TRABAJO DE FIN DE MÁSTER

Arquitectura predictiva en Streaming para Stocks

Ramón Serrano López

Tutor: Víctor Marcos Martín

Máster: Data Science & Big Data

Convocatoria: Noviembre 2019

Agradecimientos

” A mis padres, a mi familia, a mi prima, a mi tutor, a mis amigos, a mis compañeros de la carrera y de trabajo, con especial mención a mi pareja, la persona más maravillosa del mundo, sin todos ellos no sería lo que hoy soy, tan solo puedo expresar mi mas sincero agradecimiento por apoyarme durante esta etapa que ha representado el máster y en general la vida.”

Resumen

La cantidad de información que se genera cada día en el mundo va creciendo cada vez más y más de forma exponencial. Estos datos si se llegan a explotar, aportarían un gran valor a la sociedad. Con la llegada de estos volúmenes de información, se empieza a plantear el uso de tecnologías que sean capaces de realizar múltiples análisis en tiempo real. La corriente hace unos años era analizar todos los datos que residían en tu plataforma con un proceso batch, ahora se opta por la transformación, análisis y visualización en tiempo real. Esto permite tomar decisiones de negocio que aportan un gran valor, este trabajo va sobre la construcción de una arquitectura que permita esto.

Abstract

The ammount of data generated daily rises exponentially. If correctly analysed, it can bring a great value to society. With the rise of this data volumes accessibility, new real time analytics technologies start to emerge. A few years back, one would batch analyse that data, now, we tend to transform, analyse and visualize in real time that data. This enables new business logic which in turn provide grate value. This project attempts to create such architecture.

Índice

Índice de figuras

1. Introducción

Dado el auge de las tecnologías Big Data, se han empezado a desarrollar soluciones en tiempo real que permitan manejar diferentes fuentes de datos, permitiendo además, un procesamiento y análisis de forma simultanea.

Este trabajo de Fin de Máster surge en base a esa idea y trata sobre la realización de una arquitectura *Big Data* que tenga la capacidad de producir, analizar y visualizar datos en *streaming*, y no, sobre la realización del mejor modelo de machine learning para este caso de uso.

1.1. Motivación

El caso de uso propuesto por el tutor, impulsó al alumno a la realización de este Trabajo de Fin de Máster, donde plantea la definición de un sistema capaz de procesar datos socio económicos (mercados de *stocks*) e intente predecir si el precio de cierre de un *stock* va a bajar o a subir.

Las posibilidades de este tipo de sistema en la vida real son muy grandes, ya que permite aplicar sobre un flujo de datos masivo un algoritmo capaz más allá de un simple sistema experto, de realizar predicciones bastante certeras con una alta frecuencia de datos.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar la arquitectura que cumpla los requisitos del caso de uso planteado en la motivación. Además, alrededor de esta, se han planteado unos objetivos opcionales para la mejora del trabajo.

La idea general del proyecto plantea los siguientes objetivos:

- **Preparación de los datos** para su posterior producción
- **Capacidad de producir datos** del precio de cierre de cada *stock*
- **Recoger estos datos de *stock*** y crear una ventana de precios
- **Crear un modelo de *machine learning*** que sea capaz de predecir en base a una ventana de precios
- **Hacer uso del modelo en una sistema de procesamiento en *streaming*** para poder predecir las ventanas de precios si la acción del *stock* va a subir o va a bajar
- **Ingesta y visualización** de *stocks* con sus predicciones

2. Datos

Los datos provienen de Kaggle ”Huge Stock Market Datase”. Son un total de 7.195 *stocks* de Estados Unidos, cada uno con su CSV. En este *dataset* vienen todos los históricos diarios con datos de precio y volúmenes de cada *stock*. Este *dataset* es bastante popular en Kaggle y se usa principalmente para ver quién es capaz de hacer el mejor modelo a la hora de predecir el precio de las acciones.

2.1. Estructura de los datos

La cabecera de este CSV la forman las siguientes columnas:

- ***Date***: fecha del *record*
- ***Open***: precio de apertura
- ***High***: precio máximo llegado
- ***Low***: precio mínimo llegado
- ***Close***: precio de cierre
- ***Volume***: volumen del *stock*
- **OpenInt**

2.2. Procesado de los datos

Para este proyecto se ha tenido que transformar con la ayuda de Python y Spark el *dataset* original (se usará en la producción de datos), para poder constituirlo unificado y ordenado por fecha con la misma estructura, pero con un campo más ”Stock”que es el nombre del *stock*.

