FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico Tolerancia: Steam Analysis

Grupo 04

[75.74] Sistemas Distribuidos I

Segundo cuatrimestre de 2024

Grati, Lucas	102676	lgrati@fi.uba.ar
Adelsflügel, Valentina	108201	vadelsflugel@fi.uba.ar
Frenkel, Gastón Mariano	107718	gfrenkel@fi.uba.ar

Índice

Alcance

El proyecto Steam Analysis consiste en un sistema distribuido que analiza reviews de juegos y reseñas de la plataforma de videojuegos Steam a distintos títulos publicados. Las reviews poseen nombre del juego, texto del comentario y voto positivo o negativo. Por cada juego, se conoce género, fecha de publicación, etc. Se deben obtener 5 queries:

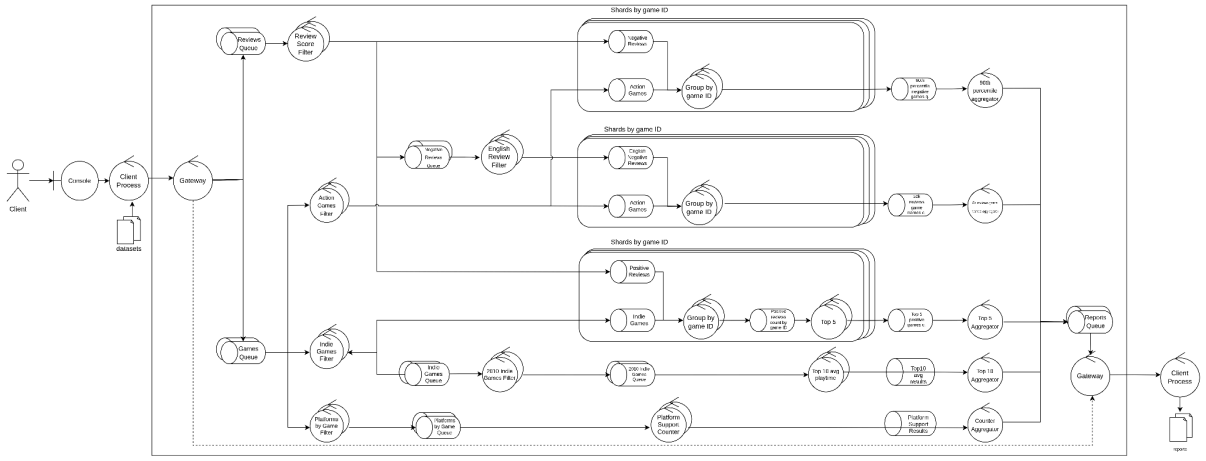
1. Cantidad de juegos soportados en cada plataforma (Windows, Linux, MAC).
2. Nombre de los juegos top 10 del género "Indie" publicados en la década del 2010 con más tiempo promedio histórico de juego.
3. Nombre de los juegos top 5 del género "Indie" con más reseñas positivas.
4. Nombre de juegos del género "shooter" con más de 50.000 reseñas positivas en idioma inglés.
5. Nombre de juegos del género "shooter" dentro del percentil 90 en cantidad de reseñas negativas.

Arquitectura

Vista Física

Se detallan en esta vista todos los componentes físicos del sistema así como las conexiones físicas entre esos componentes que conforman la solución.

Diagrama de Robustez



Este diagrama evidencia la escalabilidad de nuestro diseño:

- Podría ser necesario escalar cualquier nodo cuya carga escala con la cantidad de datos. Por eso, cada requerimiento de cada query tiene su respectivo nodo Filter. De esta manera, si la cantidad de juegos “shooter” de la query 4 y 5 fuera mucho mayor

y generara un cuello de botella, se podrían aumentar las réplicas de los nodos pertinentes.

