

UNIVERSIDADE DE SANTIAGO DE  
COMPOSTELA



ESCOLA TÉCNICA SUPERIOR DE ENXEÑARÍA

## Procesado masivo de eventos para o seu filtrado, análise e explotación

*Autor:*

**Alejandro José López Barcia**

*Titores:*

**Juan Carlos Pichel Campos**

**Javier Rubio Martín**

**Francisco Javier Álvarez Calvo**

**Grao en Enxeñaría Informática**

**Xullo 2021**

Traballo de Fin de Grao presentado na Escola Técnica Superior de Enxeñaría  
da Universidade de Santiago de Compostela para a obtención do Grao en  
Enxeñaría Informática





**D. Juan Carlos Pichel Campos**, Profesor do Departamento de Electrónica e Computación da Universidade de Santiago de Compostela, **D. Javier Rubio Martín**, *Infrastructure Support Specialist*, en ES Field Delivery Spain S.L., e **D. Francisco Javier Álvarez Calvo**, *Infrastructure Support Specialist*, en ES Field Delivery Spain S.L..

INFORMAN:

Que a presente memoria, titulada *Procesado masivo de eventos para o seu filtrado, análise e explotación*, presentada por **D. Alejandro José López Barcia** para superar os créditos correspondentes ao Traballo de Fin de Grao da titulación de Grao en Enxeñaría Informática, realizouse baixo nosa titoría no Departamento de Electrónica e Computación da Universidade de Santiago de Compostela.

E para que así conste aos efectos oportunos, expiden o presente informe en Santiago de Compostela, a 27 de xullo do 2021:

Titor,

Cotitor,

Cotitor,

Alumno,

Juan Carlos  
Pichel campos

Javier Rubio  
Martín

Francisco Javier  
Álvarez Calvo

Alejandro J.  
López Barcia



# Agradecementos

- A mamá e papá por apoiarme sempre en todo o que fago.
- Aos meus titores, Juan Carlos, Javier e Francisco Javier pola súa confianza.
- A Pablo e *Futfi*, por eses bos momentos que me destes.
- Aos amigos de toda a vida Ángel, Fran, Junior, Iván, Marcos e Fabián.



# Resumo

Neste Traballo de Fin de Grao propónse o desenvolvemento dun sistema que realice a recolección de publicacións de usuarios da rede social Twitter en base a uns termos de busca. O sistema realizará o procesado en tempo real do contido das publicacións obtidas mediante técnicas de filtración, mapeo, agrupación, agregación, redución e procesamento de linguaxe natural, para o estudo e clasificación dos textos dos usuarios. Os resultados obtidos serán conseguidos sen intervención humana e o estado do procesamento poderá ser visualizado polo usuario en todo momento.





# Índice xeral

<b>1. Introducción</b>	<b>1</b>
1.1. Motivación . . . . .	1
1.2. Obxectivos xerais . . . . .	2
1.3. Descrición do Sistema . . . . .	2
1.4. Elección do sistema . . . . .	4
1.5. Estrutura da memoria . . . . .	5
1.6. Alcance do proxecto . . . . .	6
1.6.1. Descrición do alcance do proxecto . . . . .	6
1.6.2. Entregables . . . . .	6
1.6.3. Criterios de aceptación . . . . .	6
1.6.4. Exclusiones do proxecto . . . . .	7
1.6.5. Restricións do proxecto . . . . .	7
1.6.6. Supostos do proxecto . . . . .	7
<b>2. Especificación de Requisitos</b>	<b>9</b>
2.1. Requisitos funcionais . . . . .	9
2.2. Requisitos non funcionais . . . . .	13
2.3. Catálogo de casos de uso . . . . .	15
2.4. Análise e xestión de riscos . . . . .	24
<b>3. Deseño</b>	<b>25</b>
3.1. Arquitectura do sistema . . . . .	25
3.1.1. Subsistema de obtención de datos . . . . .	26
3.1.2. Subsistema de procesamento en <i>streaming</i> . . . . .	27
3.1.3. Subsistema de almacenamento persistente . . . . .	32
3.1.4. Subsistema de visualización de resultados . . . . .	32
3.2. Contorna . . . . .	36
3.3. Padróns aplicados . . . . .	36
3.4. Diagrama de clases . . . . .	37
3.5. Diagramas de secuencia . . . . .	39
<b>4. Probas</b>	<b>41</b>
4.1. Deseño e execución das probas . . . . .	41

<b>5. Conclusións e posibles ampliacións</b>	<b>49</b>
<b>A. Manuais técnicos</b>	<b>51</b>
A.1. A topoloxía . . . . .	51
<b>B. Manuais de usuario</b>	<b>55</b>
B.1. Preparativos iniciais . . . . .	55
B.2. Despregue . . . . .	55
B.3. Posibles erros . . . . .	62
<b>Bibliografía</b>	<b>63</b>

# Índice de figuras

1.1. Comparativa do <i>throughput</i> e latencia entre Apache Kafka, Apache Pulsar e RabbitMQ. . . . .	5
2.1. Diagrama de casos de uso. . . . .	16
3.1. Diagrama de arquitectura do sistema. . . . .	26
3.2. Agregación dun <i>stream</i> de eventos nunha táboa [21]. . . . .	27
3.2. Agregación dun <i>stream</i> de eventos nunha táboa [21]. . . . .	28
3.3. <i>Tumbling time window</i> [22]. . . . .	28
3.4. <i>DAG</i> da topoloxía de procesamento. . . . .	30
3.5. Visualización do top de <i>hashtags</i> . . . . .	33
3.6. Visualización dos detalles do <i>hashtag</i> . . . . .	34
3.7. Visualización da análise de sentimento de inglés. . . . .	34
3.8. Visualización dos últimos dez <i>tweets</i> procesados que mencionan o <i>hashtag</i> . . . . .	35
3.9. Análise de sentimento de inglés . . . . .	36
3.10. Diagrama de clases do subsistema de obter datos. . . . .	37
3.11. Diagrama de clases do subsistema almacenador. . . . .	38
3.12. Diagrama de clases do subsistema web. . . . .	38
3.13. Diagrama de clases do subsistema de <i>streaming</i> . . . . .	39
3.14. Interacción do usuario co módulo de obtención de datos. . . . .	40
3.15. Interacción do usuario co módulo de obtención de datos. . . . .	40
4.1. Validación das probas de obtención de <i>tweets</i> e o seu correcto envío ao <i>broker</i> de Kafka. . . . .	42
4.2. Validación das probas de procesamento. . . . .	45
4.3. Validación das probas de almacenamento de <i>tweets</i> únicos. . . . .	46
4.4. Validación das probas sobre a interacción do módulo web coa API. . . . .	48



# Índice de cadros

2.1. RQF-001. . . . .	9
2.2. RQF-002. . . . .	10
2.3. RQF-003. . . . .	10
2.4. RQF-004. . . . .	10
2.5. RQF-005. . . . .	10
2.6. RQF-006. . . . .	11
2.7. RQF-007. . . . .	11
2.8. RQF-008. . . . .	11
2.9. RQF-009. . . . .	11
2.10. RQF-010. . . . .	12
2.11. RQF-011. . . . .	12
2.12. RQF-012. . . . .	12
2.13. RQF-013. . . . .	12
2.14. RQF-014. . . . .	13
2.15. RNF-001. . . . .	13
2.16. RNF-002. . . . .	13
2.17. RNF-003. . . . .	14
2.18. RNF-004. . . . .	14
2.19. RNF-005. . . . .	14
2.20. RNF-006. . . . .	14
2.21. RNF-007. . . . .	15
2.22. RNF-008. . . . .	15
2.23. CU-001. . . . .	17
2.24. CU-002. . . . .	18
2.25. CU-003. . . . .	19
2.26. CU-004. . . . .	19
2.27. CU-005. . . . .	20
2.28. CU-006. . . . .	21
2.29. CU-007. . . . .	22
2.30. CU-008. . . . .	23
2.31. CU-009. . . . .	23
4.1. CP-01. . . . .	41
4.2. CP-02. . . . .	42

4.3.	CP-03.	42
4.4.	CP-04.	43
4.5.	CP-05.	43
4.6.	CP-06.	43
4.7.	CP-07.	44
4.8.	CP-08.	44
4.9.	CP-09.	44
4.10.	CP-10.	45
4.11.	CP-11.	45
4.12.	CP-13.	46
4.13.	CP-14.	46
4.14.	CP-15.	46
4.15.	CP-16.	47
4.16.	CP-17.	47
4.17.	CP-18.	47
4.18.	CP-19.	48
4.19.	CP-20.	48

# Capítulo 1

## Introdución

### 1.1. Motivación

Inicialmente, as compañías almacenaban información que se podía describir como datos transaccionais, como poden ser usuarios, produtos, pedidos e outros obxectos gardados en táboas de bases de datos. Actualmente, esa descrición ampliouse, incluíndo datos relacionados con eventos. É dicir, datos que recollen cousas que suceden, en lugar de cousas que son. Este tipo de datos de eventos poden ser dende un rexistro de actividade de usuarios nun sitio web até as propias transaccións que ocorren nos sistemas operativos nas máquinas, almacenando un histórico de sucesos [1].

Dada á diversidade de información e a cantidade da mesma, o volume de datos a procesar incrementouse e a integración dos mesmos adquiriu unha maior dificultade, requirindo de sistemas que sexan capaces de procesar os datos en tempo real. Con isto, realízase unha transición dende o sistema tradicional de procesado por lotes (*batch processing*), onde se acumula a información en agrupacións, de maior ou menor tamaño, que son enviadas ao sistema de procesamento, resulta ineficiente pois prodúcese un colo de botella na recolleita e envío de datos.

O procesamento en fluxo (*stream processing*) xurde como unha alternativa para a recolección continua de datos como por exemplo a monitorización dos signos vitais, xeolocalización de vehículos militares mediante GPS ou a visualización da actividade económica do mercado financeiro. Este sistema permite recibir un fluxo de datos continuo, modelando a información de xeito uniforme, para ser lida e procesada de forma moito máis rápida e sinxela, garantindo un volume de entrada de información maior que nos modelos tradicionais cos mesmos recursos físicos [2].

O *stream processing* ten como núcleo central o procesamento continuo dun conxunto de datos que crece co tempo, xerando resultados de forma continua, a diferenza do procesamento por lotes, que os produce en intervalos fixados.

Unha das aplicacións que máis información xeran diariamente son as redes sociais. Con millóns de usuarios xerando unha gran cantidade de tráfico, este pode resultar de proveito para as compañías, pois pode ser procesado e analizado para logo poder ofrecer, por exemplo, produtos personalizados para cada usuario, ou incluso darse a coñecer en función das tendencias do momento.

Neste proxecto propónse o desenvolvemento dun sistema que recolla a información das publicacións dos usuarios da rede social Twitter, para ser analizada, recadando desta forma as opinións das persoas sobre certos temas de interese xeral.

## 1.2. Obxectivos xerais

Identifícase como obxectivo xeral formulado en base á motivación e finalidade do proxecto.

OBX-001	Elaboración dun software para o procesamento e visualización de <i>tweets</i>
Descrición	Desenvolver un software que obteña <i>tweets</i> publicados por usuarios en tempo real, os inxira, procese, analice e produza resultados que serán visualizados polos usuarios.
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta

Derivados do obxectivo principal, e como concreción do mesmo, identifícanse os seguintes obxectivos que permiten especificar as distintas funcionalidades que se queren conseguir.

## 1.3. Descrición do Sistema

Como xa se mencionou no apartado anterior, a proposta deste traballo é a implementación dun software que permita a obtención, procesado, analizado e



OBX-002	Obtención de datos
Descrición	Elaborar un sistema que, en base a uns criterios indicados, recolla <i>tweets</i> publicados en tempo real, e os transmita a un sistema de mensaxería.
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta

OBX-003	Procesado de datos e almacenamento
Descrición	Implantar un sistema preparado para recibir os <i>tweets</i> recollidos, aplique filtros, transformacións e recolla estadísticas dos mesmos en tempo real, así como proporcionar unha interface para a obtención dos resultados.
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta

OBX-004	Visualización de resultados
Descrición	O software deberá permitir a visualización dos resultados do procesamento de datos, en base ás consultas desexadas polo usuario.
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta

almacenamento de *tweets*. O deseño do sistema seguirá a arquitectura Kappa [3], e terá como nodo central que interconectará os subsistemas a plataforma Apache Kafka [4].

O subsistema de obtención de datos, recollerá os *tweets* en tempo real e envía-raos a un *cluster* de Kafka, agregándoos nunha cola. Paralelamente, executarase unha instancia de Kafka Streams [5], que realizará o procesado dos datos a medida que cheguen, identificando os *tweets* e poñéndoos noutra cola diferente, e gardando os resultados de todo o procesamento en almacenamento persistente a través da base de datos RocksDB [6] embebida en Streams permitindo a obtención dos seus resultados mediante unha API RESTful, que ofrecerá os resultados en formato JSON mediante peticións HTTP.

Por outro lado, existirá un cliente consumidor que permanecerá subscrito á cola sobre a que produce a aplicación de *streaming*, lendo a información enviada por esta e gardándoa nunha base de datos Elasticsearch [7], pois o formato NoSQL adecúase mellor para o almacenado dos *tweets* en lugar das bases de datos SQL tradicionais.

Finalmente, existirá unha aplicación web coa que o usuario poderá observar os resultados do procesamento a aplicación de *streaming* en tempo real.

## 1.4. Elección do sistema

Ademais de Apache Kafka, existen outros sistemas de mensaxería tradicionais tales como Apache Pulsar [8] ou RabbitMQ [9]. Para a comparativa con estes sistemas tómase como referencia as probas realizadas por Confluent [10], nas cales se centran no *throughput* e a latencia dos sistemas, pois son as métricas de rendemento fundamentais para os sistemas de *streaming* de eventos. As probas de *throughput* miden como de eficiente é cada sistema empregando o hardware, especialmente os discos e o procesador, mentres que as probas de latencia miden que tan cerca están ditos sistemas de proporcionar mensaxes en tempo real.

Na Figura 1.1 pódense observar os resultados das probas. A conclusión á que se chega é que Kafka entrega o mellor *throughput* en comparación cos outros dous sistemas mantendo unhas latencias considerablemente baixas.

