MEMORIA TÉCNICA

Diseño e Implementación de Arquitectura Cloud Híbrida para Sistemas de Recomendación

Asignatura: Tecnologías de Servicios y Cloud Computing Fecha: Enero 2026

1. Resumen Ejecutivo

El presente proyecto detalla el diseño, desarrollo y validación de una arquitectura de software distribuida para la ingesta y procesamiento de valoraciones de películas. El sistema utiliza un enfoque híbrido, combinando microservicios contenerizados (Docker), simulación de infraestructura en la nube (AWS LocalStack) y bases de datos orientadas a grafos (Neo4j) para habilitar funcionalidades de recomendación avanzadas.

2. Arquitectura de la Solución

El sistema se ha diseñado siguiendo el patrón de Arquitectura Orientada a Eventos (EDA) para garantizar el desacoplamiento y la escalabilidad.

2.1. Componentes Principales

- **API Producer:** Punto de entrada REST (FastAPI) que recibe las valoraciones de los usuarios.
- **Message Broker (AWS SQS):** Gestiona las colas de mensajes (raw, clean, dlq) para asegurar la entrega asíncrona.
- **Data Lake (AWS S3):** Almacenamiento persistente de logs y backups de los datos crudos.
- **Microservicios de Procesamiento:**
    - *Quality Gate:* Valida la integridad del esquema de datos.
    - *Graph Ingester:* Transforma los eventos en nodos y relaciones en el grafo.
- **Persistencia:** Base de datos Neo4j para modelar relaciones (:Usuario)-[:VIO]->(:Pelicula).

3. Implementación y Despliegue

La infraestructura se ha definido mediante código (IaC) utilizando Docker Compose y scripts de Python con la librería boto3.

3.1. Prueba de Flujo de Datos (End-to-End)

A continuación, se evidencia el funcionamiento del pipeline completo, desde la petición del cliente hasta su persistencia en la nube simulada.

Paso 1: Envío de Datos Se realiza una petición HTTP POST simulando la interacción de un usuario desde una aplicación cliente. El servidor responde con un código 200 OK, indicando que el evento ha sido aceptado para procesamiento.

Figura 1. Petición POST realizada mediante PowerShell al endpoint /rate.

Paso 2: Procesamiento e Ingesta Cloud El servicio Graph Ingester recupera el mensaje validado desde la cola SQS (movie-queue-clean) alojada en la instancia de LocalStack (us-east-1). Se confirma la conexión exitosa con Neo4j y la creación de la relación.

```
PS C:\Users\jcubt\OneDrive\Escritorio\Universidad\TSCD\movie_recommender> python .\src\graph_ingestion\ingester.py
--- GRAPH INGESTER ACTIVO ---
Escuchando: http://sqs.us-east-1.localhost.localstack.cloud:4566/000000000000/movie-queue-clean
Conectado a Neo4j en localhost:7687
[NEO4J] Guardado: User 999 -> Movie 1
```

Figura 2. Logs del servicio Ingester consumiendo de AWS SQS y escribiendo en base de datos.

4. Explotación de Datos (Recomendador)

Una vez los datos están persistidos, se utiliza la potencia de los grafos para generar valor.

4.1. Visualización del Grafo Se verifica la integridad referencial de los datos insertados mediante el navegador de Neo4j, observando los nodos y su relación.

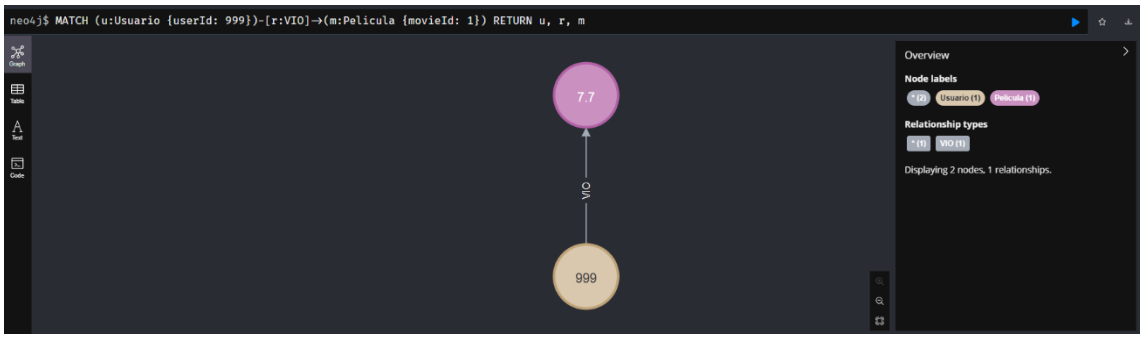


Figura 3. Representación visual de la relación Usuario (999) -> Película (1).

