75.74 Sistemas Distribuidos I - Trabajo Práctico N°2

Donato, Juan Pablo. Padrón 100839

Nov 2nd, 2021

Informe - Introduccion

El presente informe tiene como objetivo presentar las caracteristicas mas relevantes del Trabajo Práctico N°2 denominado **Middleware y Coordinación de Procesos**.

El trabajo consiste en el desarrollo de un sistema de arquitectura *Pipe & Filters*, donde se deben procesar un set de datos de preguntas y respuestas de *Stack Overflow* utilizando filtros que ejecutan determinadas operaciones en los datos, comunicandolos mediante colas de **RabbitMQ**.

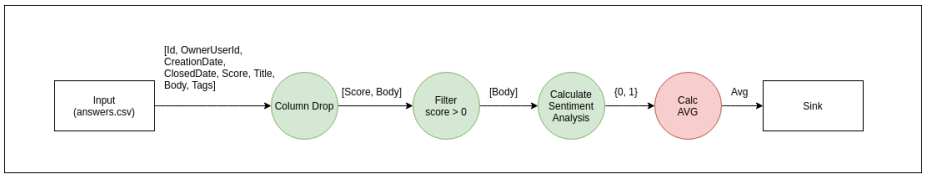
A continuacion se presentan diagramas de robustez y DAGs que servirán de entendimiento del tipo de arquitectura que se adoptó en este trabajo.

Primer Punto

La consigna del primer punto es:

```
Porcentaje de respuestas con score mayor a 10 que posea un sentiment analysis negativo
```

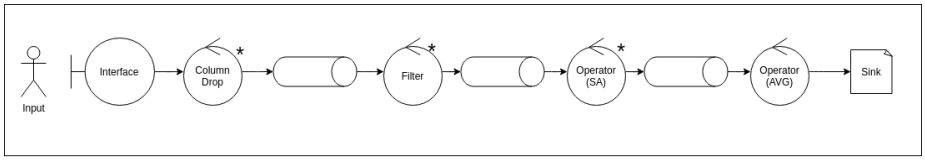
Para implementar este punto, realicé el siguiente DAG para poder visualizar mejor las tareas necesarias a realizar:



Podemos ver el flujo de datos desde la entrada a la salida, con el formato que debe tener entre etapas. Ademas, según el color de cada etapa:

- Verde: la etapa puede recibir un dato, procesarlo, y otorgar una salida, sin esperar que la etapa previa finalice (lo llamaremos *operator*)
- Rojo: la etapa recibe datos, los procesa, pero no otorga una salida hasta que la o las etapas previas finalicen en su totalidad (los llamaremos *holders*).

Teniendo en cuenta este DAG, se realiza un diagrama de robustez que muestra los distintos componentes a ser empleados para la resolución de este punto.



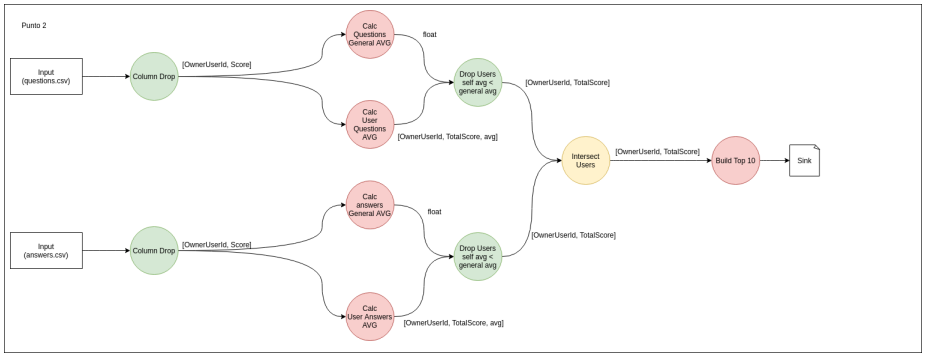
Como vemos, todos los componentes *operators* son **escalables**, y el último componente será el único que se mantendrá como único en su tipo. La comunicacion etapa a etapa se dará por colas de RabbitMQ *comunes*, a las que llamaremos *patron work_queue*.