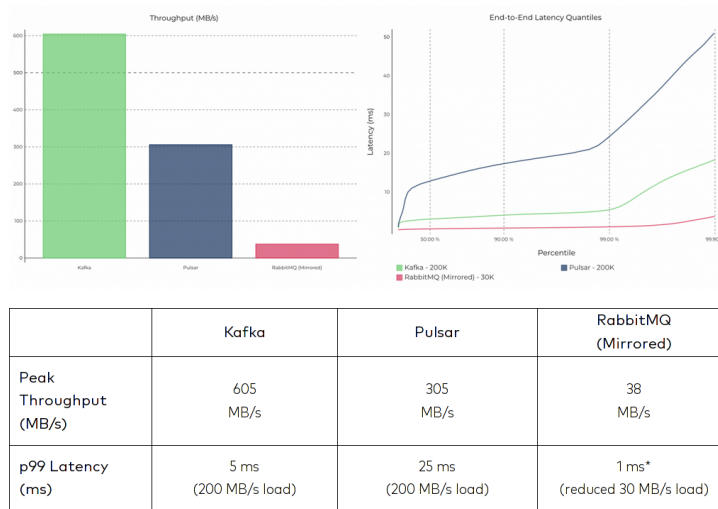


Figura 1.1: Comparativa do *throughput* e latencia entre Apache Kafka, Apache Pulsar e RabbitMQ.

A maiores, Kafka é capaz de manexar múltiples produtores independentemente de en cantos *topics* estean escribindo ou se hai varios sobre o mesmo *topic*. Do mesmo xeito, tamén permite a lectura de múltiples consumidores sobre un mesmo topic ao mesmo tempo, a diferenza do resto de sistemas que crean unha cola individual para cada consumidor.

Estes resultados pódense observar no número de empresas que empregan este sistema de mensaxería e na súa cota de mercado cun 18.94 % no momento da redacción desta memoria e máis de 19 mil empresas entre as que se atopan Spotify, Uber, Paypal, Cisco, CloudFlare e Netflix [11][12].

## 1.5. Estructura da memoria

Este documento divídese en seis capítulos nos cales se trata:

- No Capítulo 1 faise unha introdución sobre o proxecto, a motivación e obxectivos, e unha descrición do mesmo.
- No Capítulo 2 elabórase unha análise funcional do proxecto, especificando os requisitos.
- No Capítulo 3 descríbese a realización do sistema, como se divide e como interactúan as diferentes compoñentes do mesmo, así como as especificacións hardware e software necesarias.

- No Capítulo 4 deséñase un plan de probas e lévase a cabo para verificar o correcto funcionamento do sistema.

## 1.6. Alcance do proxecto

### 1.6.1. Descrición do alcance do proxecto

O produto a construír consistirá nun sistema que se encargará da obtención de datos, *tweets*, en tempo real, filtrado do seu contido para obter os que fan algunha mención a *hashtags*, obter o usuario máis relevante (con máis seguidores) de cada *hashtag* detectado, obter, como máximo, os últimos dez *tweets* que fagan mención ao *hashtag*, obter unha conta de aparicións do *hashtag* nos últimos 60 segundos, dividíndose en ventás temporais de 2 segundos. Dentro dos *hashtags* filtrados, realizar un novo filtro collendo os que están en idioma inglés, e realizar unha análise de sentimento para os *tweets* nese idioma.

O sistema deberá proporcionar unha interface coa que o usuario interactuará e poderá consultar os resultados das operacións mencionadas anteriormente. Ademais, deberá permitir ao usuario seleccionar temas de interese para a recolección de *tweets*.

### 1.6.2. Entregables

Os elementos a entregar na finalización do proxecto son:

- Software que cumpra os criterios de aceptación.
- Código fonte do software elaborado.
- Memoria do proxecto.

### 1.6.3. Criterios de aceptación

Recóllense os criterios para a aceptación do produto:

- Permite a obtención de *tweets* en tempo real.

- Realiza o filtrado, procesado e análise dos *tweets* obtidos en tempo real, e almacena os resultados das operacións en almacenamento persistente.
- Permite a realización de consultas para a manipulación e visualización dos resultados obtidos mediante unha API.

#### 1.6.4. Exclusiones do proxecto

Non se establecerá un sistema de control de acceso tanto para o almacenamento persistente como para a interface de visualización de datos.

#### 1.6.5. Restricións do proxecto

O proxecto conta coas seguintes restricións:

- O equipo de traballo está conformado por un único membro, que tomará os roles de xefe de proxecto, analista, deseñador e programador.
- A duración máxima do proxecto non superará as 412.5 horas.

#### 1.6.6. Supostos do proxecto

Os supostos do proxecto son os seguintes:

- A ferramenta empregada para o procesamento da linguaxe natural do contido dos *tweets* funciona correctamente, proporcionando un bo nivel de acerto na estimación do sentimento do seu contido.
- Os usuarios que proben o sistema posúen un mínimo coñecemento no manexo en interfaces de liña de comandos.



# Capítulo 2

## Especificación de Requisitos

Neste capítulo analízanse os requisitos máis relevantes do sistema, así coma a información que almacena e as interfaces con outros sistemas.

### 2.1. Requisitos funcionais

Os requisitos funcionais especifican as funcionalidades que o sistema ten que realizar.

RQF-001	Obtención de datos
Descrición	A aplicación deberá ser capaz de obter os <i>tweets</i> publicados en tempo real
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.1: RQF-001.

RQF-002	Especificación de temas de interese
Descrición	A aplicación deberá permitir a introdución por parte do usuario sobre temas que sexan mencionados ou relacionados nos <i>tweets</i> a recoller.
Importancia	Quedaría ben
Urxencia	Moderada
Comentarios	Sen comentarios

Cadro 2.2: RQF-002.

RQF-003	Múltiple execución de obtención de datos
Descrición	A aplicación deberá permitir a execución de paralela do módulo de obtención de datos.
Importancia	Quedaría ben
Urxencia	Baixa
Comentarios	Sen comentarios

Cadro 2.3: RQF-003.

RQF-004	Gardado en colas
Descrición	A aplicación deberá gardar os <i>tweets</i> obtidos en colas no sistema central para ser procesados polo resto de aplicacións.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.4: RQF-004.

RQF-005	Filtrado de <i>tweets</i>
Descrición	O sistema deberá procesar os <i>tweets</i> obtidos, filtrando os que teñan <i>hashtags</i>
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.5: RQF-005.



RQF-006	<i>tweets</i> duplicados
Descrición	O sistema deberá procesar os <i>tweets</i> obtidos, descartando aqueles que xa foron procesados.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.6: RQF-006.

RQF-007	Identificación de hashtags
Descrición	O sistema deberá identificar e recoller os hashtags que aparecen en cada <i>tweet</i> relacionándoos cos mesmos.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.7: RQF-007.

RQF-008	Análise de sentimento
Descrición	O sistema deberá filtrar os <i>tweets</i> con hashtags por idioma e realizar unha análise de sentimento se están escritos en inglés e almacenar, levando unha conta global dos resultados da análise para cada <i>hashtag</i> .
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.8: RQF-008.

RQF-009	Obtención de últimos <i>tweets</i>
Descrición	O sistema deberá recoller e almacenar, como máximo, os últimos dez <i>tweets</i> de cada hashtag procesado.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.9: RQF-009.

RQF-010	Obtención de usuario máis relevante
Descrición	O sistema deberá identificar para cada hashtag procesado, o usuario que realizou unha publicación do mesmo con máis seguidores.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.10: RQF-010.

RQF-011	Conta mediante ventás temporais
Descrición	O sistema deberá realizar a conta do número de veces que se menciona un hashtag durante un máximo de 60 segundos atrás no tempo, dividindo dito intervalo en 30 tramos de 2 segundos.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.11: RQF-011.

RQF-012	Agregación de resultados
Descrición	O sistema deberá agregar os resultados do procesamento en tempo real cos datos xa procesados dende o inicio.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.12: RQF-012.

RQF-013	Almacenamento persistente de datos
Descrición	O sistema deberá gardar os <i>tweets</i> obtidos, tanto os filtrados como os que non, nunha base de datos documental.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.13: RQF-013.

RQF-014	Interface para a obtención dos resultados
Descrición	O sistema deberá permitir a realización de solicitudes para obter os resultados datos que foron e están sendo procesados en tempo real.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.14: RQF-014.

## 2.2. Requisitos non funcionais

Nesta sección amósanse as condicións e restricións que o sistema debe cumprir.

RNF-001	Obtención dos datos
Descrición	O sistema empregará o <i>endpoint</i> proporcionado pola API de Twitter [13] para a descarga de <i>tweets</i> en tempo real.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.15: RNF-001.

RNF-002	Formato de envío dos datos
Descrición	O sistema empregará o formato JSON tanto para o envío dos <i>tweets</i> dun módulo a outro, como para os resultados das consultas interactivas.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.16: RNF-002.

RNF-003	Emprego de táboas e persistencia de estados
Descrición	O sistema transformará os fluxos de <i>tweets</i> en táboas aplicando os métodos necesarios para obter a información desexada, e gardará o estado das táboas.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.17: RNF-003.

RNF-004	Realización de consultas interactivas
Descrición	O sistema debe proporcionar unha API RESTful para a realización de consultas interactivas sobre os estados almacenados resultados do procesamento.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.18: RNF-004.

RNF-005	Almacenamento dos datos
Descrición	O sistema empregará unha base de datos Elasticsearch para o almacenamento persistente dos <i>tweets</i> recollidos.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.19: RNF-005.

RNF-006	Emprego de Apache Kafka como sistema de mensaxería central
Descrición	O sistema deberá empregar o servizo de Apache Kafka para o paso de <i>tweets</i> entre os diferentes subsistemas.
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.20: RNF-006.

RNF-007	Emprego de Kafka Streams para o procesamento de fluxo
Descrición	O sistema deberá empregar a API de Kafka Streams para construír unha topoloxía que permita o procesamento en tempo real dos <i>tweets</i> .
Importancia	Vital
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.21: RNF-007.

RNF-008	Posibilidade de engadir máis funcións á topoloxía de procesamento
Descrición	As funcionalidades de procesamento deberán poder ser ampliadas sen supoñer grandes dificultades.
Importancia	Importante
Urxencia	Alta
Comentarios	Sen comentarios

Cadro 2.22: RNF-008.

## 2.3. Catálogo de casos de uso

A continuación, na Figura 2.1 amósase o diagrama de casos de uso do sistema. Neste, descríbese de xeito informal a interacción entre os usuarios que empregarán o sistema e o sistema que se vai a desenvolver.

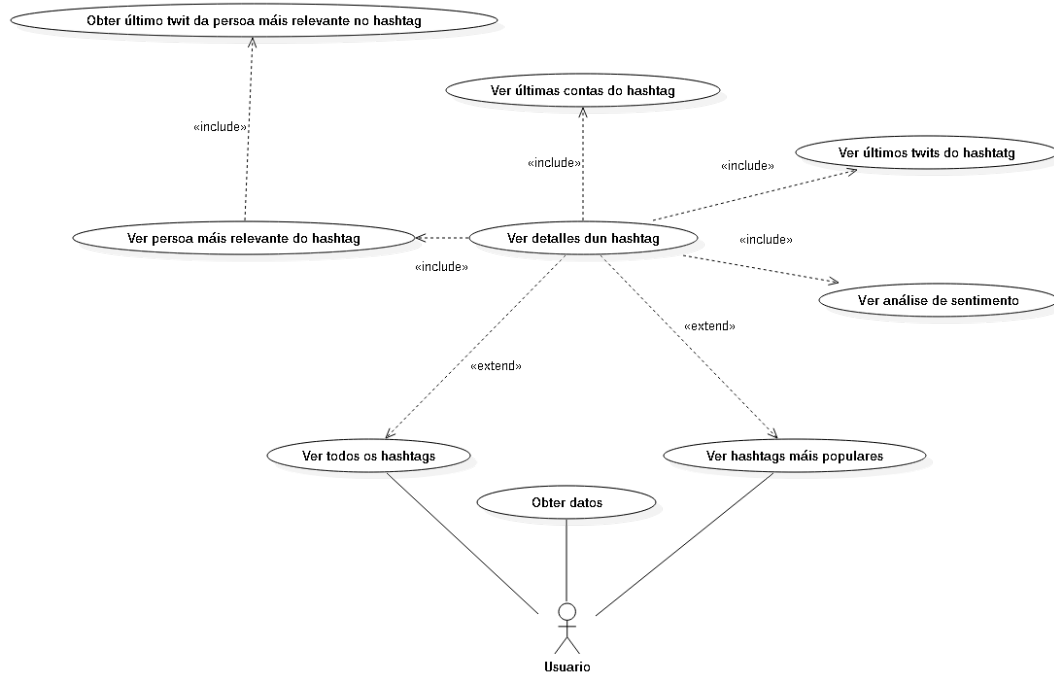


Figura 2.1: Diagrama de casos de uso.

CU-001	Obter datos
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando un usuario desexe obter datos para o seu procesado e visualización.
Precondición	Ningunha.
Escenario principal: <ol style="list-style-type: none"> <li>1. O usuario manifesta a súa intención de obter os <i>tweets</i>.</li> <li>2. O sistema solicita ao usuario os temas nos que está interesado.</li> <li>3. O usuario introduce os temas de interese.</li> <li>4. O sistema debe conectarse coa API de Twitter.</li> <li>5. O sistema comeza a recoller <i>tweets</i> relacionados cos temas introducidos polo usuario.</li> </ol>	
Postcondición	O sistema recolecta <i>tweets</i> .

Excepcións:	
<ol style="list-style-type: none"> <li>3. Se o usuario non introduce ningún tema de interese. O sistema solicitará ao usuario até que introduza como mínimo un.</li> <li>4. Se a API externa non se atopa dispoñible, o sistema non poderá descargar ningún <i>tweet</i>.</li> <li>5. Se os temas de interese introducidos polo usuario resultan pouco populares, o sistema pode tardar en recibir información da API.</li> </ol>	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	En función dos temas especificados polo usuario, o sistema pode conseguir máis ou menos información a maior ou menor velocidade.

