Simulación de Diseño y Operación de un Cluster Cassandra para una App de Streaming Global

---

📌 Contexto

En entornos distribuidos a gran escala, como los de aplicaciones de streaming global, el correcto diseño de bases de datos NoSQL como Apache Cassandra es fundamental para garantizar disponibilidad, tolerancia a fallos y escalabilidad horizontal. Esta actividad busca aplicar conceptos clave como replicación, particionamiento, organización del esquema de datos y ejecución de consultas optimizadas para un modelo de lectura intensiva.

🎯 Objetivo

Diseñar un esquema básico de un **Cluster Cassandra** para una app de streaming global, incluyendo:

- Definición de keyspaces y tablas.
- Estrategia de replicación por regiones.
- Diseño de nodos y centros de datos distribuidos.
- Consultas CQL optimizadas.
- Justificación del entorno de simulación utilizado (Docker).

⚙️ Diseño del Cluster

1. 🌍 Distribución Regional

Región	Centro de Datos (DC)	Nodos
América	DC-America	3
Europa	DC-Europa	3

Cada centro de datos simula un conjunto de nodos dentro de una región geográfica, permitiendo distribuir la carga de lectura/escritura y garantizar alta disponibilidad incluso ante fallas regionales.

---

2. 🧬 Estrategia de Replicación

Se utiliza la estrategia `NetworkTopologyStrategy`, ideal para clústeres distribuidos geográficamente. Define cuántas copias de los datos se replican por región.

```
CREATE KEYSPACE streaming_analytics
WITH replication = {
  'class': 'NetworkTopologyStrategy',
  'DC-America': 2,
  'DC-Europa': 2
};
```

> Esta configuración permite una tolerancia a fallos regional (dos nodos pueden fallar sin pérdida de datos).

---

3. 📂 Estructura del Esquema

Keyspace: `streaming_analytics`

Tabla principal: `content_views`

```
CREATE TABLE content_views (
  user_id UUID,
  episode_id UUID,
  view_timestamp TIMESTAMP,
  PRIMARY KEY (user_id, view_timestamp)
) WITH CLUSTERING ORDER BY (view_timestamp DESC);
```

- **Partition Key:** `user_id` → Agrupa por usuario.
- **Clustering Key:** `view_timestamp` → Ordena visualizaciones por fecha.
- Optimizado para consultas recientes por usuario.

🔍 Consultas CQL Optimizadas

Consulta 1: Últimas visualizaciones de un usuario

```
SELECT episode_id, view_timestamp
FROM content_views
WHERE user_id = 7fc4b2c0-13a4-11ee-be56-0242ac120002
LIMIT 10;
```

> Devuelve los últimos 10 episodios vistos por un usuario específico.

Consulta 2: Visualizaciones por día (requiere materialized view o preprocesamiento)

Una forma eficiente de manejar esto en Cassandra es diseñar una tabla adicional:

```
CREATE TABLE daily_views (
  view_date DATE,
  episode_id UUID,
  user_id UUID,
  PRIMARY KEY (view_date, episode_id)
);
```

Consulta:

```
SELECT COUNT(*)
FROM daily_views
WHERE view_date = '2025-07-28';
```

Devuelve la cantidad de visualizaciones por episodio en una fecha determinada (útil para analítica diaria).

🧪 Entorno de Simulación

📦 Herramientas Utilizadas

- **Docker + Docker Compose** para ejecutar Cassandra localmente.
- **Cassandra Web UI** expuesta en `http://localhost:8000` para pruebas visuales.
- **CLI de Cassandra (cqlsh)** para ejecutar consultas en consola.

🧱 Estructura de Docker

```
services:
  cassandra:
    image: cassandra:latest
    ports:
      - "9042:9042"
  cassandra-web:
    image: vishnubob/cassandra-web
    ports:
      - "8000:3000"
```

Permite levantar un entorno local realista y funcional para practicar consultas distribuidas y diseño de esquemas.

🧠 Justificación del Diseño

- **Escalabilidad Horizontal:** Al ser Cassandra una base NoSQL distribuida, permite escalar agregando nodos sin afectar el rendimiento.

- **Tolerancia a Fallos:** La replicación por datacenter garantiza continuidad operativa ante fallas de nodos o incluso de regiones completas.

- **Lecturas Eficientes:** Las claves primarias y clustering keys están orientadas a los patrones más frecuentes: "lo último que vio un usuario", "cuántas veces se vio un contenido", etc.

- **Modelo Realista:** Este tipo de diseño replica las necesidades técnicas de plataformas como Netflix, Spotify o Amazon Prime.

✅ Conclusión

El ejercicio permitió simular con éxito la arquitectura de una base de datos distribuida orientada a lectura intensiva, usando Cassandra como motor principal y Docker como entorno de pruebas. El modelo diseñado es robusto, escalable y preparado para despliegue global, con replicación inteligente y consultas eficientes.