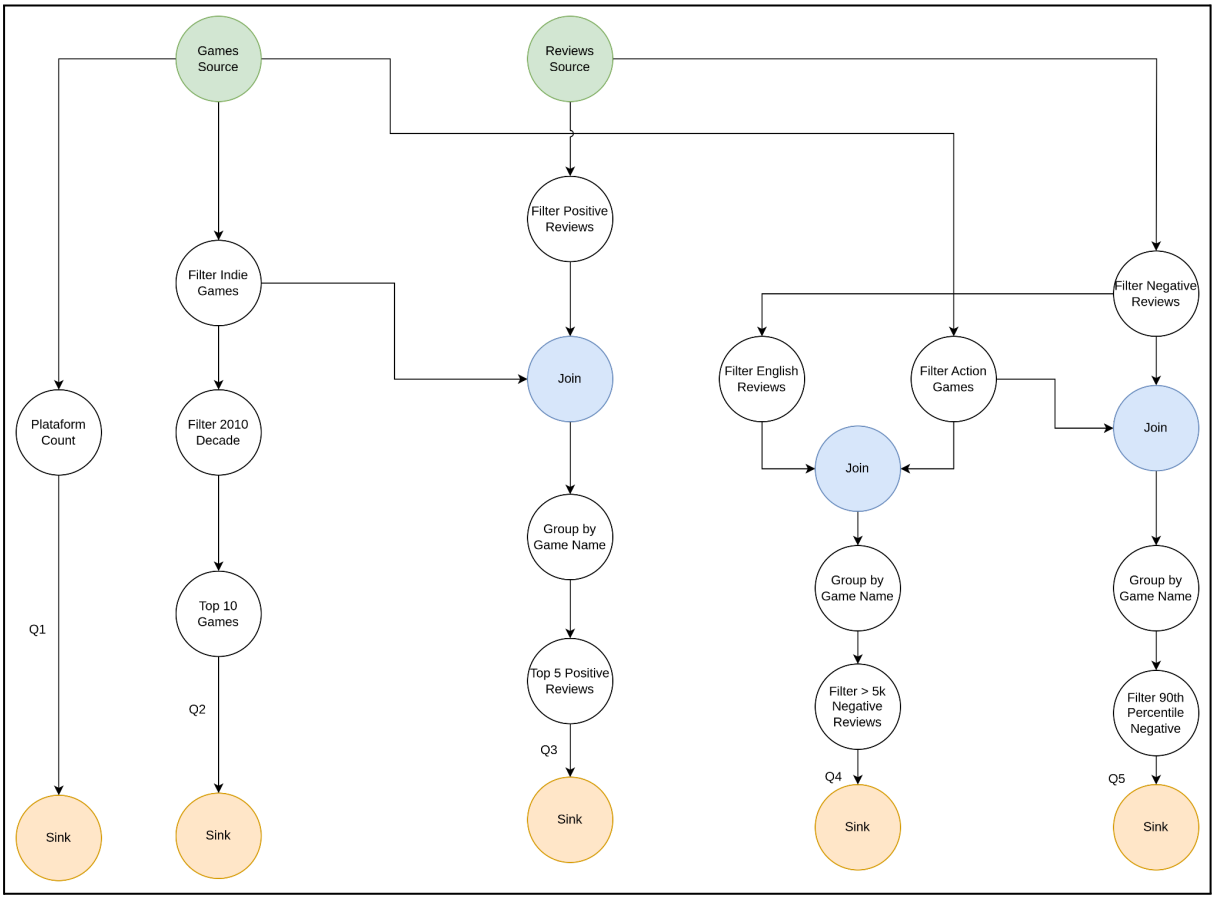
Por otro lado, con el objetivo de organizar la información y distribuír el procesamiento, se decidió utilizar *sharding* para joinear reseñas con juegos: cada registro se envía a un *shard* dependiendo de su Game ID. Esto asegura que todas las *reviews* de un juego se encuentren en el mismo nodo que el registro *juego* asociado a ellas. A su vez, permite escalar en caso de tener muchos datos, teniendo así shards más pequeños pero en mayor cantidad.

En nuestro sistema, los juegos y las reseñas llegan de forma mezclada, por lo que es necesario almacenar las reseñas hasta que se reciba el juego correspondiente. Dado que se cuenta con múltiples shards y joiners, la gestión de memoria no representa un problema, y no es necesario utilizar métodos de sincronización para coordinar este proceso.

Vista Lógica

Esta sección explica la estructura y funcionalidad del sistema. A continuación se muestra un directed acyclic graph que evidencia el flujo de datos:

DAG

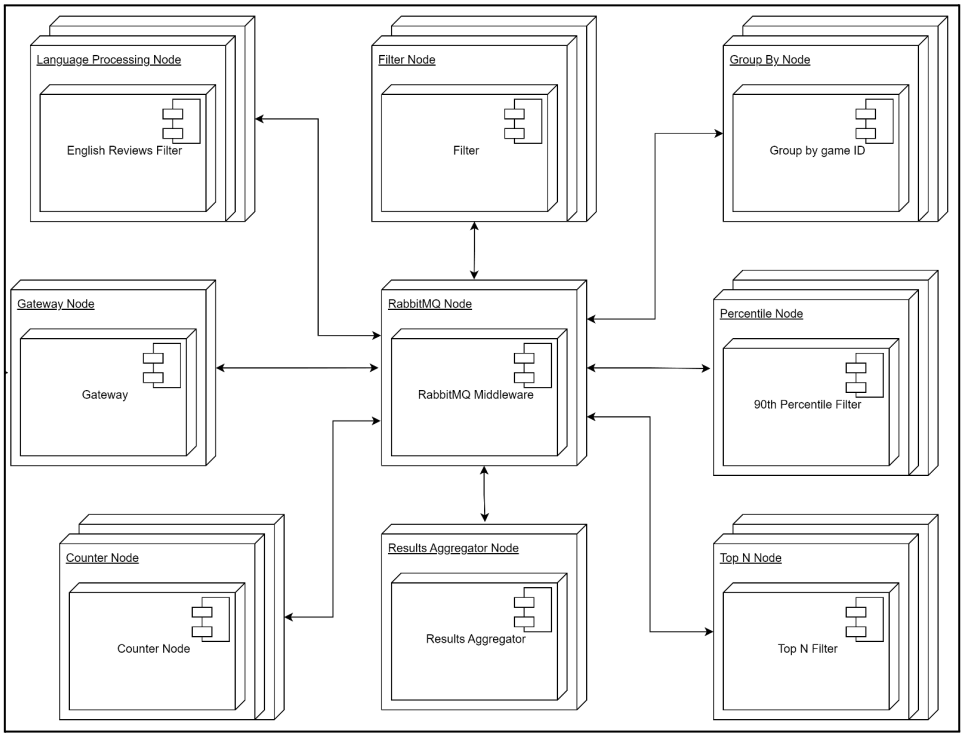


En este diagrama se puede observar varios componentes principales: nodos Filtros, nodos Group by y nodos Sink (de ahora en más Aggregators). La información fluye desde las fuentes (que serán Queues) pasando por diferentes filtros que permiten reducir la misma hasta obtener el resultado final para cada query. En algunos casos, es necesario joinear la información de una fuente con la de la otra. Acá es donde intervienen los nodos de agrupamiento.

Diagrama de Despliegue

En el diagrama se muestra la distribución de procesos en distintas computadoras. Todos los nodos se comunican a través del middleware RabbitMQ, excepto en el caso de la interacción entre el cliente y el gateway.

Para simplificar, varios nodos se agrupan en uno solo. Por ejemplo, el nodo Filter agrupa los procesos relacionados con el filtrado por género de juegos, juegos de la década de 2010 y reseñas en inglés, entre otros. De manera similar, el nodo Top N reúne todas las operaciones relacionadas con los rankings “Top”, y el nodo Results Aggregator centraliza las agregaciones de resultados.