Segundo Punto

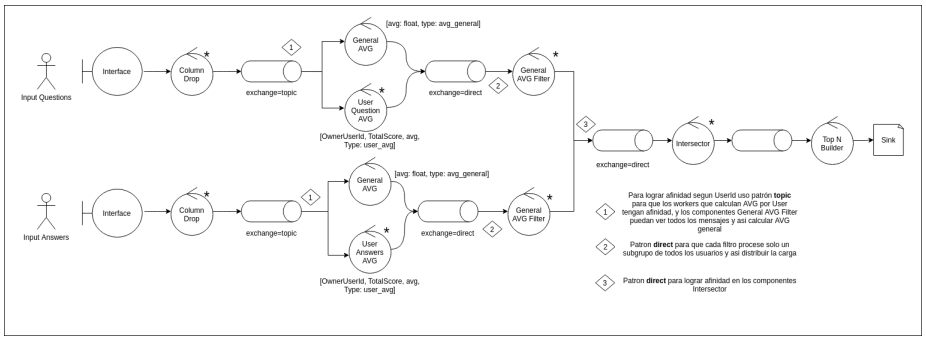
Para el segundo punto, la consigna fue:

`Top 10 de usuarios según score total (preguntas + respuestas), que tengan un puntaje promedi`

Para este caso (el más complejo de los tres), el DAG es el siguiente:



Como vemos, ahora tenemos dos inputs de datos distintos, y los flujos se vuelven un poco mas "bifurcados". Para eso, formulé el siguiente esquema:



En este nuevo diagrama, los puntos mas relevantes son:

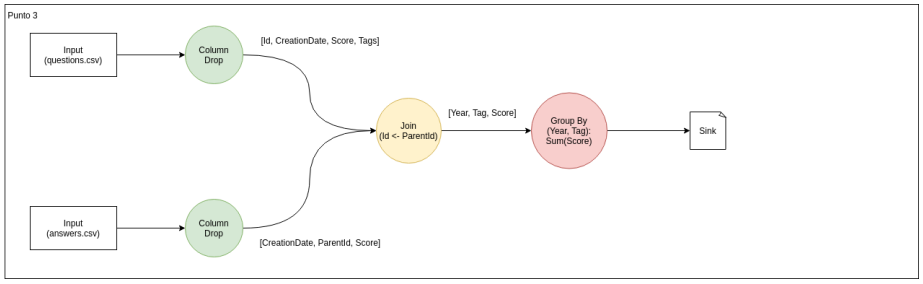
- La aparicion de nuevos esquemas o **patrones** de colas de RabbitMQ que serviran para, ademas de intercomunicar filtros, darle afinidad a los datos, de forma de distribuirlos en multiples nodos de una misma etapa a partir de una clave de hashing básica (como tomar el módulo de un dato númerico). Entre estos pátrones, estan:
  - **Direct**: nos servirá para darle afinidad a los datos y de esta forma distribuirlos en multiples nodos de una misma etapa por una clave de hashing básica de uno de los datos (como tomar el módulo de un dato númerico).
  - **Topic**: nos servirá para darle afinidad a los datos para llegar a los componentes denominados *User Question AVG* pero además permitirá al componente *General AVG* recibir la totalidad de los mensajes de esa cola para calcular el promedio global.
- La aparición de un nuevo componente (*partial operator*) en amarillo, el cuál recibe datos para procesarlos, generando una salida únicamente si ha recibido los datos necesarios para hacerlo.

Punto tres

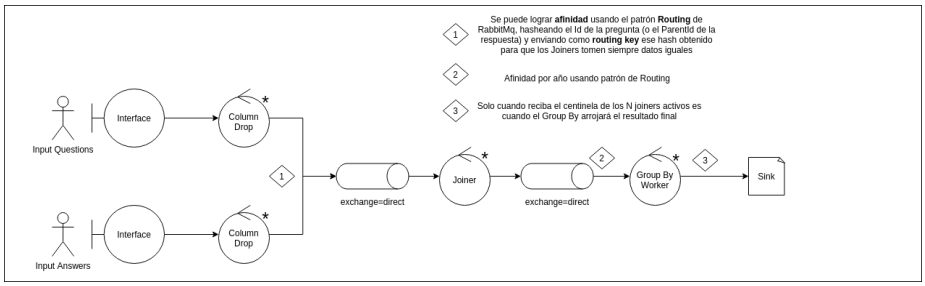
En éste último punto, la consigna fue:

`Top 10 de tags con mayor score (incluye preguntas y respuestas) de cada año`

Para esto, se formula el siguiente DAG, teniendo en cuenta las consideraciones de los puntos anteriores:



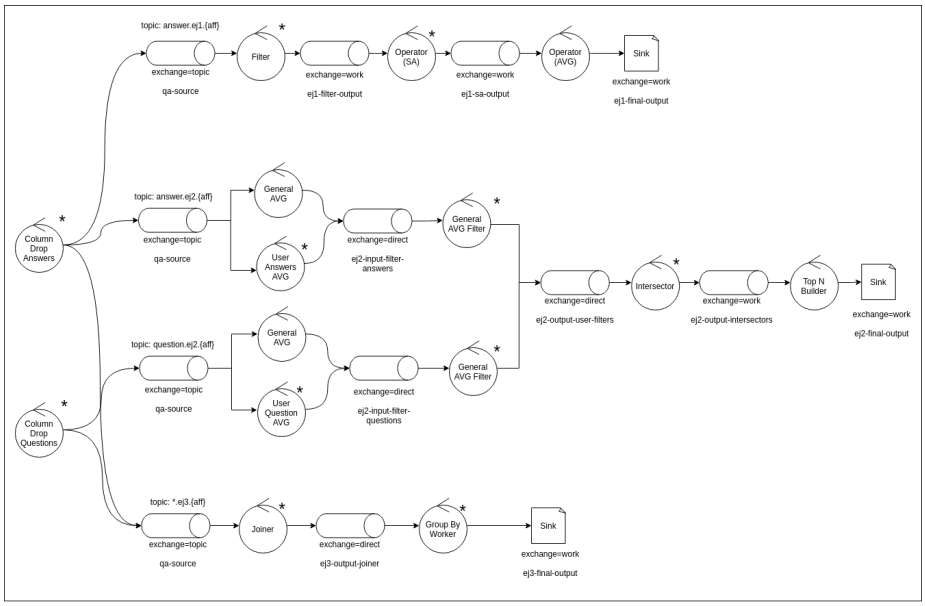
Este punto de dificultad intermedia entre los dos primeros puntos comparte el esquema de multiples inputs de datos. Para implementar esto, me guíe del siguiente diagrama de robustez:



En este caso tambien utilizacé el patrón **direct** para poder lograr afinidad de los datos al escalar los distintos componentes del flujo.

Estructura global

La división punto por punto me fue útil para diferenciar el trabajo y el alcance de cada etapa. Pero la consigna implica que el sistema pueda hacer estos tres flujos, *de forma simultanea*. Si juntamos los tres esquemas, logramos algo como esto:



La principal diferencia al unir los tres esquemas está en el componente que **elimina columnas dado el flujo que deban seguir los datos**. En este caso, dicho componente toma un input de los datos, y según la etapa, encola en la cola correspondiente las columnas que cada flujo necesita en su operatoria, utilizando

el patrón **topic**. De esta forma, estos componentes pueden:

- Procesar no una linea a la vez, sino un *chunk* de cada archivo de data inicial, y ofrecer como salida un *chunks* de datos
- Publicar mediante un exchange **topic** los datos en cada flujo, con las columnas que deban tomarse
- Lograr afinidad en los datos en aquellos *front components* de cada flujo que estuvieran replicados

Esquema del sistema

El desarrollo del trabajo está **muy orientado a la configuración de cada componente**. Esto quiere decir que cada componente recibe una cantidad (considerable) de parámetros que definirán su comportamiento, cargando todos en dos tipos de bloques: *operator* y *holder*, segun su comportamiento. Esto nos permite:

- Poder agregar nuevos componentes con nuevas funcionalidades de forma rápida
- Cambiando solamente las configuraciones adecuadas, poder adaptar y escalar nuestro sistema