Cadro 2.23: CU-001.

CU-002	Ver todos os <i>hashtags</i>
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando un usuario desexe ver tódolos <i>hashtags</i> analizados e procesados da información obtida.
Precondición	Ningunha.
Escenario principal: <ol style="list-style-type: none"> <li>1. O usuario manifesta a súa intención de ver tódolos <i>hashtags</i>.</li> <li>2. O sistema amosará tódolos <i>hashtags</i> procesados até o momento, ademais de amosar o número de aparicións de cada un.</li> <li>3. O usuario visualiza a información proporcionada polo sistema.</li> </ol>	
Postcondición	O usuario ve tódolos <i>hashtags</i> procesados procesados polo sistema e o número de aparicións de cada un.

Excepciones:	
2. Se o sistema non dispón de información almacenada nese momento, non poderá amosar ningunha información relacionada cos mesmos.	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	O caso excepcional pode darse debido aos temas de interese introducidos polo usuario, ou ben porque o sistema aínda non recolectou nin procesou información.

Cadro 2.24: CU-002.

CU-003	Ver <i>hashtags</i> máis populares
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando o usuario desexe ver os <i>hashtags</i> máis populares procesados polo sistema.
Precondición	Ningunha.
Escenario principal:	
1. O usuario manifesta a súa intención de ver os <i>hashtags</i> máis populares.  2. O sistema amosa os 50 <i>hashtags</i> con máis número de aparicións ordenados descendentemente.  3. O usuario visualiza a información proporcionada polo sistema.	
Postcondición	O usuario visualizará os <i>hashtags</i> máis populares procesados polo sistema.
Excepciones:	
2. Se o sistema non recibiu e/ou procesou a suficiente información, é posible que o número de <i>hashtags</i> amosados sexa inferior a 50.	
Importancia	Vital
Urxencia	Alta
Estado	Validado



Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.25: CU-003.

CU-004	Ver detalles dun <i>hashtag</i>
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando un usuario desexe ver detalles acerca dun <i>hashtag</i> procesado polo sistema.
Precondición	Ningunha.
<p>Escenario principal:</p> <ol style="list-style-type: none"> <li>1. O usuario manifesta a súa intención de ver os detalles acerca dun <i>hashtag</i>.</li> <li>2. O sistema amosa o usuario máis relevante que publicou sobre o <i>hashtag</i> seleccionado e o seu último <i>tweet</i> ao respecto, o número de aparicións do hashtag nos últimos 60 segundos en intervalos de 2, e os últimos 10 <i>tweets</i> publicados que mencionen dito hashtag.</li> <li>3. O usuario visualiza a información proporcionada polo sistema.</li> </ol>	
Postcondición	O usuario visualiza a información relacionada cun <i>hashtag</i> .
<p>Excepcións:</p> <ol style="list-style-type: none"> <li>2. En función do <i>hashtag</i> seleccionado polo usuario, a información amosada polo sistema pode ser inferior en número de <i>tweets</i> ou na realización da conta temporal.</li> </ol>	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.26: CU-004.

CU-005	Ver persoa máis relevante do <i>hashtag</i>
--------	---

Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando o usuario desexe ver detalles acerca dun hashtag procesado polo sistema.
Precondición	O usuario debe de indicar o hashtag do que desexe obter a información.
Escenario principal: <ol style="list-style-type: none"> <li>1. O usuario manifesta a súa intención de ver os detalles acerca dun <i>hashtag</i>.</li> <li>2. O sistema amosa o usuario con máis seguidores que publicou acerca do <i>hashtag</i>.</li> <li>3. O usuario visualiza a información proporcionada polo sistema.</li> </ol>	
Postcondición	O sistema amosa o usuario feito polo usuario máis relevante nese <i>hashtag</i> .
Excepcións:	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.27: CU-005.

CU-006	Obter último <i>tweet</i> da persoa máis relevante no <i>hashtag</i>
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando o usuario desexe ver detalles acerca dun <i>hashtag</i> procesado polo sistema.
Precondición	O usuario debe de indicar o <i>hashtag</i> do que desexe obter a información.

Escenario principal:	
<ol style="list-style-type: none"> <li>1. O usuario manifesta a súa intención de ver o <i>tweet</i> do usuario máis relevante dun <i>hashtag</i>.</li> <li>2. O sistema obtén o último <i>tweet</i> máis relevante dun <i>hashtag</i>.</li> <li>3. O sistema amosa o <i>tweet</i>.</li> <li>4. O usuario visualiza a información proporcionada polo sistema.</li> </ol>	
Postcondición	O usuario visualiza o usuario máis relevante dun <i>hashtag</i> .
Excepcións:	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.28: CU-006.

CU-007	Ver últimas contas do <i>hashtag</i>
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando o usuario desexe ver detalles acerca dun <i>hashtag</i> procesado polo sistema.
Precondición	O usuario debe de indicar o <i>hashtag</i> do que desexe obter a información.
Escenario principal:	
<ol style="list-style-type: none"> <li>1. O usuario manifesta a súa intención de obter as estadísticas referentes á aparición do <i>hashtag</i>.</li> <li>2. O sistema recolle as estadísticas de conta dos últimos 60 segundos, separándoas en fragmentos de 2 segundos.</li> <li>3. O sistema amosa ao usuario a información procesada.</li> <li>4. O usuario visualiza a información proporcionada polo sistema.</li> </ol>	

Postcondición	usuario visualiza as estadísticas de aparicións referentes ao <i>hashtag</i> seleccionado.
Excepcións:	
2. Se durante o procesamento dos datos recibidos o sistema non detecta aparicións do <i>hashtag</i> , é posible que se amosen menos fragmentos de tempo.	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.29: CU-007.

CU-008	Ver últimos <i>tweets</i> do <i>hashtag</i>
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando o usuario desexe ver detalles acerca dun <i>hashtag</i> procesado polo sistema.
Precondición	O usuario debe de indicar o <i>hashtag</i> do que desexe obter a información.
Escenario principal:	
1. O usuario manifesta a súa intención de ver os últimos <i>tweets</i> que mencionen o <i>hashtag</i> seleccionado.  2. O sistema recolle os últimos 10 <i>tweets</i> do <i>hashtag</i> indicado.  3. O sistema amosa a información recollida ao usuario.  4. O usuario visualiza os datos proporcionados polo sistema.	
Postcondición	O usuario visualiza os 10 últimos <i>tweets</i> cuxo contido mencione o <i>hashtag</i> seleccionado polo usuario.
Excepcións:	
2. Se un <i>hashtag</i> foi mencionado en menos de 10 <i>tweets</i> , o usuario visualizará un listado de <i>tweets</i> cuxo tamaño será inferior a 10, que serán todos os que mencionaron ao <i>hashtag</i> indicado.	
Importancia	Vital
Urxencia	Alta

Estado	Validado
Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.30: CU-008.

CU-009	Ver análise de sentimento
Descrición	O sistema deberá comportarse tal e como se describe no seguinte caso de uso cando o usuario desexe ver detalles acerca dun hashtag procesado polo sistema.
Precondición	O usuario debe de indicar o <i>hashtag</i> do que desexe obter a información.
Escenario principal: <ol style="list-style-type: none"> <li>1. O usuario anifesta a súa intención de ver os últimos <i>tweets</i> que mencionen o <i>hashtag</i> seleccionado.</li> <li>2. O sistema recolle os resultados nas análises de sentimento dos <i>tweets</i> en inglés.</li> <li>3. O sistema amosa a información recollida ao usuario.</li> <li>4. O usuario visualiza os datos proporcionados polo sistema.</li> </ol>	
Postcondición	O usuario visualiza os resultados da análise de sentimento dos <i>tweets</i> en inglés que mencionen o <i>hashtag</i> indicado.
Excepcións: <ol style="list-style-type: none"> <li>2. En función dos temas de interese introducidos polo usuario, é probable que o sistema non reciba <i>tweets</i> en idioma inglés.</li> </ol>	
Importancia	Vital
Urxencia	Alta
Estado	Validado
Estabilidade	Alta
Comentarios	Sen comentarios

Cadro 2.31: CU-009.

## 2.4. Análise e xestión de riscos

Neste apartado identifícanse os posibles riscos asociados ao proxecto e que fagan perigar o mesmo.

- O *endpoint* da API de Twitter empregado para obter os twits deixa de funcionar temporalmente, ou funciona de maneira errónea, proporcionando datos incorrectos. Isto imposibilitaría o acceso a novos datos ou a datos incorrectos, polo que o procesamento estancaríaase.
- O sistema resulta alimentado con máis datos dos que é capaz de procesar. Iso provocaría que se acumulasen os twits a procesar, dificultando a análise en tempo real, e incluso provocando a perda de información se o atraso é demasiado grande.

Dado que a probabilidade de que ocorran é baixa, aínda que o seu impacto sería moi grave no proxecto, resultando en modificacións no alcance e obxectivos do mesmo, estes riscos acéptanse.

# Capítulo 3

## Deseño

Este capítulo describe a arquitectura do sistema. Pártese dende unha visión global cun nivel alto de abstracción para facilitar a súa comprensión para logo tratar máis en profundidade cada compoñente do mesmo e como se comunican entre eles. Tamén se comentan as necesidades hardware e software.

### 3.1. Arquitectura do sistema

O sistema está deseñado baseándose na arquitectura Kappa. Esta arquitectura é unha simplificación da arquitectura Lambda [14], na que se elimina o sistema de procesamento de lotes, e toda a información flúe polo sistema de *streaming*. Os compoñentes desta arquitectura son:

- *Log* inmutable para a agregación de datos, de forma que soamente se poden engadir os novos datos ao final de dito *log*. Para o sistema proposto emprégase Apache Kafka.
- Sistema de computación en *streaming* que recibe a información do *log*. Para o sistema proposto emprégase Apache Streams.
- Almacenamento para a capa servidora (*Serving layer*) para proporcionar respostas optimizadas ás consultas.

A continuación amósase o diagrama de arquitectura do sistema. Como se pode observar dispón de catro subsistemas claramente diferenciados: un subsistema

de obtención de datos, un subsistema de procesamento en *streaming*, un subsistema para o almacenamento persistente e, por último, un subsistema web para a visualización de resultados.

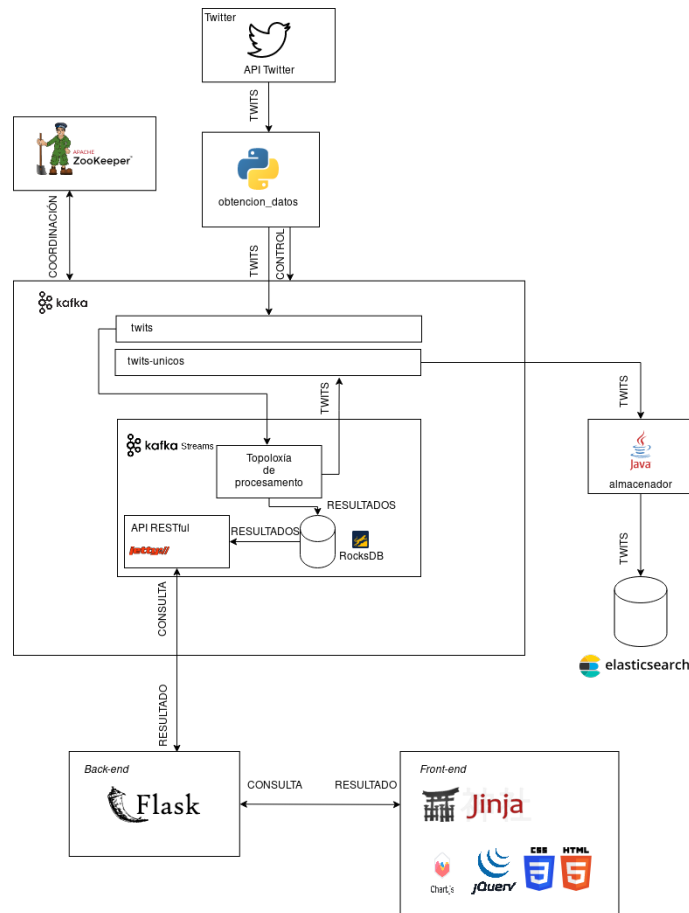


Figura 3.1: Diagrama de arquitectura do sistema.

### 3.1.1. Subsistema de obtención de datos

O subsistema de obtención de datos encárgase, por un lado, de obter as publicacións dos usuarios mediante a librería Tweepy [15]. Por outro lado, mediante a librería Kafka-Python [16] encárgase de facer unha posta a punto no *cluster* de Kafka creando os *topics* co o número de particións e factor de replicación de forma axeitada [17] e do envío dos *tweets* obtidos.

Primeiro, a aplicación solicita ao usuario un listado de termos polos que filtrar a obtención de *tweets*. A API de Twitter proporcionará os *tweets* nos que aparezan estes termos, ben sexa no texto ou en forma de *hashtags*.



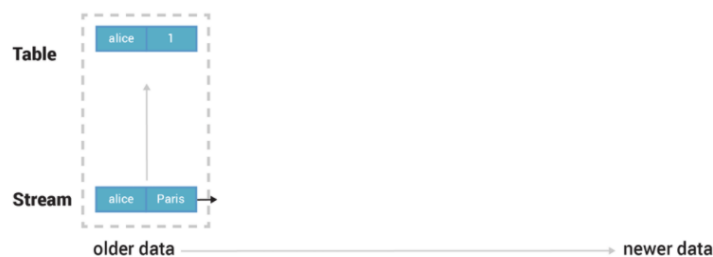
Unha vez indicados os termos, a aplicación conéctase coa API indicando as chaves e *tokens* de acceso, habéndose rexistrado previamente no portal de desenvolvedores de Twitter<sup>1</sup> e comeza a recolección de *tweets*, mediante a clase *Listener*. Cada vez que a aplicación recibe un *tweet*, envíao ao *cluster* serializándoo en formato JSON, e indica o número de offset que lle corresponde á mensaxe na partición, a modo de retroalimentación.

A aplicación de recolección deste subsistema poderá ser executada múltiples veces de forma paralela, permitindo a obtención de *tweets* sobre diferentes temas e incrementando a cantidade de datos a procesar polo subsistema de procesamento en *streaming*.