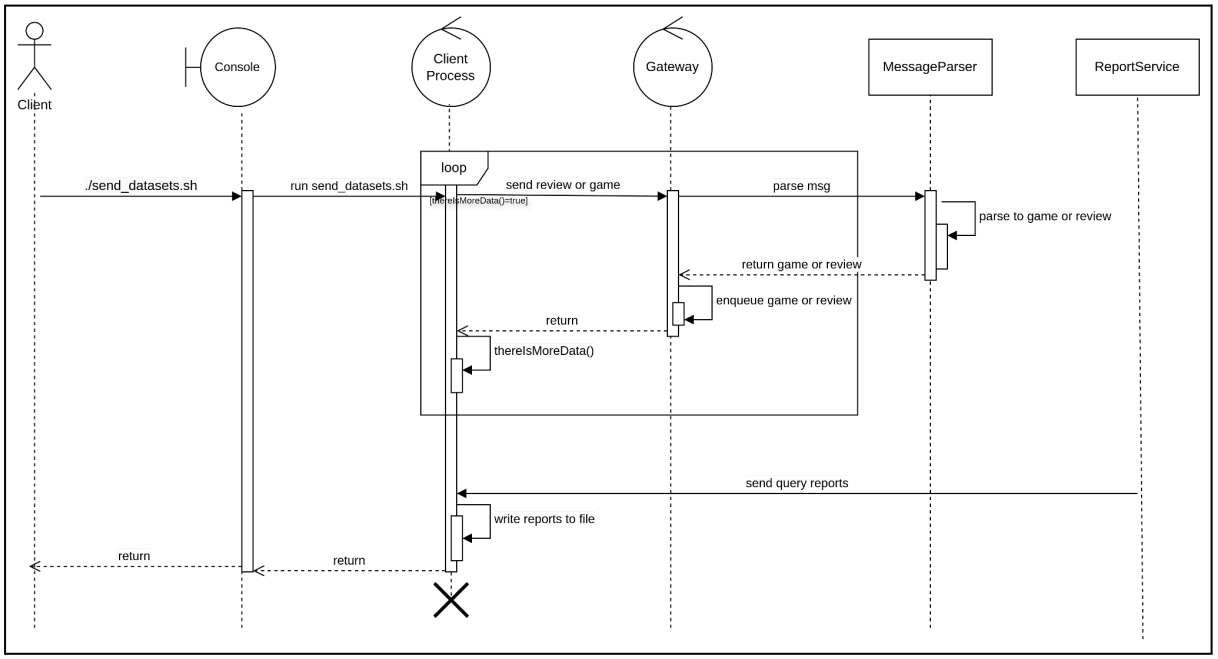


Vista de Procesos

Esta vista trata los aspectos dinámicos del sistema: los procesos de sistema y cómo se comunican. Se enfoca en el comportamiento del sistema en ejecución. La vista considera aspectos de concurrencia, distribución, escalabilidad, etc.

Se incluyen dos secciones de diagramas: secuencia y actividad. En ambos casos, se decidió diagramar sólo los casos relevantes de explicación.

Diagrama de secuencia

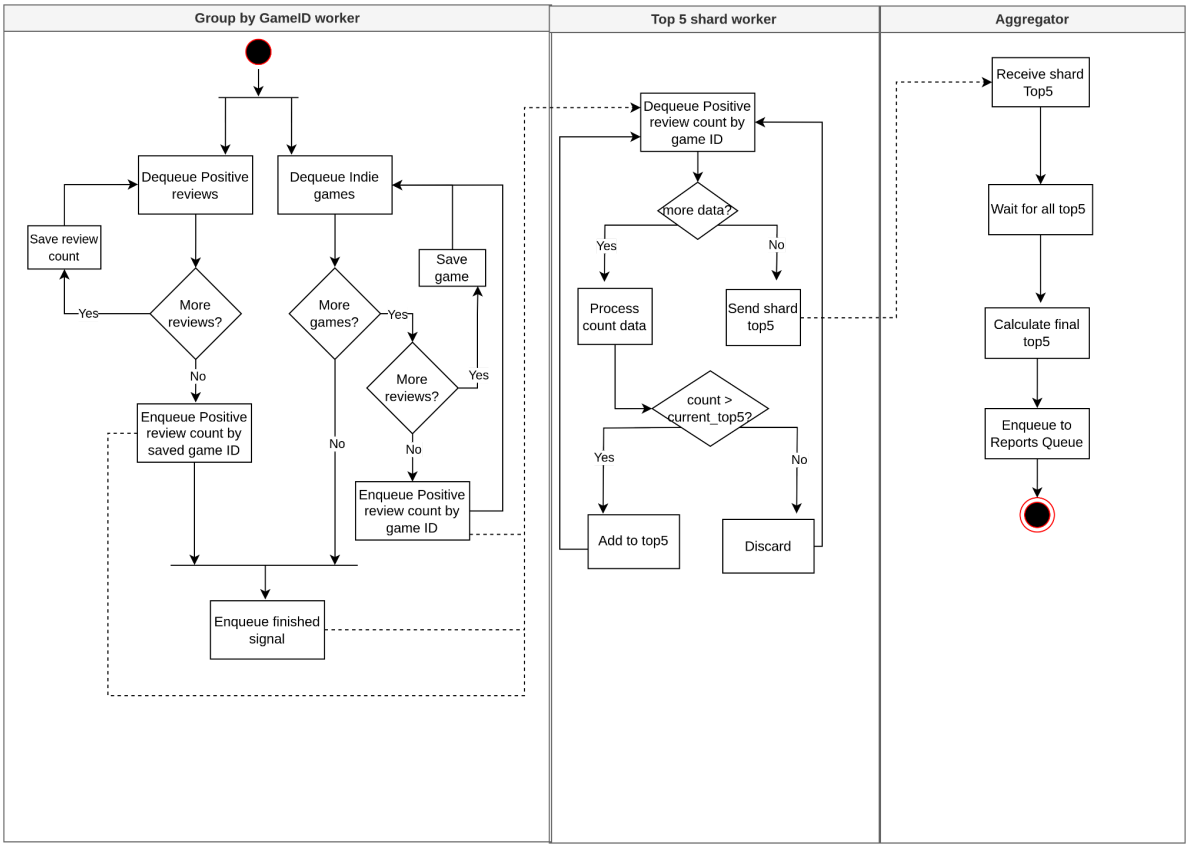


La intención del diagrama de secuencia adjunto es visualizar la relación entre el cliente y el sistema. Aspectos importantes:

- El cliente posee los datasets y los envía vía TCP iniciando un proceso *Client* en su computadora mediante un script.
- Este proceso *Client* “vive” hasta que termina de recibir todos los resultados.

