Modelo 4+1

Vista Física (Robustez y Despliegue)

Diagrama de Robustez

Componente Boundary

Propósito: Actúa como punto de entrada único al sistema, gestionando las comunicaciones bidireccionales entre múltiples clientes concurrentes y los servicios internos de procesamiento.

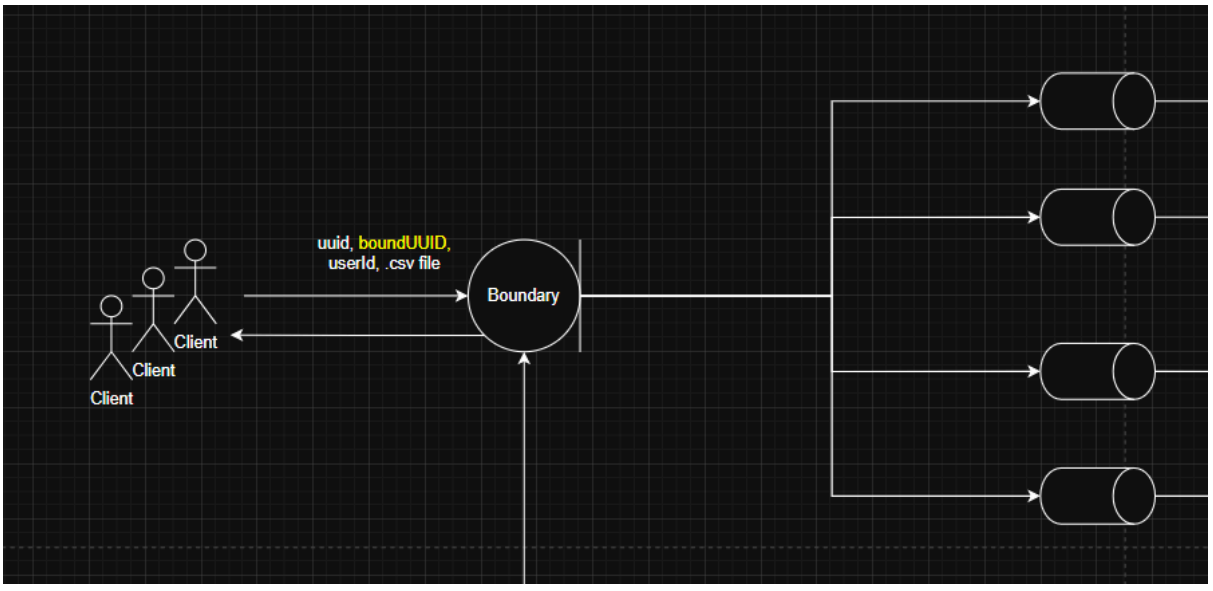
Responsabilidades principales:

- Aceptación y gestión del ciclo de vida de conexiones de clientes
- Clasificación y enrutamiento de peticiones hacia las colas de procesamiento correspondientes
- Distribución de respuestas del sistema hacia los clientes apropiados
- Mantenimiento de la continuidad de sesión mediante persistencia de identificadores únicos

Arquitectura interna: El Boundary implementa un diseño multihilo para manejar la concurrencia de manera eficiente:

- **Hilo Aceptador:** Se encarga exclusivamente de aceptar nuevas conexiones de clientes y asignarles recursos
- **Hilo Distribuidor de Respuestas:** Recibe las respuestas procesadas del sistema interno y las enruta al cliente correspondiente según su identificador
- **Hilos por Cliente:** Cada cliente activo tiene un hilo dedicado que procesa sus peticiones de manera independiente, clasificándolas por tipo (películas, créditos, reviews) y enviándolas a la cola RabbitMQ apropiada

Módulo de Persistencia: Utiliza un Write Ahead Log para mantener el mapeo entre direcciones IP de clientes y sus UUIDs únicos, garantizando la recuperación de estado ante fallos del sistema.