Listado de configuraciones y variables de ambiente

- OPERATOR_MODULE: Define el tipo de operador del bloque al que se asigna (ej: Operador que calcula el Sentiment Analysis del primer punto)
- OPERATOR_PARAMS: Parámetro en formato JSON que define los argumentos que toma cada operador
- INPUT_QUEUE_PARAMS: Parametro en formato JSON que define el formato de queue de entrada al bloque. Incluye:
  - pattern: Patron de RabbitMQ (work_queue, direct o topic)
  - queue_name o exchange_name: Nombre de la cola o exchange a conectarse
  - routing_key: Clave de ruteo a utilizar
- OUTPUT_QUEUE_PARAMS: Parametro en formato JSON que define el formato de queue de salida al bloque. Incluye los mismos campos que su par INPUT_QUEUE_PARAMS
- CENTINELS_TO_RECEIVE: Cantidad de centinelas (mensajes 'END') a recibir de la etapa previa para considerarla como finalizada. Considera la cantidad de componentes de la o las etapas previas que estuvieran "conectados" al componente actual.

- CENTINELS_TO_SEND: Cantidad de centinelas a enviar a la etapa posterior para simbolizar la finalización del procesamiento del componente actual. Toma en cuenta la cantidad de componentes que existen en la etapa posterior.

De esta forma, por ejemplo en el punto 3 podemos agregar un nuevo **joiner** en la entrada de la siguiente forma:

```
ej3-joiner-1:
    container_name: ej3-joiner-1
    image: building-block:latest
    environment:
      - PYTHONUNBUFFERED=1
      - OPERATOR_MODULE=operators.joiner
      - OPERATOR_PARAMS={"perform_affinity":true,"affinity_key":"Year",
      "affinity_divider":2}
      - INPUT_QUEUE_PARAMS={"pattern":"topic","exchange_name":"qa-source",
      "routing_key":"*.ej3.0"}
      - OUTPUT_QUEUE_PARAMS={"pattern":"direct","exchange_name":"ej3-output-joiner"}
      - CENTINELS_TO_RECEIVE=5
      - CENTINELS_TO_SEND=2
    depends_on:
      - "rabbitmq-tp2"
    command: ["./wait-for", "rabbitmq-tp2:5672", "--", "python", "basic_operator.py"]
    networks:
      - tp2-network
```

De esta forma, estamos diciendole al nuevo bloque que:

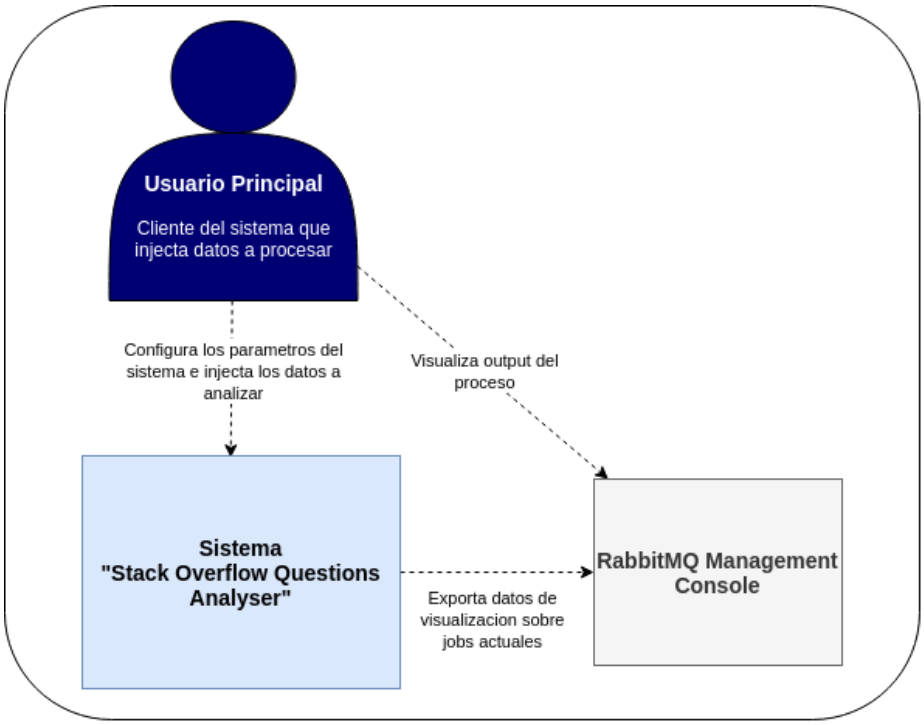
- Es el tipo `operators.joiner`
- Como parámetros del operador:
  - Debe realizar afinidad de los datos en la salida
  - La llave de afinidad es el campo `Year`
  - La clave de hashing implica tomar el resto por 2. Esto se debe a que la siguiente etapa posee dos componentes posibles para procesar la salida de este operador.
- Como cola de entrada
  - Patron "Topic"
  - El nombre del exchange a conectarse es `qa-source`
  - La clave de ruteo que debe contener el mensaje que salga de dicho exchange es `*.ej3.0`. Esto significa coloquialmente *"quiero recibir todos los mensajes (preguntas o respuestas) del ejercicio 3, cuya afinidad de datos sea 0"*. Otra clave de ruteo podria ser `answer.ej2.1`, traducida *"quiero todas los datos de tipo respuestas del ejercicio 2 cuya clave de afinidad sea 1"*.
- Como cola de salida
  - Patron "Direct"
  - Nombre del exchange de salida `ej2-input-filter-answers`