  Notar que el *Gateway* persiste, cumpliendo la función de servidor.
- El *Gateway* recibe los mensajes y un *Message Parser* dentro del mismo se encarga de decodificarlos.
- Se omite en el diagrama todo el procesamiento intermedio para denotar que el *Report Service* envía los reportes a medida que los obtiene. Se mantiene la conexión con el cliente durante todo el procesamiento.

Diagrama de actividad



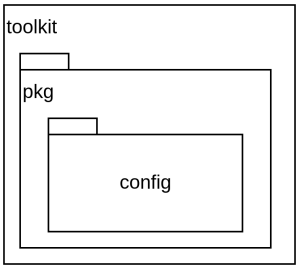
El caso presentado corresponde al procesamiento de los shards. Se puede ver cómo es el flujo del mismo y cómo, por ejemplo, el *Top 5 Worker* va obteniendo la data desencolando de las correspondientes colas que el joiner worker va acumulando.

Vista de desarrollo

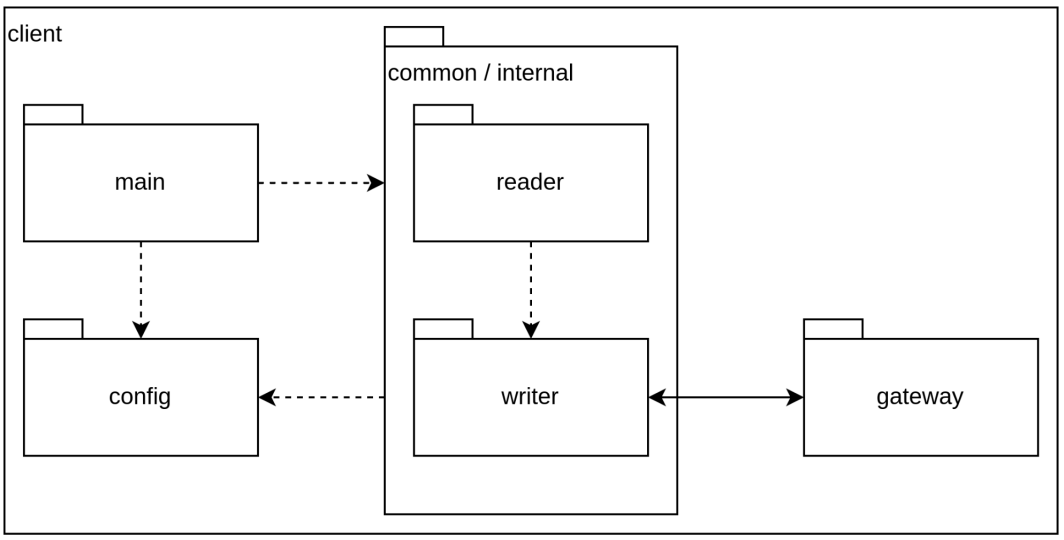
La vista de desarrollo se centra en cómo está estructurado el sistema desde la perspectiva del programador y cómo se gestiona el software. Esta vista muestra cómo se divide el sistema en módulos, componentes y subsistemas. A nivel de implementación, se refleja cómo los desarrolladores organizan, construyen y mantienen el código, destacando aspectos como la estructura de carpetas, paquetes, bibliotecas y la relación entre los diferentes módulos del software.

Diagrama de paquetes

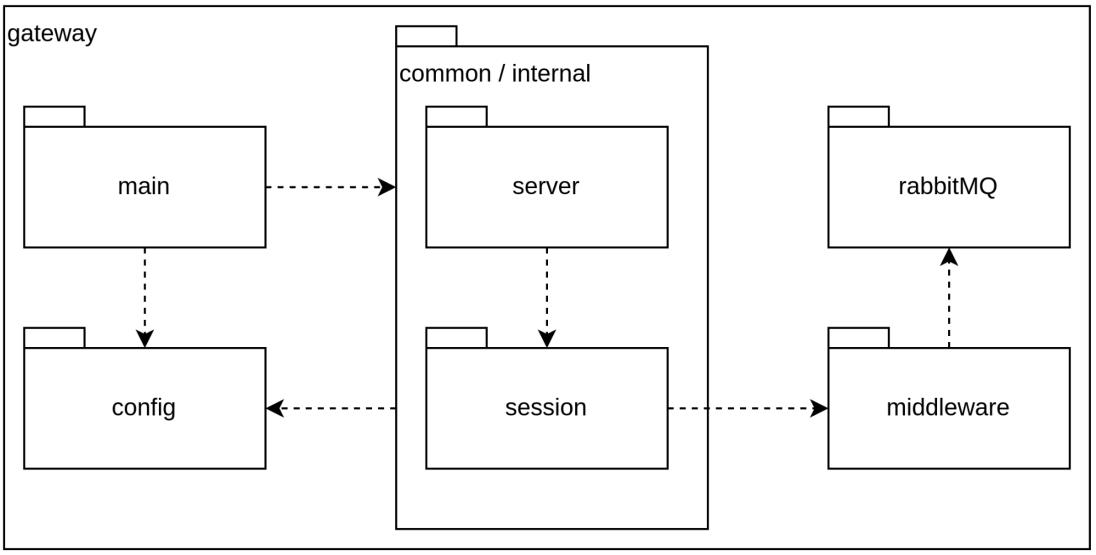
El diagrama de paquetes se encuentra dividido en distintos elementos lógicos en función de las responsabilidades de cada nodo.