Componentes Router

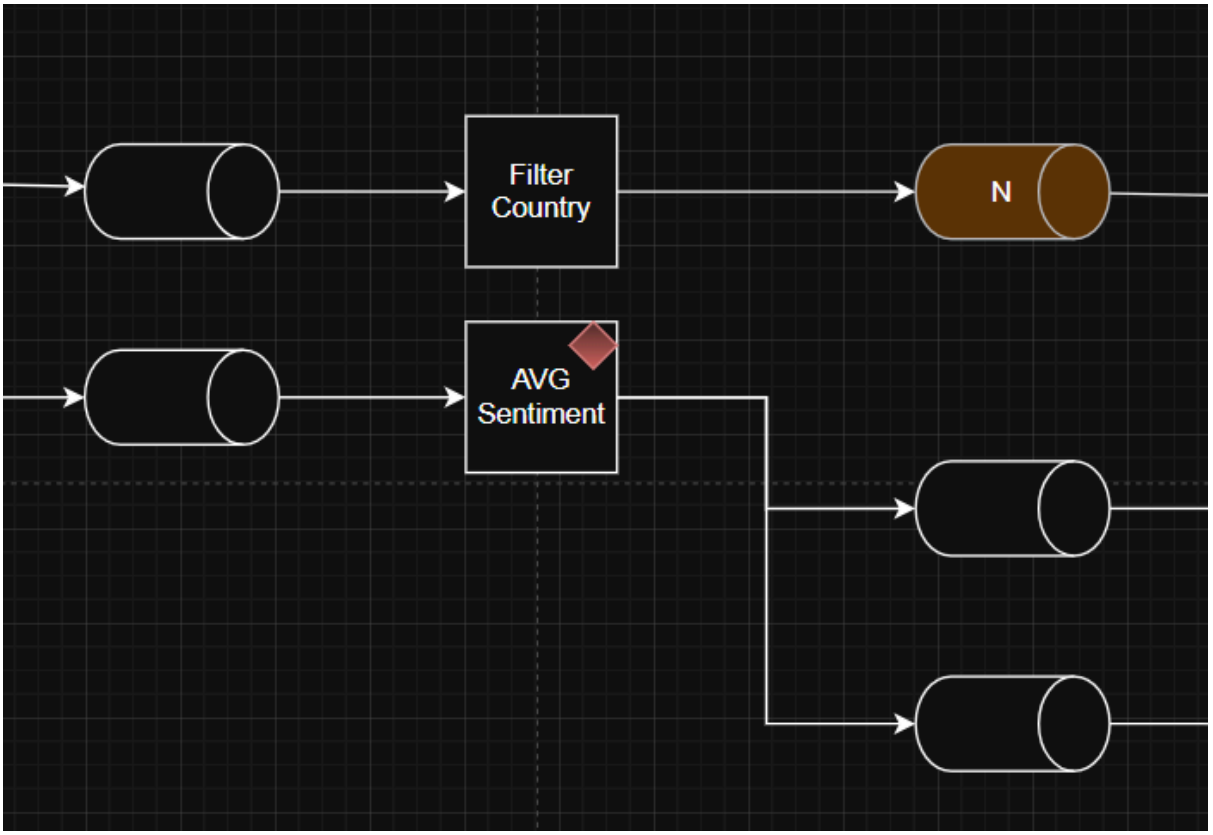
Propósito: Actúan como intermediarios inteligentes entre todos los componentes del sistema, implementando estrategias de distribución de carga y enrutamiento de mensajes.

Responsabilidades principales:

- **Load Balancing:** Distribuyen la carga de trabajo entre los distintos nodos para optimizar el rendimiento y evitar sobrecarga
- **Enrutamiento Inteligente:** Redirigen mensajes hacia el middleware apropiado según la configuración que tenga
- **Sharding Condicional:** Aplican funciones de particionamiento cuando es deseado distribuir datos específicos hacia los nodos correspondientes

Interacciones:

- Consumen mensajes de colas RabbitMQ específicas
- Redirigen los mensajes a las colas de los nodos correspondientes



Componentes Worker/Nodos de Procesamiento

Propósito: Se encargan del procesamiento de datos en cada pipeline filtrando y procesando la información recibida.