- Debe esperar la recepcion de 3 centinelas de la etapa anterior para darse como finalizada.
- Debe enviar 2 centinelas (a cada clave de afinidad entre 0 y *affinity_divider*) a la siguiente etapa para informar que ha finalizado.

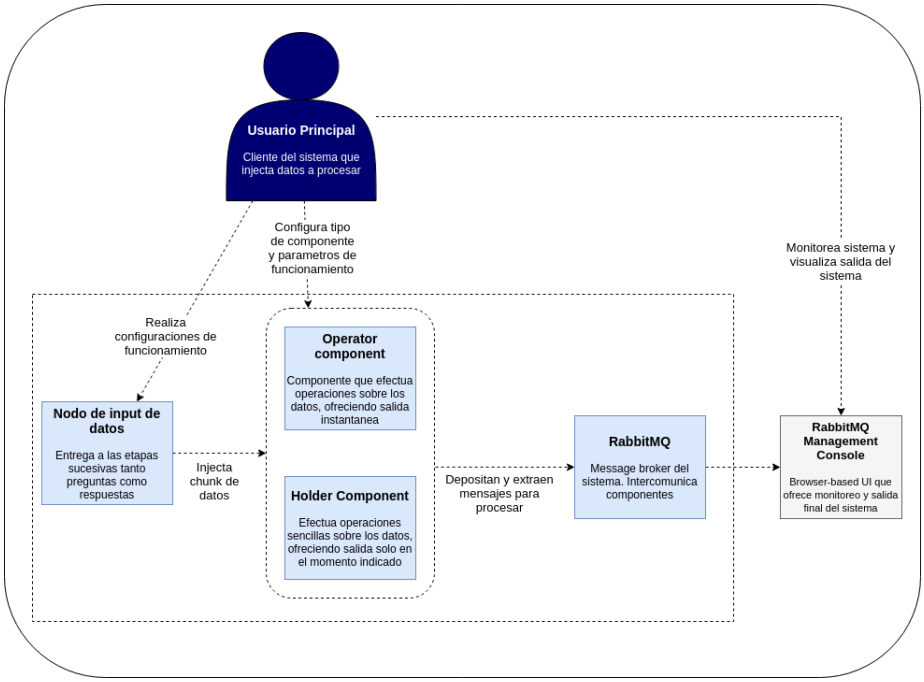
Notar en el campo `command` el comando `\wait-for` para esperar por la inicializacion de RabbitMQ para ejecutar nuestro bloque, y el llamado a `basic_operator.py` indicando que nuestro componente es un *operator*.