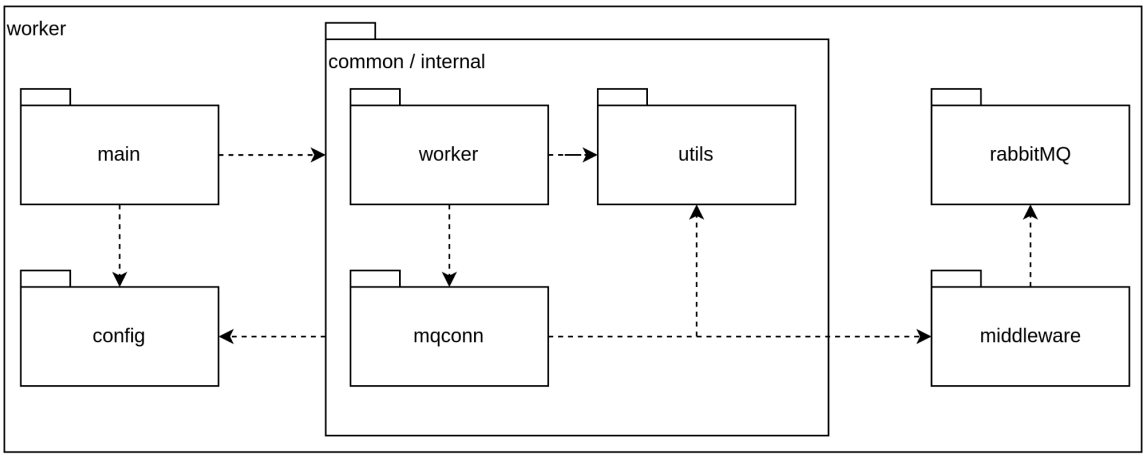
La toolkit es la responsable de almacenar el paquete de configuración que proveerá una interfaz común a todas las aplicaciones. De esta manera se evita la repitencia de código y se modulariza el comportamiento en un programa común.



El cliente es el responsable de leer los archivos a través del paquete reader y enviarlos al gateway a través de una conexión TCP haciendo uso del paquete writer. A su vez, estos paquetes se encontrarán dentro del paquete common (Python) o internal (Go).



El gateway del sistema posee dos paquetes. Por un lado el paquete server, que se encarga de recibir nuevas conexiones y gestionarlas, y por el otro, el paquete session que se encarga de la comunicación entre el gateway y el cliente, y entre el gateway y el middleware de rabbitMQ.



Cada uno de los workers y filtros contarán con la misma estructura de paquetes la cual cuenta con un paquete de utils para aquellas operaciones necesarias para el procesamiento del worker pero que pueden ser abstraídas. A su vez, el paquete worker se encargará del procesamiento de mensajes en función de la responsabilidad del mismo, mientras que mqconn hará de interfaz entre el worker y el middleware de RabbitMQ.

Escenarios

Esta vista es también conocida como vista de casos de uso. Describe cómo el sistema responde a situaciones o casos de uso específicos, que suelen representar interacciones clave entre los usuarios (o actores) y el sistema.

1. **Consultar cantidad de juegos soportados por plataforma:**
    - **Descripción:** Dado un cliente que posee el dataset de reviews y juegos, cuando ejecuta el sistema podrá consultar la cantidad de juegos soportados en cada plataforma (Windows, Linux, MAC).
2. **Obtener los top 10 juegos Indie de la década del 2010 por tiempo promedio de juego:**
    - **Descripción:** Dado un cliente que posee el dataset de reviews y juegos, cuando ejecuta el sistema podrá consultar los nombres de los 10 juegos del género "Indie" publicados en la década del 2010 con el mayor tiempo promedio histórico de juego.
3. **Obtener los top 5 juegos Indie con más reseñas positivas:**
    - **Descripción:** Dado un cliente que posee el dataset de reviews y juegos, cuando ejecuta el sistema podrá consultar los nombres de los 5 juegos del género "Indie" que tienen la mayor cantidad de reseñas positivas.