Responsabilidades principales:

- **Procesamiento de información:** Ejecutan la lógica específica según el tipo de Worker que sea
- **Persistencia de datos (tolerancia a fallos):** Almacenan datos en disco para su recuperación en caso de fallos del sistema
- **Filtro de duplicados:** No se “reprocesa” información previamente procesada.

Interacciones:

- Reciben datos desde los Routers correspondientes
- Envían resultados procesados hacia el siguiente worker del pipeline
- Son independientes entre sí y no tienen necesidad de coordinar entre ellos

Workers Stateless

Características:

- No mantienen estado interno entre peticiones
- Cada procesamiento es independiente y autocontenido
- Escalables y tolerantes a fallos
- Pueden ser reiniciados sin pérdida de información

Workers Stateful

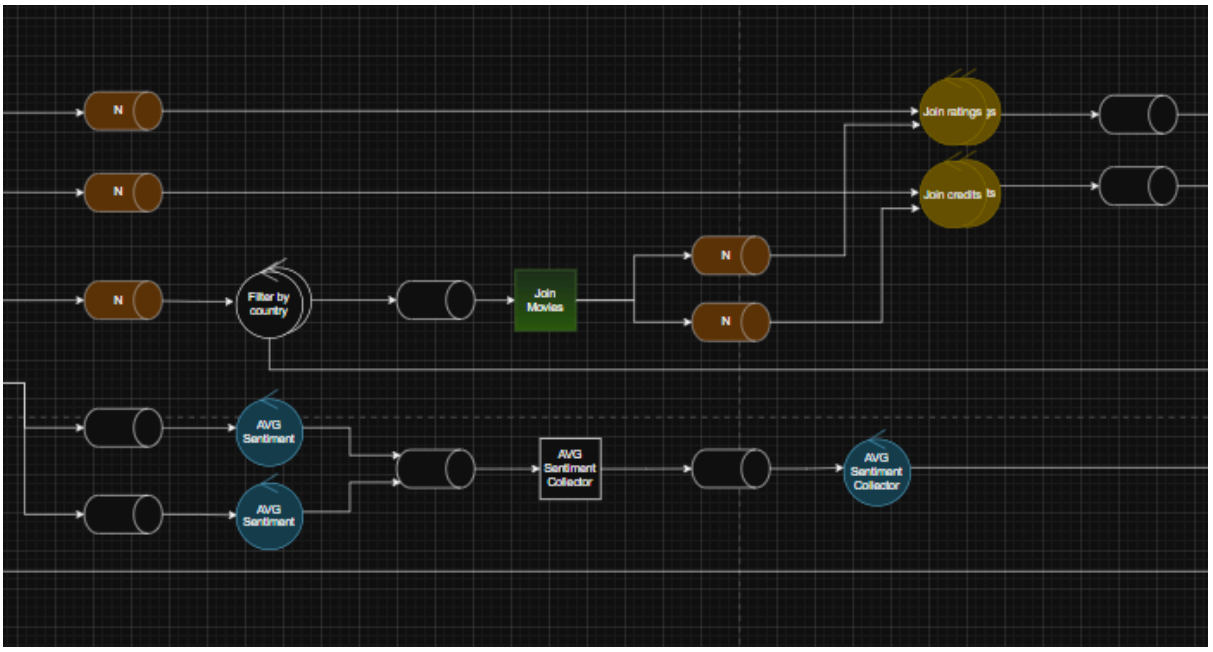
Características:

- Mantienen estado interno durante el procesamiento
- Requieren persistencia de estado para continuidad
- Requiere de un Intérprete de estado para la información a persistir

Workers JOIN (Caso Especial Stateful)

Características Únicas:

- **Consumo Dual:** Son los únicos workers que consumen concurrentemente de DOS colas diferentes
- **Persistencia Selectiva:** Implementan estrategia de persistencia únicamente de la Queue de Movies
- **Procesamiento Correlacionado:** Hacen un “JOIN” entre las movies persistidas y los credits/ratings que popean de la “segunda” queue de la cual consumen



Componentes Sentinel

Propósito: Constituyen el sistema de supervisión y resiliencia, garantizando la disponibilidad continua de los Workers mediante monitoreo activo y recuperación automática ante fallos.

