DNI válido → reenviado: 60000953
service-bd2 ✗ DNI no existe: 60000954
service-bd2 ✗ DNI no existe: 60000955
service-bd2 ✗ DNI no existe: 60000958
service-bd2 ✗ DNI no existe: 60000961
service-bd2 ✗ DNI no existe: 60000957
service-bd2 ✗ DNI no existe: 60000960
service-bd2 ✓ DNI válido → reenviado: 60000962
service-bd2 ✓ DNI válido → reenviado: 60000964
service-bd2 ✓ DNI válido → reenviado: 60000965
service-bd2 ✓ DNI válido → reenviado: 60000966
service-bd2 ✓ DNI válido → reenviado: 60000967
service-bd2 ✓ DNI válido → reenviado: 60000970
service-bd2 ✗ DNI no existe: 60000973

```

Figura 8: Validar Usuarios

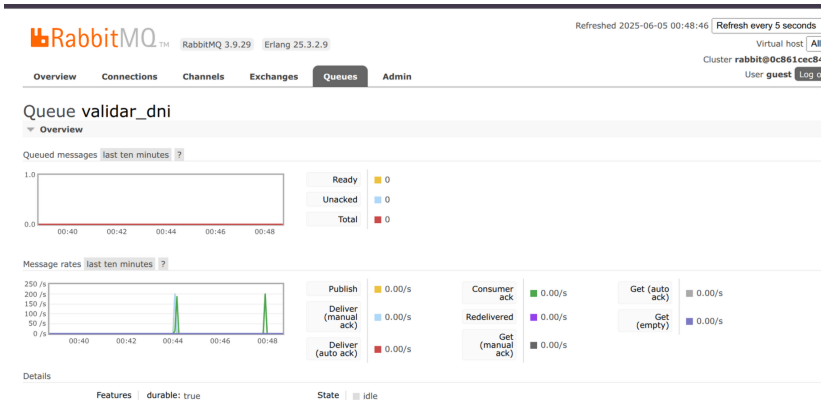


Figura 9: Queue validated DNI

- 1 Introduction
- 2 System Architecture
- 3 Communication Flow: RabbitMQ
- 4 Implementation Details
- 5 Docker and Docker Compose
- 6 Testing and Evaluation
- 7 Conclusions**

7. Conclusions

- System successfully demonstrates middleware-based communication and asynchronous processing.
- Validation of real-world scenarios: data integrity, messaging, multi-threaded persistence.
- RabbitMQ fundamental for decoupling, scalability, and fault tolerance.
- Docker and Docker Compose simplified development, deployment, and management.
- Practical and robust application of concurrent programming and distributed systems concepts.