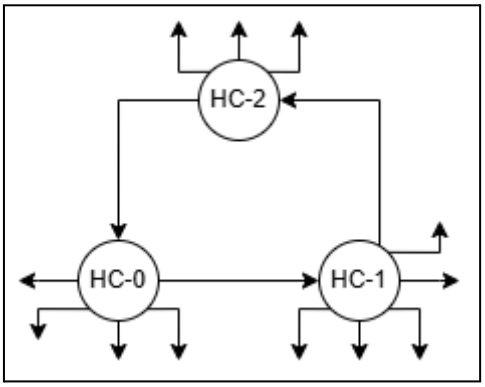
4. **Consultar juegos Shooter con más de 50.000 reseñas positivas en inglés:**
    - **Descripción:** Dado un cliente que posee el dataset de reviews y juegos, cuando ejecuta el sistema podrá consultar los nombres de los juegos del género "Shooter" que cuentan con más de 50.000 reseñas positivas en idioma inglés.
5. **Obtener juegos Shooter dentro del percentil 90 en reseñas negativas:**
    - **Descripción:** Dado un cliente que posee el dataset de reviews y juegos, cuando ejecuta el sistema podrá consultar los nombres de los juegos del género "Shooter" que se encuentran dentro del percentil 90 en cantidad de reseñas negativas.

Tolerancia a fallos

Health Check

El sistema incorpora nodos health-checkers, cuya responsabilidad principal es monitorear el estado de los demás nodos y, en caso de detectar una falla, reiniciar el nodo afectado para restaurar su funcionamiento.

Los health-checkers están organizados en una topología de anillo. En la configuración de cada health-checker, se especifican los nodos que debe supervisar y reactivar en caso de una caída. Si un health-checker también falla, es responsabilidad de su health-checker adyacente reiniciarlo.



El monitoreo se lleva a cabo mediante conexiones UDP hacia cada nodo. Utilizando parámetros configurables, un nodo se considera inactivo tras una cantidad predefinida de errores de timeout, lo que activa el proceso de recuperación para ponerlo en funcionamiento nuevamente.

Se ve una posible mejora en este diseño. Hubiera sido adecuado implementar un algoritmo de *leader election* para los health-checkers de manera que haya un líder que decida qué nodos tiene como responsables cada uno (en vez de definirlo por configuración) y que en caso de que todos se caigan, el líder los pueda re-levantar.

Duplicados

En el caso donde un nodo recibe un mensaje previamente procesado, este es descartado haciendo uso de un mapa que une un UUID del worker que envió el mensaje a un número de secuencia, el cual se corresponde con el próximo número de secuencia que se espera recibir.

Este filtro de duplicados es disponibilizado a través de una interfaz que le permite a los workers validar si un número de secuencia ya fue procesado, o actualizar el valor del próximo número de secuencia a procesar (necesario tanto durante la ejecución normal del worker como durante la recuperación).

Persistencia

Para garantizar la tolerancia a fallos, el sistema está diseñado para que un nodo que sufre una caída pueda recuperar su estado previo al reiniciarse. La persistencia de los datos es asegurada en todos los nodos mediante una interfaz que gestiona el almacenamiento de resultados, la restauración de números de secuencia para control de duplicados y la recuperación de archivos.

Durante una ejecución normal (fuera de la recuperación), un mensaje es procesado, persistido y ACKeado. Debido a que estas dos últimas operaciones no son atómicas, podría suceder que un mensaje sea persistido pero no ACKeado, lo cual provocaría un duplicado. Cabe notar que todos los mensajes que se encuentran persistidos en disco son no-duplicados y que ya fueron procesados.

En caso de que un nodo deba recuperar su estado anterior, este tendría que procesar todos los mensajes encontrados en el disco, actualizar su estado interno y reprocesar el mensaje, mas no propagar o persistir nuevamente dicho mensaje.

Debido a que todos los workers deben como mínimo recuperar los números de secuencia esperados y los generados y como máximo, actualizar su estado interno con los valores de los mensajes persistidos, es que se optó por una implementación de recuperación en tres capas concurrentes, las cuales comunican la información pertinente a través de colas.