Responsabilidades principales:

- **Monitoreo de Salud:** Supervisan continuamente el estado de todos los workers del sistema
- **Recuperación Automática:** Detectan y reviven workers caídos para mantener la disponibilidad del servicio
- **Réplicas:** Se implementó una estrategia del tipo Maestro-Esclavo con una reelección de líder del estilo Bully

Mecanismo de Supervisión:

- **Comunicación TCP:** Establecen conexiones TCP persistentes con cada worker para verificar su estado
- **Health Checks:** Realizan verificaciones periódicas de disponibilidad y capacidad de respuesta

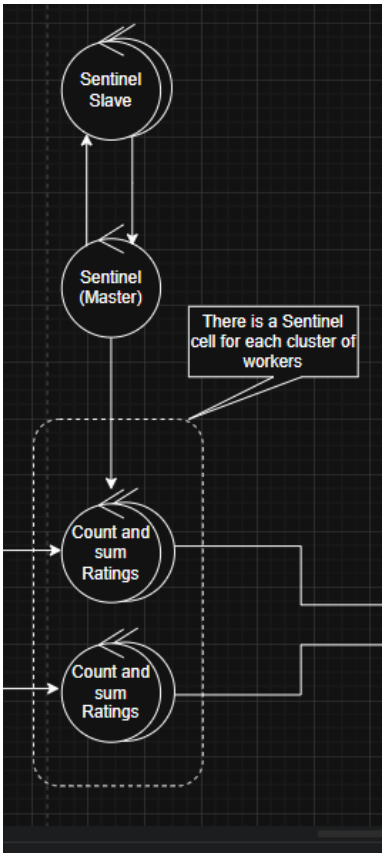
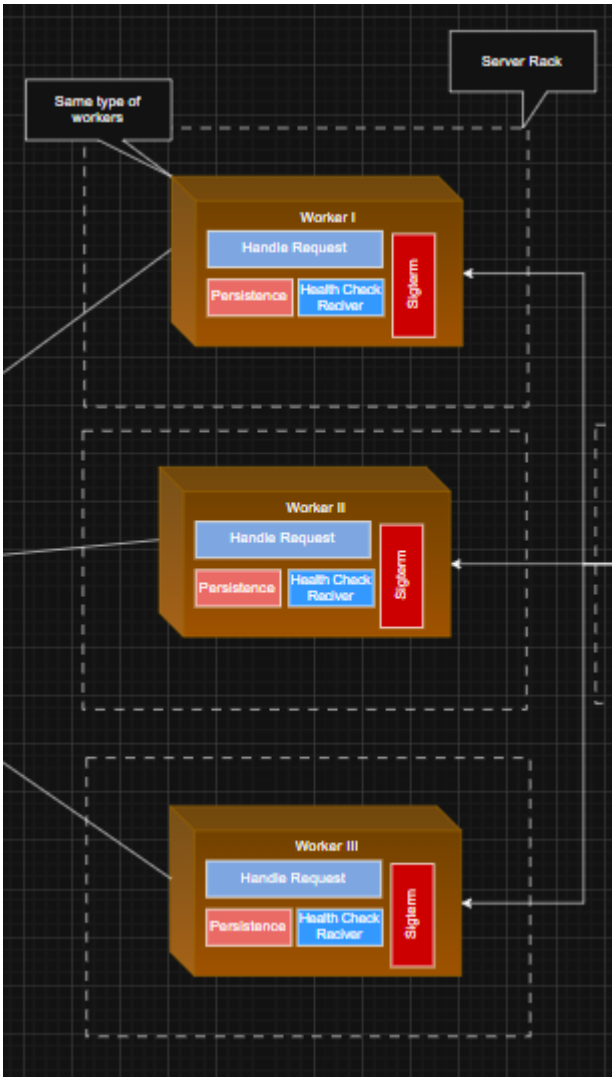
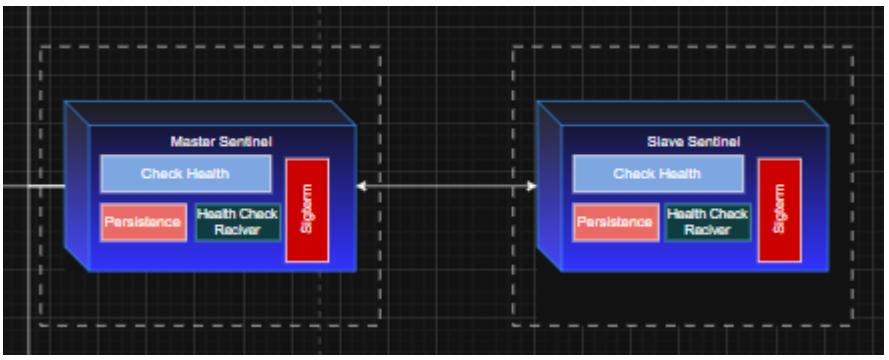