El *dataset* resultante tiene 14.887.665 *records* y ocupa 1,07 GB en disco y quedaría con la siguiente forma.

data.csv							
Date	Open	High	Low	Close	Volume	OpenInt	Stock
1962-01-02T00:00:00.000+01:00	0.6277	0.6362	0.6201	0.6201	2575579	0	ge.us
1962-01-02T00:00:00.000+01:00	6.413	6.413	6.3378	6.3378	467056	0	ibm.us
1962-01-03T00:00:00.000+01:00	0.6201	0.6201	0.6122	0.6201	1764749	0	ge.us
1962-01-03T00:00:00.000+01:00	6.3378	6.3963	6.3378	6.3963	350294	0	ibm.us
1962-01-04T00:00:00.000+01:00	0.6201	0.6201	0.6037	0.6122	2194010	0	ge.us
1962-01-04T00:00:00.000+01:00	6.3963	6.3963	6.3295	6.3295	314365	0	ibm.us
1962-01-05T00:00:00.000+01:00	0.6122	0.6122	0.5798	0.5957	3255244	0	ge.us
1962-01-05T00:00:00.000+01:00	6.3211	6.3211	6.1958	6.2041	440112	0	ibm.us
1962-01-08T00:00:00.000+01:00	0.5957	0.5957	0.5716	0.5957	3696430	0	ge.us
1962-01-08T00:00:00.000+01:00	6.2041	6.2041	6.0373	6.087	655676	0	ibm.us

3. Arquitectura Big Data

Se ha planteado una arquitectura en *streaming* que sea capaz de predecir y visualizar los resultados de los *stocks* en tiempo real.

En el siguiente diagrama, se puede ver el flujo que siguen los datos:

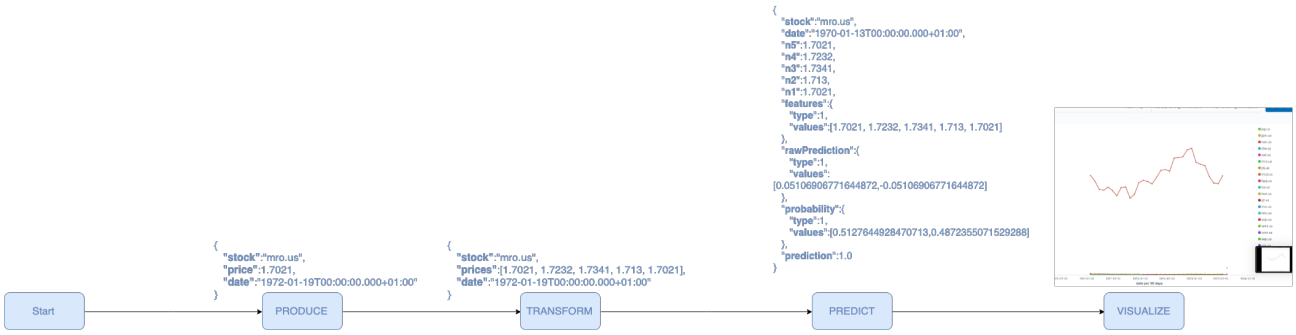


Figura 1: Diagrama de flujo de datos

Las etapas por las que pasa el dato son las siguientes:

- Se envían/producen los datos con el módulo Producer
- Estos datos se transforman con el módulo del Microservice Collector
- Se realizan predicciones en base a las transformaciones con el módulo de Spark Streaming que carga el modelo de Machine Learning
- Por último, se visualizan los datos con sus predicciones con el módulo de ELK Stack

3.1. Tecnologías y herramientas usadas

En esta sección se va a realizar una pequeña introducción sobre las tecnologías y herramientas que se han usado para desarrollar la tecnología de este proyecto.

3.1.1. Docker

Es una tecnología que usa el *kernel* de Linux y las funciones de este para crear procesos que se puedan ejecutar de manera independiente. Este desarrolla *software* a nivel del virtualización de sistema operativo en unos paquetes llamados *containers* que incluyen en ellos todo lo necesario (librerías) para que el *software* se ejecute. Todos estos están hosteados por el Docker Engine. Posee una amplia comunidad *opensource* y un repositorio donde las organizaciones o individuos pueden subir sus propias imágenes para que estén a disposición de la comunidad.