- La capa más abstraída y de más bajo nivel es la encargada de leer las líneas del archivo y parsearlas en estructuras fácilmente procesables por el sistema.
- La capa intermedia, aún común a todos los workers, es la encargada de actualizar el estado relacionado a los números de secuencia esperados y a generar.
- Por último, la capa de procesamiento, la cual es particular de cada worker y es la encargada de actualizar el estado interno del worker en función del mensaje recibido. Cabe destacar que los filtros no implementan esta capa debido a que no actualizan su estado interno con el payload del mensaje recibido (más allá de la recuperación de los números de secuencia).

Gateway

El gateway es el único nodo del sistema que mantiene conexiones TCP con los clientes, mientras que el resto de los nodos se comunican únicamente entre sí a través del middleware y no interactúan directamente con los clientes.

Para garantizar la consistencia en los resultados, cada batch que el gateway emite debe ser identificada de manera única mediante su número de secuencia (sequence ID). Esto es fundamental para evitar que el gateway propague la misma batch con diferentes identificadores, lo que podría generar inconsistencias, ya que los demás nodos no podrían detectar que se trata de la misma batch enviada de forma duplicada.

Para resolver este problema, el cliente informa al gateway el número de cada batch transmitida. El gateway utiliza esta información para construir el sequence ID de manera consistente, incluso ante posibles caídas o reinicios. De esta forma, se asegura que las retransmisiones o fallas no generen duplicados con identificadores distintos.

Dado que el gateway es susceptible a caídas y recuperaciones, el cliente deberá ahora filtrar los resultados que reciba del gateway, ignorando aquellos que lleguen de manera duplicada.

Por último, el gateway consume los resultados de las consultas de la cola de reports. Este proceso, al ser parte de la comunicación a través del middleware, emplea los mismos mecanismos de control de duplicados y recuperación (interfaz) que utilizan los demás nodos del sistema.

Oportunidades de Mejora

Si bien los aggregators no suponen en cuello de botella en el sistema, sería posible escalarlos de la misma forma que escala el gateway: segmentandolos mensajes según ID de cliente. De esta forma, todos los resultados de un mismo cliente irían a parar al mismo aggregator y, en caso de haber más de un cliente corriendo en paralelo, la carga provocada por estos se dividiría entre las instancias de aggregators disponibles.

Con el fin de mejorar el rendimiento de los workers stateless (por ejemplo: filtros de juegos o de reviews), se podría implementar un cambio de forma que las distintas instancias de cada uno de estos workers consuman de una misma cola, de forma que cuando una de las instancias se caiga, las demás podrían aprovechar el work-stealing para continuar con el procesamiento sin que la caída de dicha instancia afecte significativamente al sistema.

Sin embargo, esto involucraría realizar cambios en el manejo de los números de secuencia ya que podría suceder que una instancia reciba un mensaje anterior al último procesado debido a que podría corresponder al de una instancia caída.

Por último, una forma de mejorar el rendimiento a la hora de recuperar el estado de una instancia caída sería utilizar archivos de log segmentados por ID de cliente, de forma que, cuando se termina de procesar los mensajes de un cliente, este archivo podría ser eliminado para no tener que procesar sus mensajes en otra ocasión.

Si bien esta mejora ayudaría al rendimiento de la recuperación para múltiples clientes, no ayudaría para la recuperación de un cliente con un volumen de datos extremadamente elevado. Para este caso particular, podría utilizarse un archivo de checkpoint que permita a la instancia identificar a partir de qué mensaje debe procesar. Cabe destacar que estos cambios involucrarían una necesidad de actualizar el sistema de persistencia de números de secuencia para evitar eliminar o saltear el procesamiento de alguno de ellos.

División de tareas

Se propone una planificación tentativa del desarrollo.

Código Cliente: lectura de archivos y envío al gateway	lgrati@fi.uba.ar
Código Cliente: lectura de reportes y escritura en archivos	lgrati@fi.uba.ar
Gateway: config inicial y aceptar nuevas conexiones	Valentina Adelsflügel
Gateway: parseo de mensajes y uso de rabbitMQ	Valentina Adelsflügel
Gateway: envío de reportes	lgrati@fi.uba.ar
Filtros del primer nivel (review score, shooters, indie, platforms)	Gaston Mariano Frenkel
Query 1 & 2 workers + aggregators	lgrati@fi.uba.ar
Joiners (Group by)	Gaston Mariano Frenkel
Shards workers + aggregators	Valentina Adelsflügel
Persistencia	Gaston Mariano Frenkel
Filtrado de duplicados	Gaston Mariano Frenkel
Recuperación workers	Valentina Adelsflügel Gaston Mariano Frenkel
Recuperación gateway	Lucas Grati