Diagrama de Despliegue

El sistema no tiene ningún requerimiento en términos de despliegue, todo componente es independiente de los demás y no necesitan coordinar para el correcto funcionamiento del pipeline, si es recomendable instanciar distintos worker del mismo tipo en distintos servidores físicos, asegurando que si ocurre una falla externa a nivel hardware, impidiéndole al centinela designado a dicho grupo de workers revivirlos,no se frene el procesamiento de información.



Siguiendo la misma idea, una celda de centinelas (el maestro y sus esclavos) deberían de estar en distintas computadoras físicas por la misma razón.



Vista de Procesos

DAG

Flujos de Procesamiento Principal

El sistema implementa **cuatro flujos principales de procesamiento** que pueden ejecutarse de manera concurrente y están optimizados para diferentes tipos de consultas:

Query 5: Análisis de Sentimientos

Flujo de Procesamiento:

- **Entrada:** Datos de Movies con información de reviews
- **Clasificación:** Los datos se clasifican automáticamente en positivos o negativos según algoritmos de análisis de sentimientos
- **Agrupación:** Se agrupan por categoría de sentimiento (POSITIVE/NEGATIVE)
- **Agregación:** Se calculan métricas estadísticas (promedio, ratio, conteo) por cada grupo de sentimiento
- **Concurrencia:** Procesamiento paralelo de múltiples reviews simultáneamente

Query 1: Filtrado Geográfico y Temporal

Flujo de Procesamiento:

- **Entrada:** Datos de Movies con metadatos completos
- **Filtrado Secuencial:**
    - Primer filtro: Películas posteriores al año 2000
    - Segundo filtro: Películas de países específicos (ARG o ESP)
- **Agrupación:** Organización por género cinematográfico
- **Concurrencia:** Filtros aplicados en paralelo cuando es posible, agrupación posterior sincronizada

Query 4: Análisis de Participación de Actores

Flujo de Procesamiento:

- **Entrada Dual:** Correlación concurrente de datos de Credits y Movies
- **Join Operación:** Workers JOIN especializados correlacionan actores con películas
- **Conteo de Participaciones:** Cálculo del número de películas por actor
- **Ranking:** Ordenamiento por cantidad de participaciones
- **Concurrencia:** Procesamiento paralelo de múltiples actores, sincronización en fase de ranking

Query 3: Correlación Movies-Ratings con Agregación Avanzada

Flujo de Procesamiento:

- **Entrada Dual:** Procesamiento concurrente de Movies y Ratings
- **Join Correlacionado:** Workers JOIN estatales correlacionan películas con sus calificaciones
- **Agregación Compleja:** Cálculo de métricas estadísticas avanzadas (promedios, máximos, mínimos)
- **Concurrencia:** Múltiples workers JOIN procesan diferentes segmentos de datos simultáneamente

Vista Lógica

UML de clases relacionadas en la persistencia (UML en Readme del repo)

Componentes de Almacenamiento y Persistencia

Sistema de Persistencia (Write-Ahead Log)

Propósito: Proporciona durabilidad y recuperación de datos mediante un mecanismo de logging robusto que garantiza la integridad de las operaciones.

Responsabilidades principales:

- **Persistencia Confiable:** Implementa protocolo de commit en dos fases (PROCESSING → COMPLETED)
- **Deduplicación de Mensajes:** Previene procesamiento duplicado mediante tracking de IDs incrementales por cliente y nodo
- **Recuperación Automática:** Restaura estado consistente tras fallos del sistema
- **Checkpointing:** Consolida logs individuales en puntos de control para optimizar almacenamiento

Arquitectura Modular:

Cada worker instancia una clase que implementa la interfaz “DataPersistenceInterface”, la cual tiene 3 métodos publicos persist, retrieve y clear, a su vez la “DataPersistenceInterface” utiliza una clase que implementa la interfaz “StateInterpreterInterface” la cual tiene 3 métodos publicos, format_data, parse_data, merge_data, de esta forma cada worker puede instanciar su propia clase que implemente “StateInterpreterInterface” de forma intercambiable, de esta forma la manera en que se persiste la data es facil de modificar para cada worker.

- **WriteAheadLog:** Gestiona registro, puntos de control y recuperación. Cumple con el contrato de la interfaz DataPersistenceInterface.
- **StateInterpreterInterface:** Desacopla lógica de negocio de mecánica de persistencia
- **StorageInterface:** Abstrae operaciones del sistema de archivos

Interacciones:

- Utilizada por todos los workers para persistir estado crítico y/o deduplicación de mensajes
- Integrada en el flujo de procesamiento de mensajes (POP → Verificación de duplicados → Procesamiento → Envío de datos → Persistencia → ACK)
- Coordina con sistema de colas para garantizar exactly-once processing

Sistema de Comunicación Asíncrona

RabbitMQ Message Broker

Propósito: Facilita comunicación asíncrona y desacoplada entre componentes del sistema.

Responsabilidades principales:

- **Enrutamiento de Mensajes:** Colas especializadas por tipo de datos
- **Garantía de Entrega:** Persistence y acknowledgments para operaciones críticas
- **Tolerancia a Fallos:** Message durability

Arquitectura de Colas:

- Colas de entrada por tipo de datos
- Colas de salida para resultados procesados
- Colas específicas y personales para cada worker

Patron Arquitectónico Principal Implementado

Event-Driven Architecture

- **Comunicación Asíncrona:** Eventos como mecanismo principal de integración
- **Loose Coupling:** Componentes independientes comunicados vía un message broker
- **Eventual Consistency:** Procesamiento distribuido con convergencia de estado

Separación de Responsabilidades

Capa de Presentación

- **Boundary:** Único punto de entrada, manejo de conexiones cliente
- **Protocolos de Comunicación:** Abstracción de detalles de red

Capa de Distribución

- **Routers:** Load balancing, sharding, enrutamiento inteligente
- **Message Broker (RabbitMQ):** Comunicación asíncrona, delivery guarantees

Capa de Procesamiento

- **Workers Stateless:** Agregaciones independientes de información pasada y futura
- **Workers Stateful:** Agregaciones acumulativas dependientes de un estado anterior

- **Workers JOIN:** Operaciones de correlación multi-fuente. Almacena movies y procesa credits/ratings según las movies que tiene persistidas