Diagrama C4

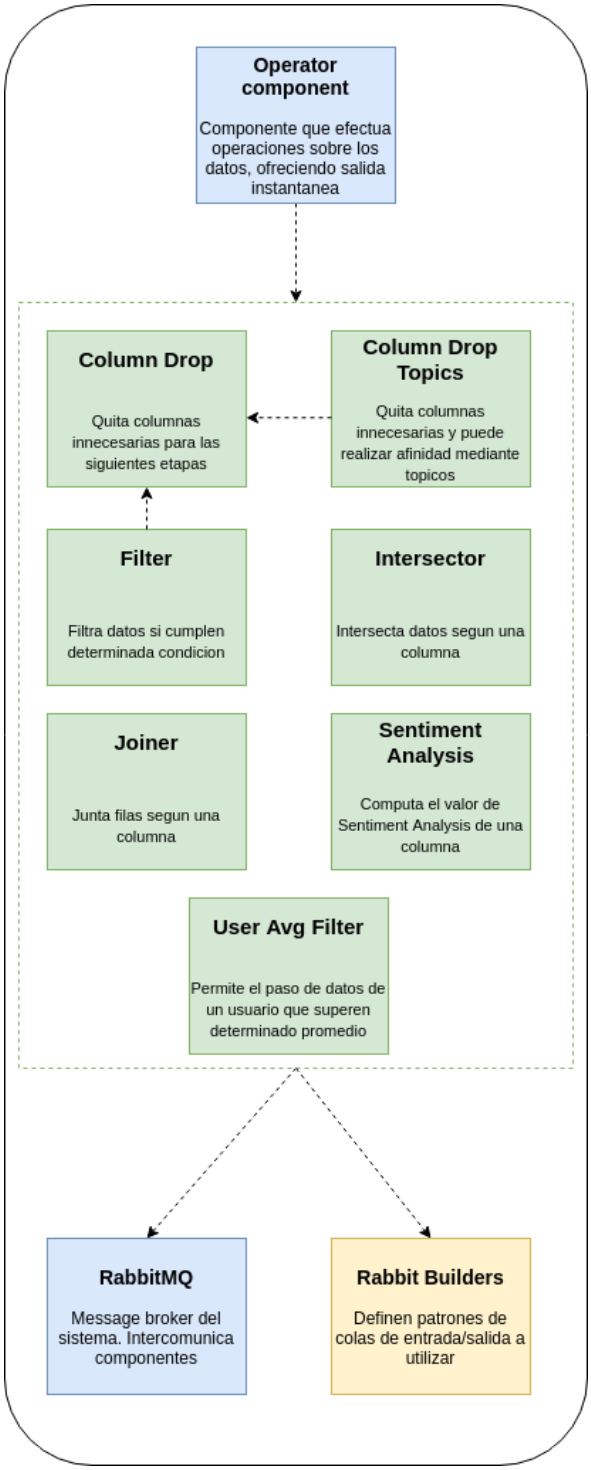
Level 1

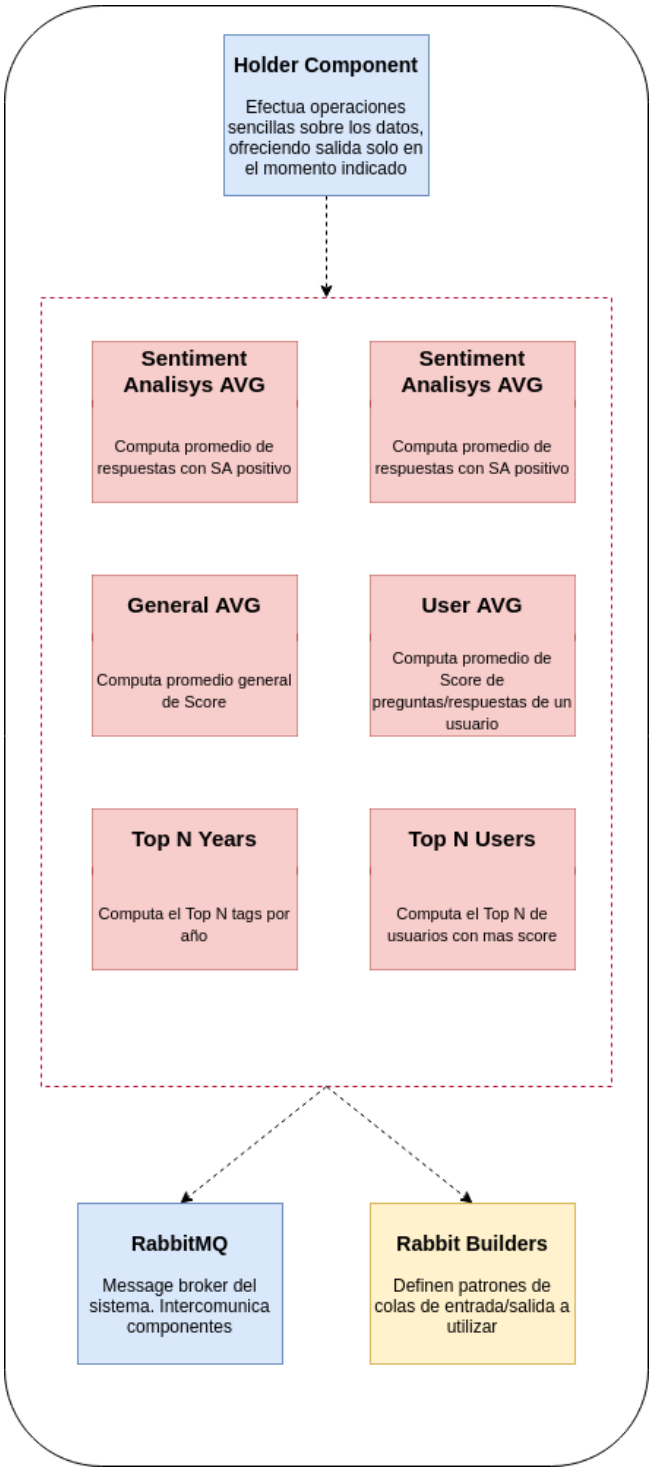


Level 2

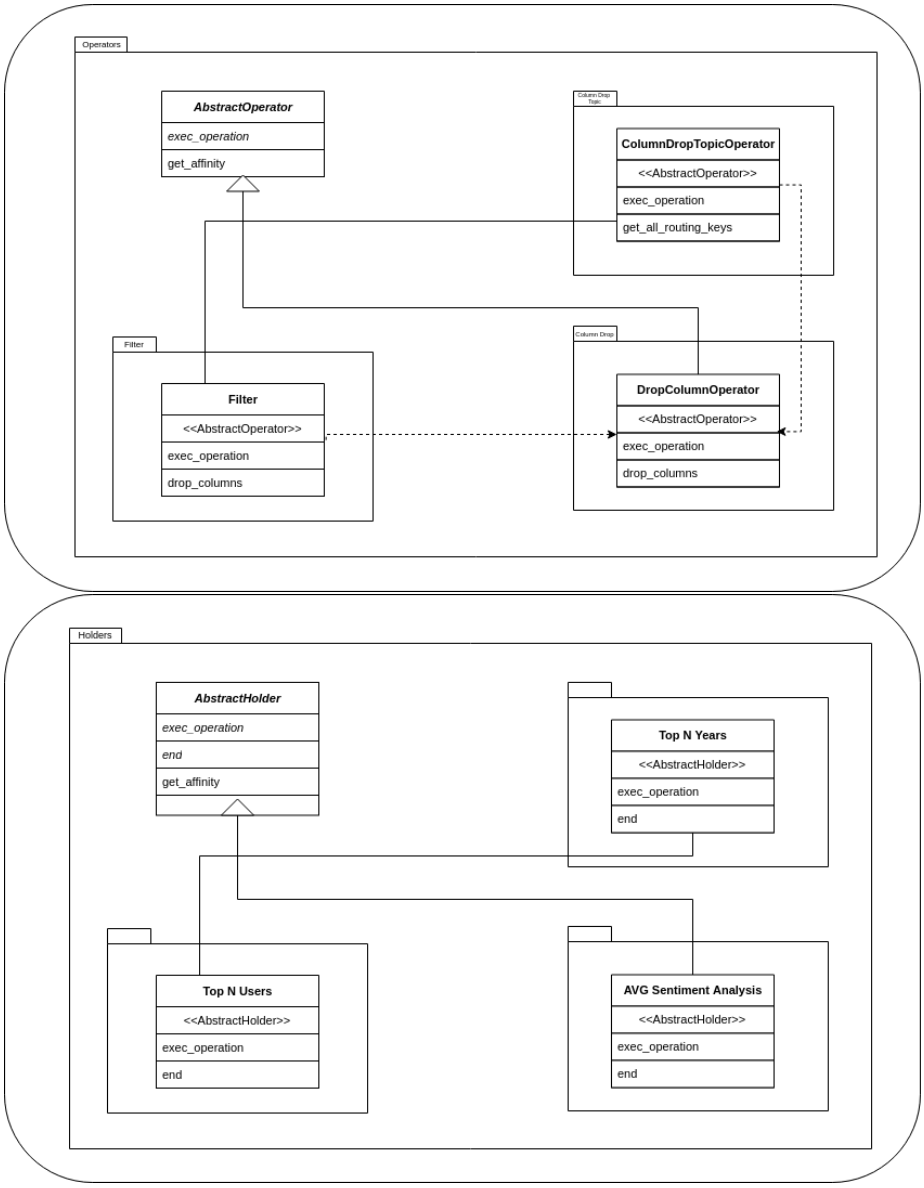

Usuario Principal
Cliente del sistema que injecta datos a procesar
Configura tipo de componente y parametros de funcionamiento
Monitorea sistema y visualiza salida del sistema
Realiza configuraciones de funcionamiento
Operator component
Componente que efectua operaciones sobre los datos, ofreciendo salida instantanea
Nodo de input de datos
Entrega a las etapas sucesivas tanto preguntas como respuestas
Injecta chunk de datos
Holder Component
Efectua operaciones sencillas sobre los datos, ofreciendo salida solo en el momento indicado
Depositan y extraen mensajes para procesar
RabbitMQ
Message broker del sistema. Intercomunica componentes
RabbitMQ Management Console
Browser-based UI que ofrece monitoreo y salida final del sistema