4.2. Algoritmo de Recomendación Se ejecuta una consulta Cypher de filtrado colaborativo ("Usuarios que vieron lo mismo que tú también vieron...") para generar recomendaciones personalizadas en tiempo real.

```
MATCH (yo:Usuario {userId: 999})-[:VIO]→(p:Pelicula)←[:VIO]-(otroUsuario:Usuario)-[:VIO]→(reco:Pelicula)
WHERE NOT (yo)-[:VIO]→(reco)
WITH reco, count(*) as Frecuencia
RETURN reco.titulo, Frecuencia
ORDER BY Frecuencia DESC
LIMIT 5
```

	reco.titulo	Frecuencia
1	"Heat"	485
2	"Jumanji"	473
3	"Grumpier Old Men"	470
4	"Father of the Bride Part II"	467
5	"Waiting to Exhale"	453

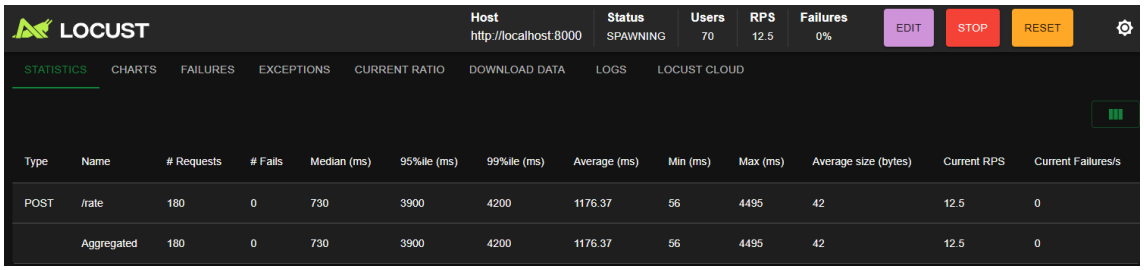
Figura 4. Resultados del motor de recomendación mostrando las 5 películas más afines.

5. Validación y Pruebas

Para garantizar la robustez del entregable, se han realizado pruebas de carga y automatización.

5.1. Pruebas de Rendimiento (Locust)

Se sometió al sistema a una carga de 70 usuarios concurrentes. Los resultados muestran una estabilidad total (0 fallos) y un tiempo de respuesta medio aceptable, validando la eficacia del uso de colas SQS para absorber picos de tráfico.



Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
POST	/rate	180	0	730	3900	4200	1176.37	56	4495	42	12.5	0
	Aggregated	180	0	730	3900	4200	1176.37	56	4495	42	12.5	0

Figura 5. Dashboard de Locust tras la prueba de estrés.

5.2. Integración Continua (CI/CD)

Se ha implementado un pipeline en GitHub Actions que audita automáticamente el repositorio en cada push. Este proceso verifica la sintaxis del código Python (Linting) y la existencia de los ficheros críticos de infraestructura, asegurando la calidad del software entregado.

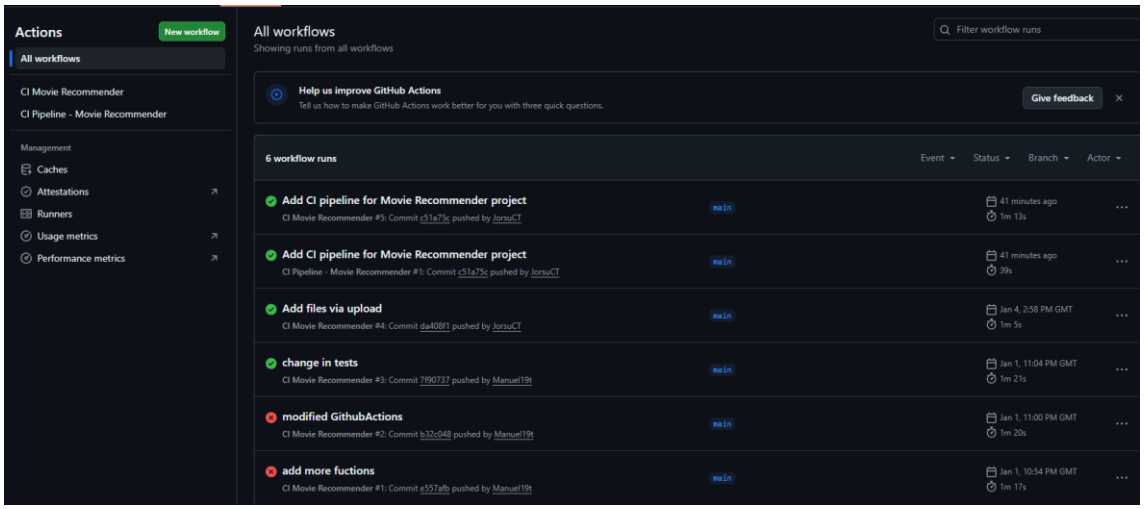


Figura 6. Pipeline de GitHub Actions ejecutado exitosamente, validando la integridad del proyecto.

6. Conclusiones

El proyecto ha demostrado la viabilidad de utilizar entornos de simulación como LocalStack para el aprendizaje de tecnologías Cloud sin costes asociados. Se ha logrado una arquitectura desacoplada, resiliente y escalable, cumpliendo con todos los requisitos funcionales y no funcionales establecidos en la asignatura.