### 3.1.2. Subsistema de procesamento en *streaming*

Este subsistema realiza o procesamento dos datos en tempo real. Está implementado mediante a librería de Kafka Streams. Primeiro, defínese a topoloxía de nodos de procesamento na cal o nodo inicial consume os *tweets* do *topic tweets* e envíaos aos seguintes nodos. Cada un destes nodos de procesamento realiza unha tarefa e envía os resultados aos nodos fillos. Cada rexistro é procesado **completamente** por tódolos nodos do grafo antes de entrar o seguinte rexistro da partición. Mediante o encadeamento de múltiples procesadores, pódense crear lóxicas de procesamento complexas mentres que cada compoñente da mesma é sinxela, descompoñendo a topoloxía global en distintas subtopoloxías [18][19].

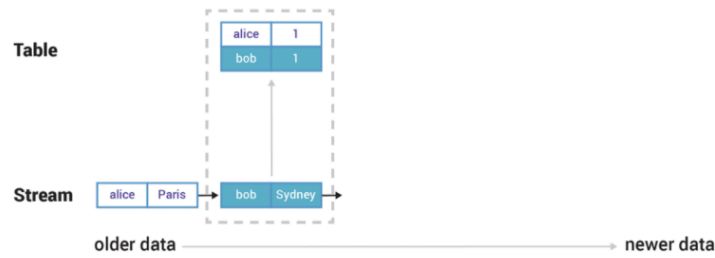
Durante o procesamento dos rexistros, os nodos poden aplicar diferentes transformacións aos datos, convertendo o *stream* nunha táboa [20]:



(a) Conversión do primeiro rexistro do fluxo a táboa.

Figura 3.2: Agregación dun *stream* de eventos nunha táboa [21].

<sup>1</sup><https://developer.twitter.com/en>.



(b) Agregación do seguinte rexistro á táboa.

Figura 3.2: Agregación dun *stream* de eventos nunha táboa [21].

### Ventás temporais

Como xa se indicou na especificación de requisitos, o subsistema de procesamiento deberá levar a conta temporal dos *hashtags* que aparezan. Isto lógrase facendo uso das **ventás temporais**, que delimitan o fluxo do *stream* en función de como se apliquen. Neste caso empréganse as *Tumbling time windows*, que modelan unha ventá de tamaño fixo, neste caso de 2 segundos, que, ao cumprilo prazo, gardan o estado en almacenamento persistente e reinician a conta.

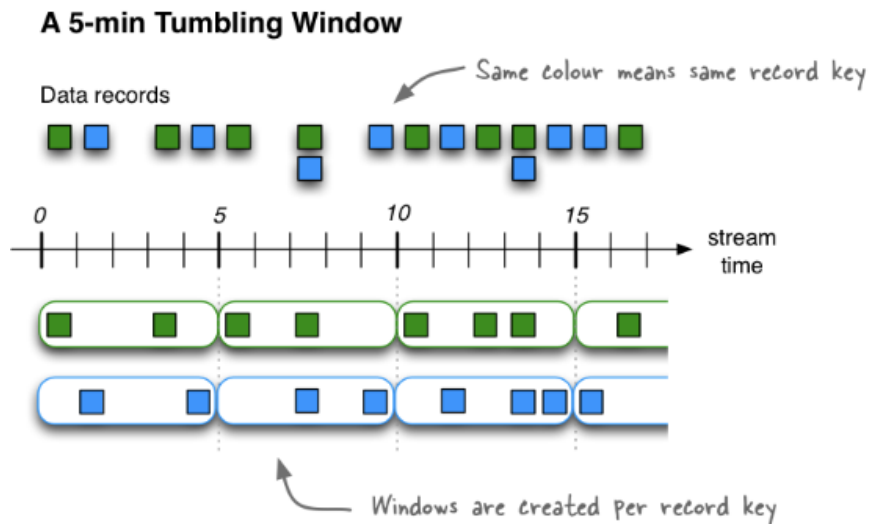


Figura 3.3: *Tumbling time window* [22].

## A topoloxía do sistema

Na Figura 3.4 obsérvase o diagrama da topoloxía de procesamento empregada. Neste, obsérvanse diferentes nodos *sink* e nodos *source*. Como xa se mencionou o nodo inicial extrae o fluxo de datos do *topic tweets*, mediante un **deserializador** que mapea o *array* de bytes a un obxecto da clase `Twit`. A continuación, os obxectos `Twit` pasan por un filtrado inicial no cal se identifica cada *tweet* polo seu *id*, e descártanse os repetidos. Os *tweets* filtrados envíanse a un *topic* denominado *twits-unicos*, que será o que lerá o consumidor do subsistema de almacenamento.

Posteriormente aplícase un filtro detectando se o texto do *tweet* contén algún *hashtag*<sup>2</sup>. En caso de que os *tweets* non conteñan ningún *hashtag* no seu texto son descartados continuando co seguinte rexistro.

Unha vez filtrados os textos cuxo contido conteñan algún *hashtag*, o seguinte nodo transforma o fluxo mediante un ***.flatMap()*** orixinando un novo *stream*, no que cada elemento está composto por un *ArrayList* de elementos, sendo a súa chave o propio *hashtag* identificado e o valor o *tweet* no que se menciona. Este novo fluxo de datos serve como fonte de datos para cada unha das análises que o sistema ten que realizar.

Os nodos e cadros indicados con “repartición” realizan cambios no tipo de fluxo mediante a selección de novos tipos para o par chave-valor. Esta transformación é o resultado de aplicar diferentes métodos, tales como ***.groupByKey()***, que xunta tódolos rexistros coa mesma chave, para despois aplicar outra transformación sobre os propios rexistros, podendo cambiar os tipos da chave e valor.

Nos nodos intermedios, aplícanse diferentes métodos como ***.aggregate()*** para a agregación dos valores dos rexistros, ***.count()*** para levar a conta do número de rexistros nas agrupacións de *streams* e ***.reduce()*** para combinar os valores dos rexistros. Ademais, faise uso do método ***.windowedBy()*** para establecer ventás temporais, e realizar desta forma a conta da aparición dos *hashtags* no tempo delimitado. A análise do sentimento sobre o contido dos *tweets* faise na subtopoloxía 1, facendo uso do módulo de procesamento de linguaxe natural Stanford CoreNLP [23].

Por último, nas subtopoloxías finais faise o almacenamento do estado dos datos en *stores*, indicado nos nodos como *materialize*, mediante o proceso de **materialización**<sup>3</sup>.

---

<sup>2</sup>A mención a un *hashtag* realízase da forma # e unha palabra ou frase, sen incluír espazos en branco, por exemplo, *#ETSE-USC*.

<sup>3</sup>Esta materialización realízase aplicando métodos coma ***.aggregate()*** pero especificando un *data store*.



Figura 3.4: *DAG* da topoloxía de procesamento.

Os *stores* conterán os resultados do procesamento realizado até o momento en tempo real e empregaranse como base de datos de solo lectura para a realización de consultas para a súa visualización. Existirán *stores* cuxos datos que resulten da acumulación dos resultados do procesamento, coma por exemplo as ventás temporais de 2 segundos, mentres que o contido de outros será sobrescrito, como pode ser o *tweet* co usuario máis relevante. A consulta dos datos farase a través dunha API RESTful grazas a un servidor web embebido na propia aplicación de Streams.

Dita API desprégase mediante o servidor web Eclipse Jetty [24] para permitir acceder aos estados internos da aplicación dende o exterior. Acompañado deste servidor faise uso de JAX-RS [25] para a creación do servizo web RESTful mediante o que poderá interactuar o subsistema web para obter os resultados.

A implementación, facendo uso das *Interactive Queries* [26] de Kafka Streams, permite consultar e percorrer os estados locais almacenados, e devolverá os resultados solicitados en función da consulta realizada<sup>4</sup>. A continuación lístanse os métodos que permite a API:

Petición	GET /
Descrición	Devolve o listado dos 50 <i>hashtags</i> con máis aparicións.

Petición	GET /todos
Descrición	Devolve o listado de tódolos <i>hashtags</i> que foron procesados polo sistema.

Petición	GET /{hashtag}
Descrición	Devolve a conta do número de veces que se detectou o <i>hashtag</i> indicado.

Petición	GET /{hashtag}/twit
Descrición	Devolve o <i>tweet</i> da persoa máis relevante que fixo unha publicación mencionando o <i>hashtag</i> indicado.

Petición	GET /{hashtag}/ultimos_twits
Descrición	Devolve o listado de últimos <i>tweets</i> almacenados que mencionen dito <i>hashtag</i> .

<sup>4</sup>Tómanse como exemplos os amosados de Confluent do seu repositorio de Github: <https://github.com/confluentinc/kafka-streams-examples>

Petición	GET /{hashtag}/sentimento/ingles
Descrición	Devolve o resultado da análise de sentimento dos <i>tweets</i> que mencionen dito <i>hashtag</i> e que estean escritos en idioma inglés.

Petición	GET /{hashtag}/window
Descrición	Devolve o número de veces que aparece o <i>hashtag</i> indicado nos últimos 60 segundos.

### 3.1.3. Subsistema de almacenamento persistente

Este subsistema encargarase de almacenar os *tweets* recollidos nunha base de datos *NoSQL* para a persistencia dos mesmos, pois o *cluster* de Kafka elimina os datos pasada unha semana [27]. Desta forma almacénanse os datos de forma segura, en caso de ser necesarios no futuro. Os campos da información almacenada na base de datos serán exactamente os mesmos que os da clase *Twit*, que se representará no diagrama de clases máis adiante.

Para este subsistema emprégase a librería Consumer API<sup>5</sup>, para a lectura dos datos nas particións e almacenalos en Elasticsearch facendo uso do cliente da librería JAX-RS.

### 3.1.4. Subsistema de visualización de resultados

Este subsistema proporciona unha interface gráfica que permite a visualización en tempo real dos datos procesados polo subsistema de *streaming*. A súa implementación faise mediante o *microframework* Flask [28] para a parte *back-end* mediante o mapeado de URLs a funcións grazas ó *app routing* [29]. Tamén uso da librería APScheduler [30] de Python, co que se programa un *job* en segundo plano para consultar periodicamente a información do estado do procesamiento en tempo real. Para a parte de *front-end* faise o uso de HTML5, CSS3, Jinja2 [31], jQuery [32] e Chart.js [33].

A vista principal amosa mediante unha gráfica os *hashtags* máis populares e a súa conta. Premendo nun dos *hashtags*, avánzase á vista de detalles.

<sup>5</sup><https://kafka.apache.org/documentation/#consumerapi>.

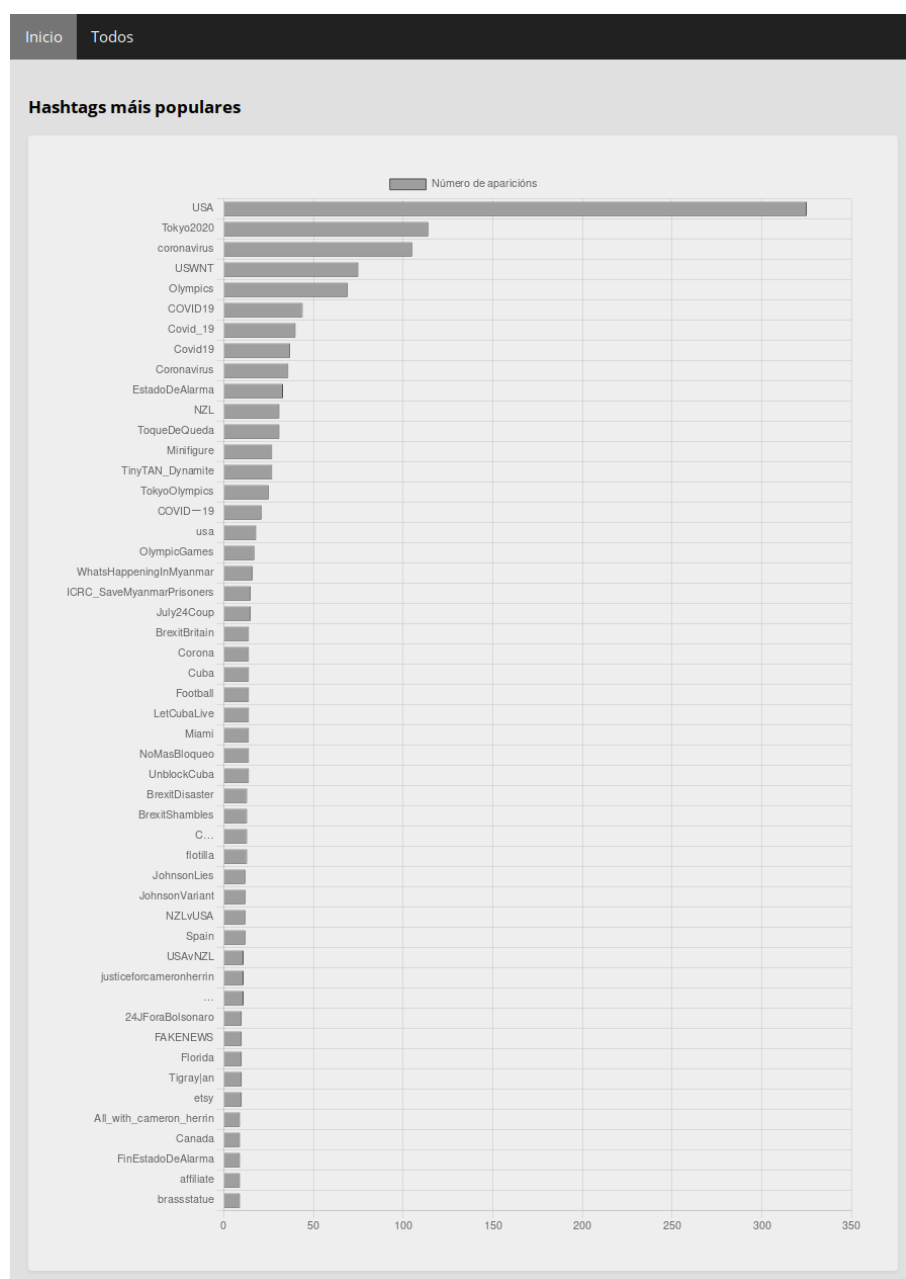


Figura 3.5: Visualización do top de *hashtags*.