Capa de Almacenamiento

- **Write-Ahead Log:** Durabilidad, deduplicación, recuperación

Capa de Infraestructura y Resiliencia

- **Sentinels:** Monitoreo, recuperación automática, high availability
- **Algoritmo Bully:** Elección de líder distribuida, coordinación

Vista de Desarrollo

Diagrama de paquetes

Gateway Block

- Contiene el **Boundary** y **Protocol**
- Actúa como punto de entrada al sistema, manejando las comunicaciones externas y protocolos de red
- Utiliza el Serializer Block para procesar datos y el Communication block para enviar la información al resto del pipeline.

Communication Block

- Contiene **RabbitMQ** como sistema de mensajería
- Gestiona la comunicación asíncrona entre componentes del sistema

Persistence Block

- Incluye **WriteAheadLog** como mecanismo de persistencia, el cual utiliza interfaces de almacenamiento para desacoplar el manejo del file system. (**StorageInterface** y **StateInterpreterInterface**)

Serializer Block

- Contiene la clase **Serializer**
- Se encarga de la serialización/deserialización de datos

Worker Block

- Contiene la clase **Worker**
- Utiliza **RabbitMQClient** (para popear y pushear la data) y **SentinelBeacon** (para contar con resiliencia)
- Procesa la información del pipeline.

Resilience Block

- Incluye el **Sentinel**
- Proporciona capacidades de monitoreo y resilencia del sistema
- Supervisa el estado de los workers y otros componentes

Vista de Escenarios

Escenario Principal: Procesamiento de Query Completa

Descripción

Este escenario representa el flujo completo desde que un cliente envía una consulta hasta que recibe los resultados procesados, demostrando la interacción entre todos los componentes del sistema.

Actores

- **Cliente**: Usuario que envía queries de análisis de datos
- **Sistema de Procesamiento**: Conjunto de componentes (workers) que procesan la información

Flujo Principal

1. Iniciación de Conexión

Componentes involucrados: Boundary (Vista Física), Gateway Block (Vista de Desarrollo)

- El cliente establece conexión TCP con el Boundary
- El **Hilo Aceptador** del Boundary acepta la nueva conexión
- Se asigna un UUID único al cliente y se persiste usando nuestra clase Write-Ahead Log
- Se crea un **Hilo por Cliente** dedicado para manejar las peticiones

2. Recepción y Clasificación de Query

Componentes involucrados: Boundary, Serializer Block, Communication Block

- El cliente envía datos (movies, credits, reviews) al sistema
- El **Hilo por Cliente** recibe y clasifica la petición por tipo
- Se utiliza el **Serializer** para procesar los datos entrantes
- Los datos se enrutan a la cola RabbitMQ correspondiente

3. Distribución y Enrutamiento

Componentes involucrados: Routers (Vista Física), RabbitMQ (Vista Lógica)

- Los **Routers** consumen mensajes de las colas RabbitMQ específicas
- Aplican estrategias de **Load Balancing** para distribuir la carga
- Implementan **Sharding Condicional** cuando es necesario
- Redirigen mensajes hacia las colas de los workers apropiados

4. Procesamiento por Workers

Componentes involucrados: Worker Block (Vista de Desarrollo), Workers Stateless/Stateful (Vista Física)

Para Workers Stateless (ej: Query 1 - Filtrado):

- Pop de datos desde la cola asignada
- Verificación de duplicados usando Write-Ahead Log
- Procesamiento: filtrado por año (>2000) y país (ARG/ESP)
- Agrupación por género cinematográfico
- Push de resultados a la siguiente cola del pipeline

Para Workers Stateful (ej: Query 5 - Análisis de Sentimientos):

- Pop de datos manteniendo estado interno
- Clasificación de reviews en POSITIVE/NEGATIVE
- Agregación acumulativa de métricas estadísticas
- Push de resultados agregados
- Persistencia de estado usando StateInterpreterInterface