Level 3


Operator component
Componente que efectua operaciones sobre los datos, ofreciendo salida instantanea
Column Drop
Quita columnas innecesarias para las siguientes etapas
Column Drop Topics
Quita columnas innecesarias y puede realizar afinidad mediante topicos
Filter
Filtra datos si cumplen determinada condicion
Intersector
Intersecta datos segun una columna
Joiner
Junta filas segun una columna
Sentiment Analysis
Computa el valor de Sentiment Analysis de una columna
User Avg Filter
Permite el paso de datos de un usuario que superen determinado promedio
RabbitMQ
Message broker del sistema. Intercomunica componentes
Rabbit Builders
Definen patrones de colas de entrada/salida a utilizar


Holder Component
Efectua operaciones sencillas sobre los datos, ofreciendo salida solo en el momento indicado
Sentiment Analisys AVG
Computa promedio de respuestas con SA positivo
Sentiment Analisys AVG
Computa promedio de respuestas con SA positivo
General AVG
Computa promedio general de Score
User AVG
Computa promedio de Score de preguntas/respuestas de un usuario
Top N Years
Computa el Top N tags por año
Top N Users
Computa el Top N de usuarios con mas score
RabbitMQ
Message broker del sistema. Intercomunica componentes
Rabbit Builders
Definen patrones de colas de entrada/salida a utilizar

Level 4


Operators
AbstractOperator
exec_operation
get_affinity
ColumnDropTopicOperator
<<AbstractOperator>>
exec_operation
get_all_routing_keys
Filter
Filter
<<AbstractOperator>>
exec_operation
drop_columns
DropColumnOperator
<<AbstractOperator>>
exec_operation
drop_columns
Holders
AbstractHolder
exec_operation
end
get_affinity
Top N Years
<<AbstractHolder>>
exec_operation
end
Top N Users
<<AbstractHolder>>
exec_operation
end
AVG Sentiment Analysis
<<AbstractHolder>>
exec_operation
end