Na vista de detalles pódense observar os resultados do procesamento: o total de aparicións do *hashtag* seleccionado, o usuario máis popular que mencionou dito *hashtag* e o seu último *tweet* en relación a este, unha gráfica que reproduce a aparición do *hashtag* nos últimos 60 segundos en intervalos de 2 segundos. Unha sección para observar a análise de sentimento dos *tweets* en inglés, e por último os últimos dez *tweets* que fan mención ao *hashtag* seleccionado.

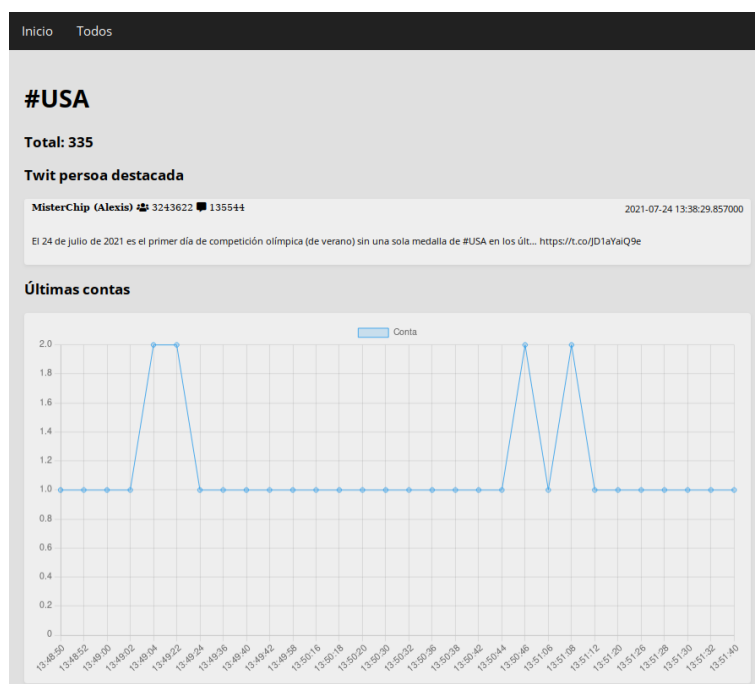


Figura 3.6: Visualización dos detalles do *hashtag*.

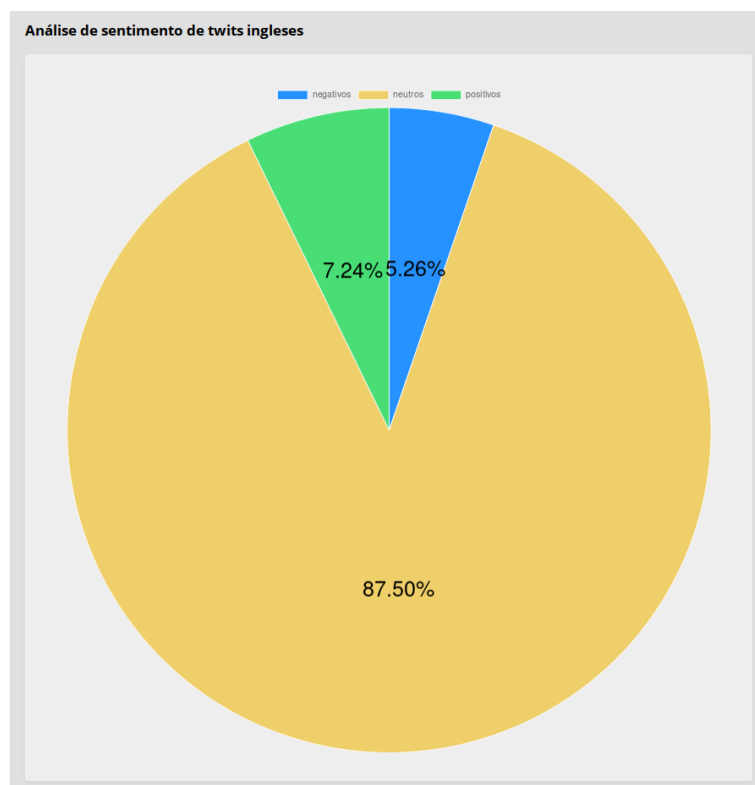


Figura 3.7: Visualización da análise de sentimento de inglés.



Últimos tweets		
<b>Silke</b> 🗨️ 1111 📢 313085		2021-07-24 13:52:17.771000
Mir fehlen die Worte ... <a href="https://t.co/RfpaspH233">https://t.co/RfpaspH233</a> #FoxNews #Vaccination #USA		
<b>Brunogafernandes</b> 🗨️ 85 📢 1019		2021-07-24 13:52:29.352000
RT @2010MisterChip: El 24 de julio de 2021 es el primer día de competición olímpica (de verano) sin una sola medalla de #USA en los últimos...		
<b>Jesus Ferran</b> 🗨️ 613 📢 15932		2021-07-24 13:52:36.248000
RT @marcferran: 🇺🇸 J. Holiday, Middleton y Booker no han entrenado hoy con el Team #USA -- según camino a Tokio, ha dicho LaVine. Popo...		
<b>Rodrigo Caldeira</b> 🗨️ 55 📢 3186		2021-07-24 13:52:36.585000
RT @2010MisterChip: El 24 de julio de 2021 es el primer día de competición olímpica (de verano) sin una sola medalla de #USA en los últimos...		
<b>Elias Abner Monasterio</b> 🗨️ 491 📢 44504		2021-07-24 13:52:47.130000
RT @2010MisterChip: El 24 de julio de 2021 es el primer día de competición olímpica (de verano) sin una sola medalla de #USA en los últimos...		
<b>Matthew Ehlers</b> 🗨️ 1788 📢 27508		2021-07-24 13:52:48.435000
Breakfast with Rose Lavelle. Delish! #USWNT #NZLvUSA 🇺🇸 #USA <a href="https://t.co/CHwecHyYxP">https://t.co/CHwecHyYxP</a>		
<b>Andi Purcell</b> 🗨️ 303 📢 12205		2021-07-24 13:52:53.045000
I'm obsessed with the way Rose Lavelle can absolutely dominate in big tournaments as a young player #uswnt #USA		
<b>Eldrick ISB-David FC</b> 🗨️ 6911 📢 72640		2021-07-24 13:52:58.797000
RT @2010MisterChip: El 24 de julio de 2021 es el primer día de competición olímpica (de verano) sin una sola medalla de #USA en los últimos...		
<b>Marcos Suarez</b> 🗨️ 226 📢 621		2021-07-24 13:53:03.888000
RT @CaraotaDigital: #USA   Advierten sobre posible colapso de avenida cercana a Champlain Towers 📺 <a href="https://t.co/a4mV6D2z8">https://t.co/a4mV6D2z8</a>		
<b>henriqueau</b> 🗨️ 465 📢 40258		2021-07-24 13:53:05.303000
RT @2010MisterChip: El 24 de julio de 2021 es el primer día de competición olímpica (de verano) sin una sola medalla de #USA en los últimos...		

Figura 3.8: Visualización dos últimos diez *tweets* procesados que mencionan o *hashtag*.

Na vista de todos poderase visualizar o número total de *hashtags* diferentes procesados até o momento, así como a conta individual de cada un, ordenados en forma descendente. Premendo en calquera un deles, pasarase á vista de detalles, explicada anteriormente.

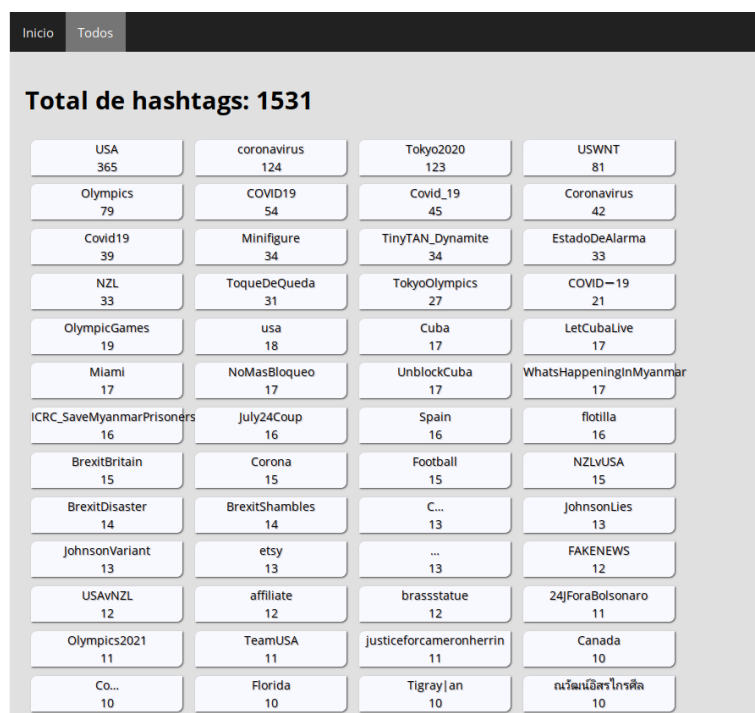


Figura 3.9: Análise de sentimento de inglés

## 3.2. Contorna

Os servizos de Apache Kafka, Apache ZooKeeper [34] e Elasticsearch execútanse en contedores empregando Docker [35] e Docker-Compose [36], cunha imaxe para cada servizo. As dependencias das aplicacións escritas en Java xestiónanse mediante Maven [37], ademais da compilación e execución das mesmas.

Tendo en conta a escala do actual proxecto e o consumo de recursos dos servizos empregados, a máquina na que se despreguen os contedores xunto coas aplicacións dos distintos módulos, debe posuír, como mínimo, **4 núcleos** e **8 GB** de memoria RAM.

## 3.3. Patróns aplicados

Emprégase o patrón publica-subscribe que permite separar o provedor da información dos consumidores. As aplicacións que empregan este patrón soamente deben coñecer o sistema intermediario no que se depositan as mensaxes.

Ademais tamén se emprega o patrón Modelo Vista Controlador para a interface web, e DAO para as consultas á API de Streams.

### 3.4. Diagrama de clases

A continuación amósanse os diagramas de clases de cada subsistema. Estes representan a estrutura e o comportamento de cada obxecto do sistema e as súas relacións co resto de obxectos.

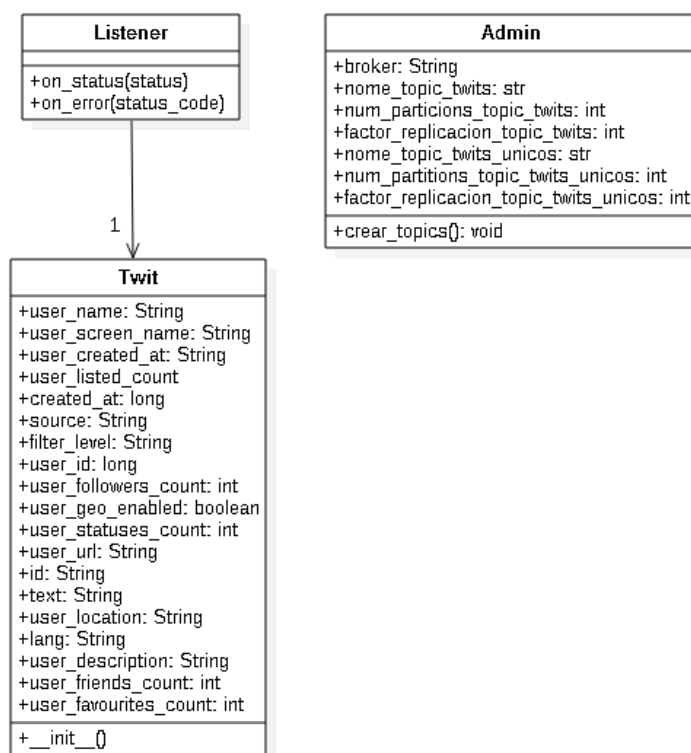


Figura 3.10: Diagrama de clases do subsistema de obter datos.

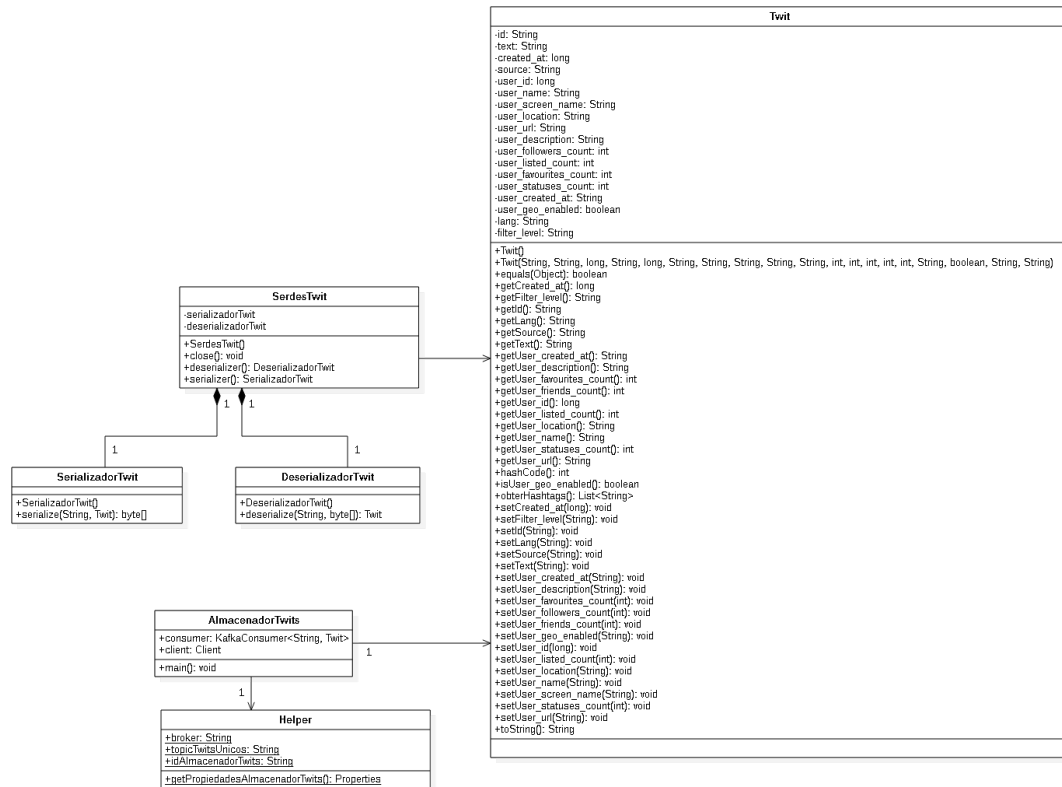


Figura 3.11: Diagrama de clases do subsistema almacenador.