Ejemplo concreto con los workers JOIN (Query 3 y 4):

- **Consumo Dual**: Pop concurrente de DOS colas (movies + credits/ratings)
- **Persistencia Selectiva**: Almacenamiento solo de movies
- **Procesamiento Correlacionado**: JOIN entre movies persistidas y datos de la segunda cola

5. Monitoreo y Resiliencia

Componentes involucrados: Sentinels (Vista Física), Resilience Block (Vista de Desarrollo)

- Los **Sentinels** supervisan continuamente todos los workers vía TCP
- Realizan health checks periódicos
- En caso de fallo de worker, el Sentinel lo detecta y revive automáticamente
- El algoritmo **Bully** mantiene la elección de líder entre Sentinels

6. Persistencia y Deduplicación

Componentes involucrados: Write-Ahead Log (Vista Lógica), Persistence Block (Vista de Desarrollo)

- Cada mensaje procesado se registra con protocolo PROCESSING → COMPLETED
- Se previene procesamiento duplicado mediante tracking de IDs incrementales

- Checkpointing consolida logs para optimizar almacenamiento

7. Retorno de Resultados

Componentes involucrados: Boundary, Communication Block

- Los resultados finales llegan a colas de salida
- El **Hilo Distribuidor de Respuestas** del Boundary los consume
- Se identifica el cliente correspondiente usando el UUID persistido
- Los resultados se envían de vuelta al cliente apropiado

Escenarios de Desconexión Abrupta del Cliente

Componentes involucrados: Boundary, Routers, Workers (todos los tipos), RabbitMQ

Flujo de Limpieza Distribuida:

1. **Detección de Desconexión**:
    - El **Hilo por Cliente** en el Boundary detecta pérdida de conexión TCP
    - Se identifica el client_id (UUID) del cliente desconectado
2. **Generación de DISCONNECT_MARKER**:
    - El Boundary genera un mensaje especial `DISCONNECT_MARKER` con el client_id
    - Se propaga este marcador a TODAS las colas RabbitMQ del sistema
3. **Propagación en Cascada por Routers**:
    - Cada Router recibe el `DISCONNECT_MARKER`
    - Elimina cualquier mensaje pendiente asociado al client_id
    - Repropaga el `DISCONNECT_MARKER` a las colas downstream de sus workers asignados
4. **Limpieza en Workers Stateless**:
    - Procesan el `DISCONNECT_MARKER` de forma inmediata
    - Propagan el marcador a su cola de salida correspondiente
5. **Limpieza en Workers Stateful**:
    - Reciben el `DISCONNECT_MARKER` y eliminan estado persistido del client_id
    - Utilizan Write-Ahead Log para registrar la operación de limpieza
    - Propagan el marcador
6. **Idempotencia Garantizada**:
    - Si un nodo recibe múltiples `DISCONNECT_MARKER` para el mismo client_id, la operación se ejecuta sin efectos secundarios
7. **Finalización de Limpieza**:

- El Boundary elimina el mapeo client_id ↔ dirección IP del Write-Ahead Log
- Se libera el **Hilo por Cliente** asociado
- Sistema queda completamente limpio de rastros del cliente desconectado

Escenarios Extras

Escenario de Fallo de Worker

1. Sentinel detecta worker no responsivo via health check
2. Sentinel revive el worker automáticamente
3. Worker recupera estado desde Write-Ahead Log
4. Procesamiento continúa sin pérdida de datos

Escenario de Fallo de Sentinel

1. Slaves detectan fallo del Sentinel maestro
2. Se ejecuta algoritmo Bully para reelección
3. Nuevo Sentinel maestro asume responsabilidades
4. Continuidad de supervisión garantizada

Escenario de Sobrecarga del Sistema

1. Routers detectan alta carga en workers específicos
2. Load balancing redistribuye mensajes a workers menos cargados
3. Sistema mantiene throughput óptimo
4. No se pierde capacidad de procesamiento