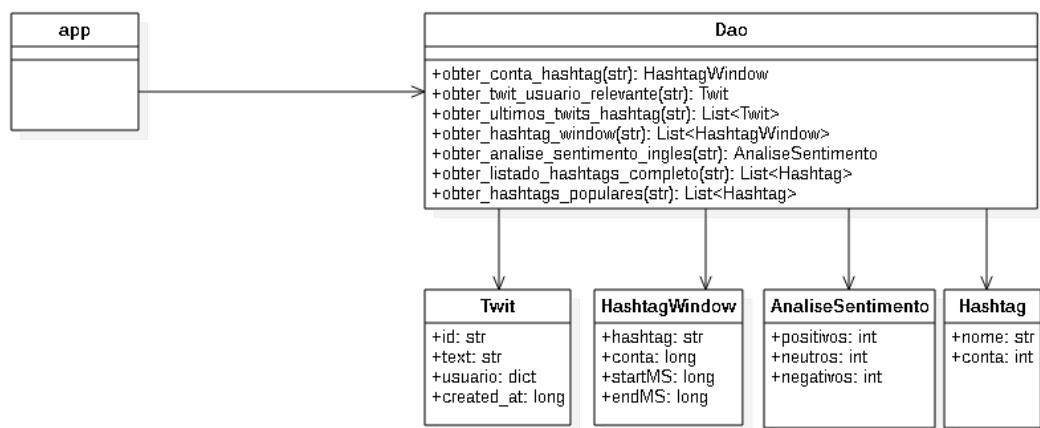


Figura 3.12: Diagrama de clases do subsistema web.



Por último amósanse os diagramas de secuencia para representar a interacción do usuario co sistema e como se realiza a comunicación entre os distintos procesos.

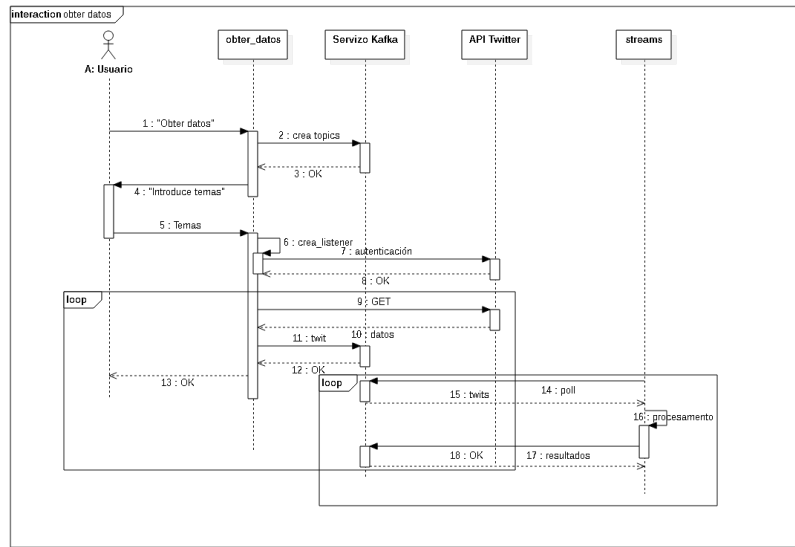


Figura 3.14: Interacción do usuario co módulo de obtención de datos.

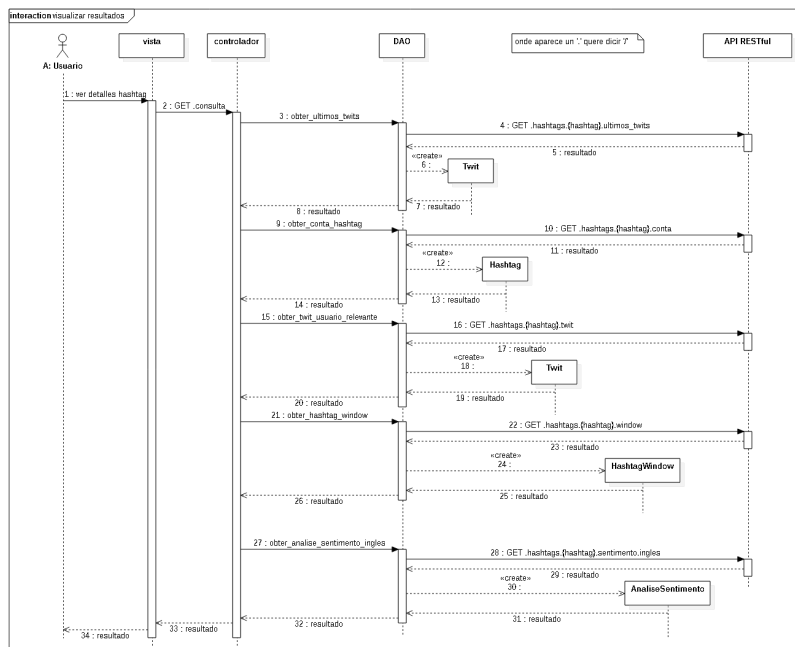


Figura 3.15: Interacción do usuario co módulo de obtención de datos.

# Capítulo 4

## Probas

Neste capítulo preséntase un grupo de probas e o seu resultado para comprobar que o correcto funcionamento do sistema.

### 4.1. Deseño e execución das probas

Deséñanse unha serie de probas para garantir que cada subsistema funciona adecuadamente e que se cumpren os requisitos necesarios para a aceptación do proxecto, xunto coas evidencias de validación das mesmas, axuntando as capturas correspondentes.

As probas foron feitas empregando JUnit [38], AssertJ [39] e os módulos propios para *testing* de Kafka nas aplicacións desenvoltas en Java, mentres que nas aplicacións escritas en Python empregouse o *framework* unittest [40].

Identificador	CP-01
Requisitos	[RQF-001], [RQF-002], [RQF-004]
Descrición	Execútase o módulo de obtención de <i>tweets</i> para comprobar que se envían ao <i>broker</i> e que se poden consumir polo resto de aplicacións subscritoras.
Resultado esperado	O consumidor deserializa correctamente os <i>tweets</i> enviados polo produtor.
Estado	Superada

Cadro 4.1: CP-01.

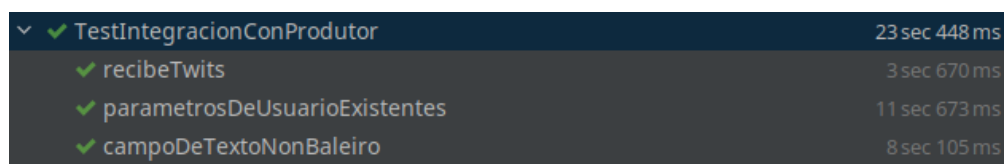
Identificador	CP-02
Requisitos	[RQF-001], [RQF-002], [RQF-004]
Descrición	Execútase o módulo de obtención de <i>tweets</i> para comprobar que se envían ao <i>broker</i> e que se detecta o campo de texto para realizar a análise dos mesmos.
Resultado esperado	O consumidor deserializa correctamente os <i>tweets</i> enviados polo produtor e detecta que o campo de texto non está baleiro.
Estado	Superada

Cadro 4.2: CP-02.

Identificador	CP-03
Requisitos	[RQF-001], [RQF-002], [RQF-004]
Descrición	Execútase o módulo de obtención de <i>tweets</i> para comprobar que se envían ao <i>broker</i> e que se deserializa correctamente o usuario para unha parte da análise en tempo real.
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta o usuario que publicou un tweet.
Estado	Superada

Cadro 4.3: CP-03.

Nesta batería de probas compróbase o correcto funcionamento da obtención *tweets*, do envío dos mesmos ao *broker* e de que os consumidores subscritos poden deserializar correctamente os bytes e convertelos a obxectos `Twit` para o seu uso. Na Figura 4.1 amósanse os seus resultados.



✓ TestIntegracionConProdutor	23 sec 448 ms
✓ recibeTwits	3 sec 670 ms
✓ parametrosDeUsuarioExistentes	11 sec 673 ms
✓ campoDeTextoNonBaleiro	8 sec 105 ms

Figura 4.1: Validación das probas de obtención de *tweets* e o seu correcto envío ao *broker* de Kafka.



Identificador	CP-04
Requisitos	[RQF-003]
Descrición	Filtrado de <i>tweets</i> repetidos en caso de que se obteña o mesmo twit dúas ou máis veces.
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta o identificador de <i>tweet</i> repetido e descarta o rexistro para o resto de nodos de procesado.
Estado	Superada

Cadro 4.4: CP-04.

Identificador	CP-05
Requisitos	[RQF-005]
Descrición	Detección de <i>hashtags</i> no contido dalgúns <i>tweets</i> , os cales pasarán a ser analizados.
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta, dentro do texto dos <i>tweets</i> , cales conteñen <i>hashtags</i> .
Estado	Superada

Cadro 4.5: CP-05.

Identificador	CP-06
Requisitos	[RQF-006]
Descrición	Detección de <i>hashtags</i> repetidos.
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta, dentro do texto dos <i>tweets</i> , cales conteñen <i>hashtags</i> e filtra os <i>hashtags</i> repetidos dun mesmo tweet, quedando soamente con unha aparición.
Estado	Superada

Cadro 4.6: CP-06.

Identificador	CP-07
Requisitos	[RQF-010]
Descrición	Comprobar que a conta do número de aparicións dos <i>hashtags</i> nos <i>tweets</i> se realiza de forma correcta dentro dun intervalo de 2 segundos.
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta e realiza a conta das veces que aparece o mesmo <i>hash-tag</i> en distintos <i>tweets</i> en menos de 2 segundos.
Estado	Superada

Cadro 4.7: CP-07.

Identificador	CP-08
Requisitos	[RQF-010], [RQF-011], [RQF-012]
Descrición	Comprobar que a conta do número de aparicións dos <i>hashtags</i> nun intervalo de 4 segundos se divide en 2 intervalos de 2 segundos, realizando correctamente as contas de cada un deles.
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta e realiza a conta das veces que aparece o mesmo <i>hash-tag</i> en distintos <i>tweets</i> e divide as deteccións en intervalos de 2 segundos.
Estado	Superada

Cadro 4.8: CP-08.

Identificador	CP-09
Requisitos	[RQF-009]
Descrición	Comprobar que ao enviar varios <i>tweets</i> facendo mención a un <i>hashtag</i> o procesador almacena o <i>tweet</i> cuxo usuario é máis relevante
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta o <i>tweet</i> co usuario máis relevante e almacénalo.
Estado	Superada

Cadro 4.9: CP-09.

Identificador	CP-10
Requisitos	[RQF-007]
Descrición	Comprobar que se realiza un filtrado de <i>tweets</i> escritos en inglés e se realiza unha análise de sentimento dos mesmos
Resultado esperado	O módulo de procesamento en <i>streaming</i> detecta <i>tweets</i> en inglés e acumula os resultados das análises nun resultado global.
Estado	Superada

Cadro 4.10: CP-10.

Na anterior batería de probas trátase de comprobar o funcionamento de cada unha das subtopoloxías do módulo de *streaming*, ademais do correcto filtrado dos nodos que descartan *tweets* repetidos, identifican *tweets* con *hashtags* e descartan a repetición do mesmo *hashtag* nun *tweet*. A continuación amósanse os resultados obtidos:

✓ TwitStreamsTestsUnitarios	1 sec 864 ms
✓ filtradoTwitInglesEAnaliseDeSentimento	1 sec 257 ms
✓ usuarioMaisRelevanteHashtag	254 ms
✓ almacenadoDezUltimosTwits	80 ms
✓ deteccionHashtagsRepetidosEnTwit	87 ms
✓ contaTemporalDeHashtags2HashtagWindows	44 ms
✓ contaTemporalDeHashtags	42 ms
✓ deteccionTwitsConHashtag	62 ms
✓ probaFiltradoTwitsRepetidos	38 ms

Figura 4.2: Validación das probas de procesamento.

Identificador	CP-11
Requisitos	[RQF-004]
Descrición	Comprobar que o almacenador pode ler os <i>tweets</i> únicos producidos polo procesador en <i>streaming</i>
Resultado esperado	O almacenador lee os tweets non repetidos.
Estado	Superada

Cadro 4.11: CP-11.

Identificador	CP-12
Requisitos	[RQF-004], [RQF-013]
Descrición	Comprobar que o almacenador envía os <i>tweets</i> únicos á base de datos de Elasticsearch
Resultado esperado	O garda os tweets únicos no servidor de Elasticsearch.
Estado	Superada

Cadro 4.12: CP-13.

Estas dúas sinxelas probas serven para comprobar que o módulo de almacenamento pode ler os *tweets* filtrados únicos e gardalos correctamente na base de datos. Na seguinte Figura pódense ver os resultados:

AlmacenadorTest	7 sec 73 ms
recibeTwitDeTopic	3 sec 673 ms
gardarTwitEnBD	3 sec 400 ms

Figura 4.3: Validación das probas de almacenamento de tweets únicos.

Identificador	CP-14
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> sobre a información dos <i>hash-tags</i> máis populares.
Resultado esperado	O módulo web recibe información acerca dos <i>hash-tags</i> máis populares procesados até o momento.
Estado	Superada

Cadro 4.13: CP-14.

Identificador	CP-15
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> sobre o número de aparicións dun <i>hashtag</i> .
Resultado esperado	O módulo web recibe información acerca do número de veces que se contabilizou un <i>hashtag</i> .
Estado	Superada

Cadro 4.14: CP-15.

Identificador	CP-16
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> sobre o número de aparicións dun <i>hashtag</i> que non existe.
Resultado esperado	O módulo web obtén un erro ao consultar a conta sobre as aparicións dun <i>hashtag</i> inexistente.
Estado	Superada

Cadro 4.15: CP-16.

Identificador	CP-17
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> para obter o usuario máis relevante dun <i>hashtag</i> .
Resultado esperado	O módulo web recibe información sobre o usuario máis relevante do <i>hashtag</i> e o seu último <i>tweet</i> .
Estado	Superada

Cadro 4.16: CP-17.

Identificador	CP-18
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> sobre os últimos <i>tweets</i> dun <i>hashtag</i> .
Resultado esperado	O módulo web recibe un listado de tweets de tamaño inferior ou igual a 10.
Estado	Superada

Cadro 4.17: CP-18.

Identificador	CP-19
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> sobre a conta de aparicións dun <i>hashtag</i> .
Resultado esperado	O módulo web recibe un listado coas contas de aparicións, indicando a ventá temporal de cada unha e a súa conta.
Estado	Superada

Cadro 4.18: CP-19.

Identificador	CP-20
Requisitos	[RQF-0014]
Descrición	Comprobar que o módulo web pode consultar á API integrada en <i>streams</i> sobre os resultados da análise de sentimento de <i>tweets</i> en inglés que mencionen un <i>hashtag</i> .
Resultado esperado	O módulo web recibe os resultados da análise de sentimento, indicando o número de <i>tweets</i> positivos, negativos e neutrais.
Estado	Superada

Cadro 4.19: CP-20.

Esta última batería de probas permite comprobar o correcto funcionamento entre o módulo web e a API RESTful embebida no módulo de *streaming*, para, desta forma, consultar os resultados e amosalos na páxina web. Na Figura 4.4 válidase a comunicación entre ambos módulos:

```
test_obter_analise_sentimento (__main__.TestAdapter) ... ok
test_obter_conta_hashtag (__main__.TestAdapter) ... ok
test_obter_conta_hashtag_inexistente (__main__.TestAdapter) ... ok
test_obter_hashtag_window (__main__.TestAdapter) ... ok
test_obter_hashtags_populares (__main__.TestAdapter) ... ok
test_obter_ultimos_twits_hashtag (__main__.TestAdapter) ... ok
test_obter_usuario_relevante (__main__.TestAdapter) ... ok

-----
Ran 7 tests in 0.017s

OK
```

Figura 4.4: Validación das probas sobre a interacción do módulo web coa API.

## Capítulo 5

# Conclusións e posibles ampliacións

Neste traballo levouse a cabo unha primeira toma de contacto co procesamento masivo de datos e o *Big Data*, adquirindo unha gran cantidade de información aparentemente inconexa que, despois que pasar por unha topoloxía de procesamento, obtéñense resultados que relacionan os datos entre si. Ademais, realízase mediante a arquitectura Kappa, polo que se reduce a complexidade do sistema resultando nun con poucos compoñentes modulares e simples.

Debido a que Apache Kafka actúa como un sistema central de mensaxería polo que pode pasar información non só dun tipo, coma a que se procesa neste proxecto, senón de diferentes mediante a asignación de *topics*, polo que permite ter diferentes aplicacións produtoras xerando datos cara o *cluster*, mentres que existen aplicacións consumidoras que len os datos que desexen. Grazas a isto, este sistema é facilmente escalable, tanto de forma vertical, executando varias instancias das mesmas aplicacións nunha mesma máquina, como horizontalmente, executando as aplicacións en diferentes equipos.

Ademais, a implementación dunha topoloxía de *streaming* permite realizar o procesamento en paralelo da información que vai chegando ao *cluster*, de forma que se poden realizar diferentes tipos de filtrados, análises, agrupacións, e, en resumo, calquera tipo de transformación sobre os datos, grazas á introdución dos nodos de procesamento e á composición de subtopoloxías, que permiten escalar de forma moi sinxela as capacidades de procesamento do sistema de *streaming*.

Tamén se observou que debido á sinxeleza de como actúa a aplicación de *streaming* para almacenar o estado do procesamento, e a velocidade de iteración sobre o mesmo, permite elaborar unha interface sobre a que un conxunto de

aplicacións externas poden consultar os datos almacenados de forma moi rápida e nun formato estandarizado.

Unha desvantaxe que se observa durante as probas é que no caso de que as aplicacións produtoras envíen unha gran cantidade de información, e a medida que a topoloxía global se volve máis complexa, o tempo de procesamento aumenta, e como as mensaxes soamente entran na etapa de procesamento cando remata a anterior, provócase unha acumulación das mensaxes a procesar.

Por isto, o principal obxectivo de mellora deste proxecto é incrementar a velocidade de procesamento da aplicación de *streaming* para que non se acumulen rexistros sen procesar nas particións, chegando ao seu borrado total. É necesario ter en conta tamén que, aínda que se incremente a velocidade de procesamento até o punto de que non se produza un colo de botella, estes poderían ocasionarse noutros puntos do sistema, principalmente no módulo de obtención de datos, pois as librerías empregadas constitúen unhas importantes limitacións á hora de recadar unha gran cantidade de información.



# Apéndice A

## Manuais técnicos

Este manual céntrase na explicación da estrutura da topoloxía de procesamento empregada para o procesado dos *tweets* en tempo real.

### A.1. A topoloxía

A clase da topoloxía empregada atópase no módulo `modulo_streaming`, na ruta `src/main/java/Topoloxia.java`. Esta comeza cunhas liñas iniciais para inicializar a clase para a análise de sentimento, e as seguintes inicializan os `Serdes` personalizados, empregados para a serialización e deserialización:

---

```
AnalizadorSentimentoIngles.init();

SerdesTwit serdesTwit = new SerdesTwit();
SerdesListaTwit serdesListaTwit = new SerdesListaTwit();
SerdesAnaliseSentimento serdesAnaliseSentimento = new
    SerdesAnaliseSentimento();
```

---

O nodo **source** créase mediante a seguinte sentenza:

---

```
// Fluxo inicial que obten todas mensaxes do topic
KStream<String, Twit> twitKStream = streamsBuilder.stream(
    Helper.topicStream());
```

---

Mediante o método `.stream()` do constructor do *stream* créase o primeiro nodo da topoloxía obtendo os datos do *topic* pasado como parámetro. Devolve un

obxecto to tipo `KStream` que ten como par chave-valor os tipos `String` e `Twit`.

A creación e concatenación de nodos ven dada polo emprego dos métodos destes obxectos. No seguinte trozo de código amósase o filtrado inicial de *tweets* duplicados, do seu gardado noutro *topic* e do filtrado de *tweets* con *hashtags*. Pódese observar como se aplican os métodos `groupByKey()`, `to()` e `filter()` para as accións mencionadas.

---

```
// Filtra os tweets que se repitan
KStream<String , Twit> tweetsNonRepetidos = twitKStream.
    groupByKey().reduce((aggVal , newVal) -> aggVal).toStream
    ();

// Almacena os tweets unicos nun topic separado
tweetsNonRepetidos.to(Helper.topictweetsUnicos);

// Fluxo que filtra aqueles tweets que tenen hashtags
KStream<String , Twit> tweetsConHashtag = tweetsNonRepetidos.
    filter((clave , twit) -> twit.obterHashtags().size() > 0);
```

---

A concatenación de distintos métodos pode dar lugar a outro tipo de obxectos, como pode ser `KTable`. Estas representan o fluxo de datos do `KStream` coas transformacións aplicadas cos distintos métodos, mediante táboas.

Na seguinte porción de código, recóllense os *tweets* en idioma inglés, agrúpanse e realízase unha **agregación** na cal se inicializa un obxecto `AnaliseSentimento` cuxos atributos de sentimento positivo, negativo e neutro serán 0, e que mediante a análise do texto do *tweet*, estes veranse incrementados. Mediante a materialización indícase que se desexa gardar a táboa en almacenamento persistente, e como o obxecto a gardar é da clase `AnaliseSentimento` o cal difire da clase inicial do *stream* `Twit`, é necesario indicar os `Serdes` para a serialización:

---

```
// Obtemos o analyse de sentimento dos tweets que mencionan
    cada hashtag en idioma ingles
KTable<String , AnaliseSentimento>
    analyseSentimentotweetsIngles = hashtagsStream.filter((
    hashtag , twit) -> twit.getLang().equals("en"))
    .groupByKey(Grouped.with(Serdes.String(), serdesTwit)).
    aggregate(
    AnaliseSentimento::new,
    (hashtag , twit , analyse) -> {
        String sentimento = AnalizadorSentimentoIngles.
            findSentiment(twit.getText());
        switch (sentimento) {
```

```

        case "Neutral": analyse.incrementarSentimentoNeutro
            (); break;
        case "Positive": analyse.
            incrementarSentimentoPositivo(); break;
        case "Negative": analyse.
            incrementarSentimentoNegativo(); break;
    }
    return analyse;
}, Materialized.<String, AnaliseSentimento, KeyValueStore<
    Bytes, byte[]>>as(Helper.storeSentimentoIngles)
    .withKeySerde(Serdes.String())
    .withValueSerde(serdesAnaliseSentimento));

```

---

A modificación dos **Serdes** indicando clases que non correspondan dará lugar a erros de incompatibilidade ao serializar/deserializar nos *topics*. O resto de nodos posúen unha estrutura semellante, polo que non se van a explicar, agás o de realizar a conta temporal.

Este último nodo, emprega a clase **Windowed<K>** na cal se indica o tipo da chave, neste caso o tipo será **String**. En lugar de aplicar métodos de transformación, emprégase o método de agrupación por ventás de tempo, **windowedBy()**, no que se indica a duración da ventá temporal, de 2 segundos:

```

// Agrupamos o Stream de hashtags por clave
// e agregamolos nunha taboa, levando a conta do numero de
// veces que aparece cada hashtag
KTable<Windowed<String>, Long> contaHashtagsWindowedKTable =
    hashtagsStream.groupByKey(Grouped.with(Serdes.String(),
        serdesTwit))
    .windowedBy(TimeWindows.of(Duration.ofSeconds(2))).count(
        Materialized.as(Helper.storeListadoHashtagWindow));

```

---

Estas chaves de tipo **Windowed<K>**, posúen, ademais do propio valor da chave, dous atributos especiais, nos que se indica o inicio e o final da ventá temporal, mediante marcas de tempo con precisión de milisegundos, que serán as que se empregan despois para realizar a gráfica co histórico de aparicións no último minuto.



# Apéndice B

## Manuais de usuario

### B.1. Preparativos iniciais

Antes de executar os distintos módulos, é necesario instalar diferentes paquetes para a execución dos servizos e a instalación de librerías de Python. Para iso, supoñendo que se está empregando unha máquina con sistema operativo Ubuntu 20.04, adxúntase o seguinte *script* `preparativos.sh`:

---

```
#!/bin/sh

sudo apt-get install docker docker-compose maven pip

pip install tweepy kafka-python flask apscheduler
```

---

, localizado na carpeta `preparativos` á que se debe acceder para executalo seguinte forma, asegurándose que posúe permisos de execución:

---

```
$ chmod +x ./preparativos.sh
$ ./preparativos.sh
```

---

### B.2. Despregue

Una vez instalados os paquetes, pódese proceder co arranque dos servizos de Kafka, ZooKeeper e Elasticsearch mediante un ficheiro `docker-compose.yml`. O seu contido é o seguinte:

```
version: '2'
services:
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch
      :7.4.0
    container_name: elasticsearch
    environment:
      - xpack.security.enabled=false
      - discovery.type=single-node
    ulimits:
      memlock:
        soft: -1
        hard: -1
      nofile:
        soft: 65536
        hard: 65536
    cap_add:
      - IPC_LOCK
    volumes:
      - elasticsearch-data:/usr/share/elasticsearch/data
    ports:
      - 9200:9200
      - 9300:9300

  zookeeper:
    container_name: zookeeper
    image: confluentinc/cp-zookeeper:6.1.1
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    container_name: kafka
    image: confluentinc/cp-kafka:6.1.1
    depends_on:
      - zookeeper
    ports:
      - 9092:9092
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,
        PLAINTEXT_HOST://localhost:9092
```

```
KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:
PLAINTEXT, PLAINTEXT_HOST: PLAINTEXT
KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
KAFKA_MESSAGE_MAX_BYTES: 2097152
```

```
volumes:
  elasticsearch-data:
    driver: local
```

Para iniciar os contedores, execútase o seguinte comando:

```
$ sudo docker-compose up
```

Coa primeira execución decárganse as imaxes que se van e empregar. Unha vez rematadas as descargas, lanzaranse os contedores. Para comprobar que os contedores están correndo, ábrese unha nova terminal e sitúase na mesma ruta onde está o ficheiro `docker-compose.yml` e lánzase o seguinte comando:

```
$ sudo docker-compose ps
```

, que debe producir unha saída **igual** á seguinte:

Name	Command Ports	State
elasticsearch	/usr/local/bin/docker-entr ... 0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp	Up
kafka	/etc/confluent/docker/run 0.0.0.0:9092->9092/tcp	Up
zookeeper	/etc/confluent/docker/run 2181/tcp, 2888/tcp, 3888/tcp	Up

Unha vez comprobado que os contedores están correndo, pódese proceder coa execución das distintas aplicacións.

**Execución do módulo de obtención de *tweets*** Accédese á carpeta `modulo_obtencion` e execútase o comando:

```
$ python3 obter_twits.py
```

---

O programa creará os *topics* no servizo de Kafka e preguntará sobre os temas de interese dos *tweets* a buscar. Unha vez introducidos, empezará a descargalos e envíalos ao *broker*. Amósase unha execución de exemplo:

---

```
Creanse os topics...
Indica os temas de interese separados con coma:spain,
    coronavirus,usa
Temas de interese:
['spain', 'coronavirus', 'usa']
Enviado twit a offset: 0
Enviado twit a offset: 1
Enviado twit a offset: 2
Enviado twit a offset: 3
Enviado twit a offset: 4
Enviado twit a offset: 5
Enviado twit a offset: 6
Enviado twit a offset: 7
Enviado twit a offset: 8
Enviado twit a offset: 9
Enviado twit a offset: 10
Enviado twit a offset: 11
Enviado twit a offset: 12
Enviado twit a offset: 13
Enviado twit a offset: 14
Enviado twit a offset: 15
Enviado twit a offset: 16
Enviado twit a offset: 17
Enviado twit a offset: 18
Enviado twit a offset: 19
Enviado twit a offset: 20
```

---

O número de offset indica que a deposición do *tweet* no *broker* levouse a cabo correctamente, ademais de indicar o número de *tweets* enviados.

**Execución do módulo de *streaming*** Abrindo outra terminal, sitúase dentro da carpeta `modulo_streaming`. Unha vez aquí, emprégase maven para lanzar a aplicación directamente, da seguinte forma:

---

```
$ mvn exec:java
```

---



A saída da execución serán mensaxes informativas da seguinte forma:

---

```

2021-07-26 11:43:09 INFO StreamTask:251 - stream-thread [
    twit-streams-b364ba2d-265b-4767-9600-39d7a008fc32-
    StreamThread-1] task [4_0] Restored and ready to run
2021-07-26 11:43:09 INFO ConsumerCoordinator:1354 - [
    Consumer clientId=twit-streams-b364ba2d-265b-4767-9600-39
    d7a008fc32-StreamThread-1-consumer, groupId=twit-streams]
    Found no committed offset for partition twit-streams-
    conta-hashtagwindow-repartition-0
2021-07-26 11:43:09 INFO StreamTask:251 - stream-thread [
    twit-streams-b364ba2d-265b-4767-9600-39d7a008fc32-
    StreamThread-1] task [5_0] Restored and ready to run
2021-07-26 11:43:09 INFO StreamThread:851 - stream-thread [
    twit-streams-b364ba2d-265b-4767-9600-39d7a008fc32-
    StreamThread-1] Restoration took 477 ms for all tasks [0
    _0, 1_0, 2_0, 3_0, 4_0, 5_0]
2021-07-26 11:43:09 INFO StreamThread:230 - stream-thread [
    twit-streams-b364ba2d-265b-4767-9600-39d7a008fc32-
    StreamThread-1] State transition from PARTITIONS_ASSIGNED
    to RUNNING
2021-07-26 11:43:09 INFO KafkaStreams:320 - stream-client [
    twit-streams-b364ba2d-265b-4767-9600-39d7a008fc32] State
    transition from REBALANCING to RUNNING
2021-07-26 11:43:10 WARN NetworkClient:1100 - [Producer
    clientId=twit-streams-b364ba2d-265b-4767-9600-39
    d7a008fc32-StreamThread-1-producer] Error while fetching
    metadata with correlation id 4 : {twits-unicos=
    LEADER_NOT_AVAILABLE}
2021-07-26 11:43:27 INFO RocksDBTimestampedStore:100 -
    Opening store conta-hashtagwindow.1627300800000 in
    regular mode

```

---

**Execución do módulo de almacenamento** Abrindo outra terminal, sitúase dentro da carpeta `modulo_almacenamento`. Unha vez aquí, lánzase a aplicación da mesma forma que a de *streaming*:

---

```
$ mvn exec:java
```

---

Cuxa saída será semellante á que se amosa a continuación, indicando que se están almacenando os *tweets* na base de datos de Elasticsearch:

---

```
Twit 1419730829517930500 almacenado.
```

---

```

Twit 1419730831090896902 almacenado.
Twit 1419730831283822592 almacenado.
Twit 1419730831334064132 almacenado.
Twit 1419730831631962113 almacenado.
Twit 1419730831854211078 almacenado.
Twit 1419730831900282884 almacenado.
Twit 1419730832395227142 almacenado.
Twit 1419730832659472391 almacenado.

```

---

**Execución do módulo web** Por último, ábrese unha última terminal, sitúase dentro da carpeta `modulo_web` e lánzase a aplicación Flask `app.py`:

---

```
$ python3 app.py
```

---

A saída deste último será exactamente igual que a seguinte:

---

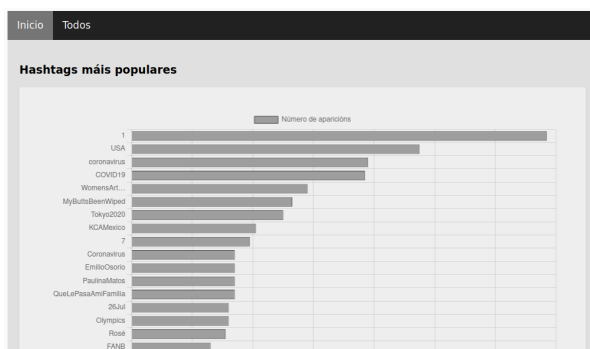
```

* Serving Flask app 'app' (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a
production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://localhost:5500/ (Press CTRL+C to quit)

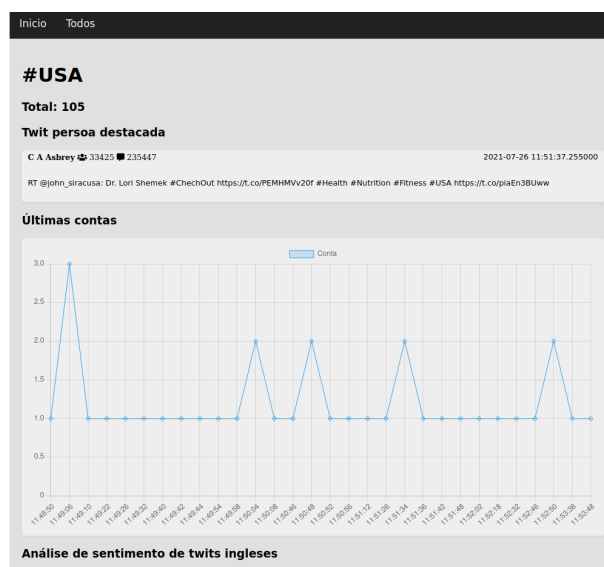
```

---

**Visualización de resultados** Para ver os resultados do procesamento en tempo real, ábrese un navegador web, neste caso Firefox, e accédese á url `http://localhost:5500`. Aquí observarase unha gráfica que amosará de forma descendente os *hashtags* procesados e a súa conta:



Movendo o cursor por encima de cada barra amosará o número exacto da conta de cada *hashtag*. Premendo na barra, redirixirá á vista de detalles do *hashtag* seleccionado, onde se poderán ver os resultados do procesamento en tempo real:



Premendo na xanela Todos pasarase á vista global de *hashtags* procesados, na cal, facendo click nun deles, redirixirá á vista de detalles mencionada anteriormente:

Inicio

Todos

Total de hashtags: 1536

1	USA	coronavirus	COVID19
182	126	99	91
MyButtsBeenWiped	WomensArt...	7	Tokyo2020
83	65	59	58
KCAMexico	26jul	Coronavirus	FANB
56	53	49	43
GNB	EmilioOsorio	Olympics	PaulinaMatos
43	39	39	39
QueLePasaAmiFamilia	Rosé	GNBGDP...	usa
39	39	29	29
2	5	7Más7EquilibrioSaludable	PREVENCIÓN
21	20	20	20
COVID19.	CyberArmyofEthiopia...	Ethiopians	TokyoOlympics
19	18	18	18
KISS_OR_DEATH	Putin	iTunes	6
15	15	15	14
LakeShow	essayhelp	7Más7EquilibrioSaluda...	EscuadrónSaludable,
14	14	13	13
Florida	11	16	21
13	12	12	12
32	Florida.	Good4U	Morat
12	12	12	12
covid19	ÚltimoMinuto]	7Más7EquilibrioSaludable...	Covid_19
12	12	11	11
Cuba	Spain	SusanaCala	28
11	11	11	10

**Parar a execución** Para parar a execución tanto dos contedores coma dos módulos, basta con desprazarse polas terminais nas que se executan as aplicacións e realizar a combinación de teclado **Ctrl-C**.

### B.3. Posibles erros

Se se executan as aplicacións antes de lanzar os contedores obteranse erros de conexión co *broker* tales coma os seguintes que se amosan a continuación, nas execucións do módulo de obter *tweets* e *streaming*:

---

```
kafka.errors.NoBrokersAvailable: NoBrokersAvailable
```

---

---

```
Connection to node -1 (localhost/127.0.0.1:9092) could not  
be established. Broker may not be available.
```

---

Por tanto, hai que asegurarse de que os servizos estean activos antes de lanzar os módulos.

Ademais, o módulo web obtén a información do módulo de *streaming*, polo que se este non se encontra en execución, amosaranse erros do seguinte tipo:

---

```
Failed to establish a new connection: [Errno 111] Connection  
refused '))
```

---

# Bibliografía

- [1] Jay Kreps. *I Heart Logs: Event Data, Stream Processing, and Data Integration*. O'Reilly Media, Inc.
- [2] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, Jennifer Widom. *Models and Issues in Data Stream Systems*. Disponible en [https://infolab.usc.edu/csci599/Fall2002/paper/DML2\\_streams-issues.pdf](https://infolab.usc.edu/csci599/Fall2002/paper/DML2_streams-issues.pdf) [27/07/2021].
- [3] Jay Kreps. *Questioning the Lambda Architecture*. <https://www.oreilly.com/radar/questioning-the-lambda-architecture/> [27/07/2021].
- [4] Apache Software Foundation. <https://kafka.apache.org>. [27/07/2021].
- [5] Apache Software Foundation. <https://kafka.apache.org/documentation/streams/>. [27/07/2021].
- [6] Facebook, Inc. <https://rocksdb.org>. [27/07/2021].
- [7] Elasticsearch B.V. <https://www.elastic.co/es/what-is/elasticsearch>. [27/07/2021].
- [8] Apache Software Foundation. <https://pulsar.apache.org>. [27/07/2021].
- [9] VMWare, Inc. <https://www.rabbitmq.com>. [27/07/2021].
- [10] Alok Nikhil, Vinodh Chandar. *Benchmarking Apache Kafka, Apache Pulsar, and RabbitMQ: Which is the Fastest?* <https://www.confluent.io/blog/kafka-fastest-messaging-system/>. [27/07/2021].
- [11] Enlyft. *Companies using Apache Kafka*. <https://enlyft.com/tech/products/apache-kafka>. [27/07/2021].
- [12] Ola Puchta-Górska. *Facts about Kafka every business should know*. <https://blog.softwaremill.com/facts-about-kafka-every-business-should-know-1249a3b9db74>. [27/07/2021].

- [13] Twitter, Inc. Twitter API. <https://developer.twitter.com/en/docs/twitter-api>. [27/07/2021].
- [14] Martin Kleppmann. *Designing Data-Intensive Applications the big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, Inc. pp. 497-498.
- [15] Joshua Roesslein. <https://www.tweepy.org>. [27/07/2021].
- [16] Dana Powers, David Arthur, and Contributors. <https://kafka-python.readthedocs.io/en/master/index.html>. [27/07/2021].
- [17] Emil Koutanov. *Effective Kafka A Hands-On Guide to Building Robust and Scalable Event-Driven Applications with Code Examples in Java*. Leanpub.
- [18] William P. Bejeck Jr.. *Kafka Streams in Action Real-time apps and micro-services with the Kafka Streams API*. Manning.
- [19] Alexander Dean, Valentin Crettaz. *Event Streams in Action Real-time event systems with Kafka and Kinesis*. Manning.
- [20] Confluent, Inc.. *Streams Concepts*. <https://docs.confluent.io/platform/current/streams/concepts.html#stateful-stream-processing>. [26/07/2021].
- [21] Michael Noll. *Streams and Tables in Apache Kafka: A Primer*. <https://www.confluent.io/blog/kafka-streams-tables-part-1-event-streaming/>. [26/7/2021].
- [22] Confluent, Inc.. *Streams DSL: Windowing*. <https://docs.confluent.io/platform/current/streams/developer-guide/dsl-api.html#windowing>. [26/07/2021].
- [23] Stanford NLP Group. <https://stanfordnlp.github.io/CoreNLP/>. [27/07/2021].
- [24] Eclipse Foundation. <https://www.eclipse.org/jetty/>. [27/07/2021].
- [25] Oracle Corporation and/or its affiliates. JAX-RS: The Java™ API for RESTful Web Services. <https://jcp.org/en/jsr/detail?id=311>. [27/07/2021].
- [26] Confluent Inc. *Kafka Streams Interactive Queries*. <https://docs.confluent.io/platform/current/streams/developer-guide/interactive-queries.html>. [27/07/2021].
- [27] Neha Narkhede, Gwen Shapira, Todd Palino. *Kafka: The Definitive Guide, Real-time data and stream processing at scale*. O'Reilly Media, Inc. Capítulo 2: *Meet Kafka*.

- [28] Pallets. *Flask web development, one drop at a time*. <https://flask.palletsprojects.com/en/2.0.x/>. [27/07/2021].
- [29] Miguel Grinberg. *Flask Web Development, 2nd Edition*. O'Reilly Media, Inc.
- [30] Alex Grönholm. *Advanced Python Scheduler*. <https://apscheduler.readthedocs.io/en/stable/>. [27/07/2021].
- [31] Pallets. *Jinja*. <https://jinja.palletsprojects.com/en/3.0.x/>. [27/07/2021].
- [32] OpenJS Foundation e contribuidores de JQuery. *JQuery write less, do more*. <https://jquery.com>. [27/07/2021].
- [33] Contribuidores de Chart.js. *Chart.js*. <https://www.chartjs.org>. [27/07/2021].
- [34] Apache Software Foundation. *Welcome to Apache ZooKeeper™*. <https://zookeeper.apache.org>. [27/07/2021].
- [35] Docker, Inc. *Docker: Empowering App Development for Developers*. <https://www.docker.com>. [27/07/2021].
- [36] Docker, Inc. *Overview of Docker Compose*. <https://docs.docker.com/compose/>. [27/07/2021].
- [37] Apache Software Foundation. *Apache Maven Project*. <https://maven.apache.org>. [27/07/2021].
- [38] The JUnit Team. *JUnit 5*. <https://junit.org/junit5/>. [27/07/2021].
- [39] Contribuidores de AssertJ. *AssertJ - fluent assertions java library*. <https://assertj.github.io/doc/>. [27/07/2021].
- [40] Python Software Foundation. *unittest — Unit testing framework*. <https://docs.python.org/3/library/unittest.html>. [27/07/2021].