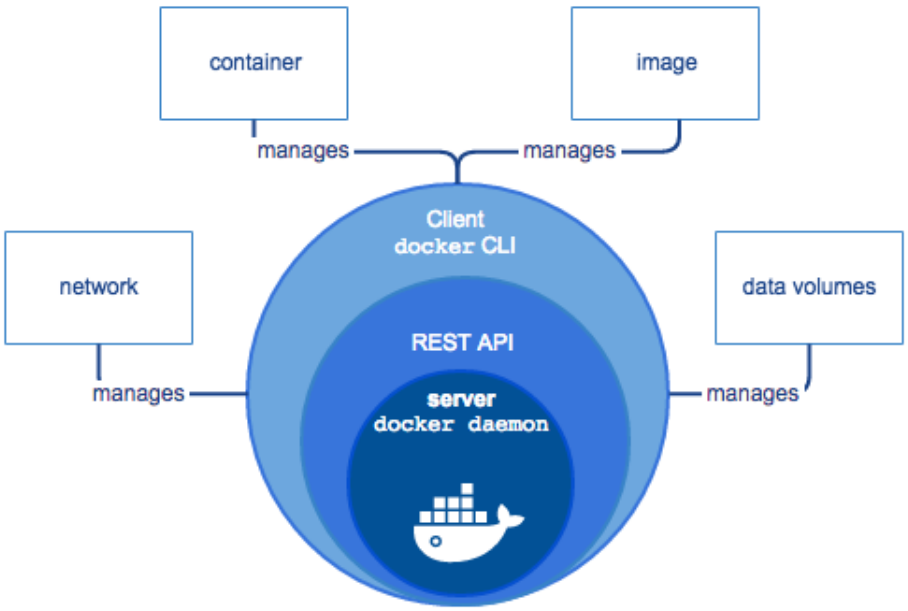


Figura 2: Arquitectura Docker

3.1.2. Apache Kafka

Es una plataforma *online* distribuida que tiene tres capacidades clave:

- Permitir la publicación y suscripción a flujos de datos. Es muy similar a una cola de mensajes
- Guardar los flujos de datos con tolerancia a fallos
- Procesar los flujos de datos en *streaming*

Generalmente se usa para dos tipos de caso de uso:

- Creación de flujos de datos *online* entre sistemas y aplicaciones
- Creación de aplicaciones *real time* que transforman o reaccionan a los flujos de datos

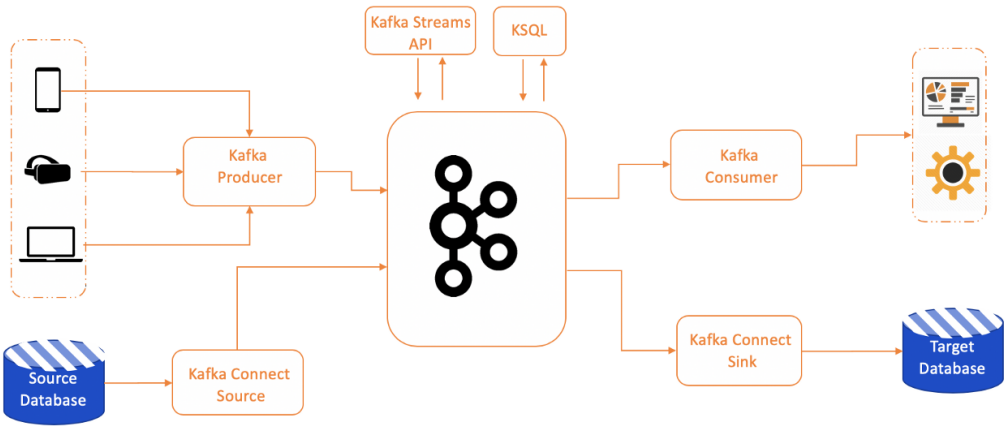


Figura 3: Arquitectura Kafka

3.1.3. Apache Spark

Es un motor de procesamiento clusterizado de propósito general muy rápido, basado en su antecesor Hadoop Map Reduce. Permite dividir y paralelizar el trabajo; es muy escalable, las tareas de procesamiento se reparten entre todas las máquinas del *cluster* y provee unas APIs de alto nivel en Scala, Java, R y Python. Igualmente, posee un motor optimizado capaz de ejecutar operaciones en grafos; tiene un lenguaje SQL llamado Spark Sql para el procesamiento de datos; dos versiones de procesamiento (Batch y Streaming) y una librería de *machine learning* MLlib. En el proyecto se usa la 2.4.4.

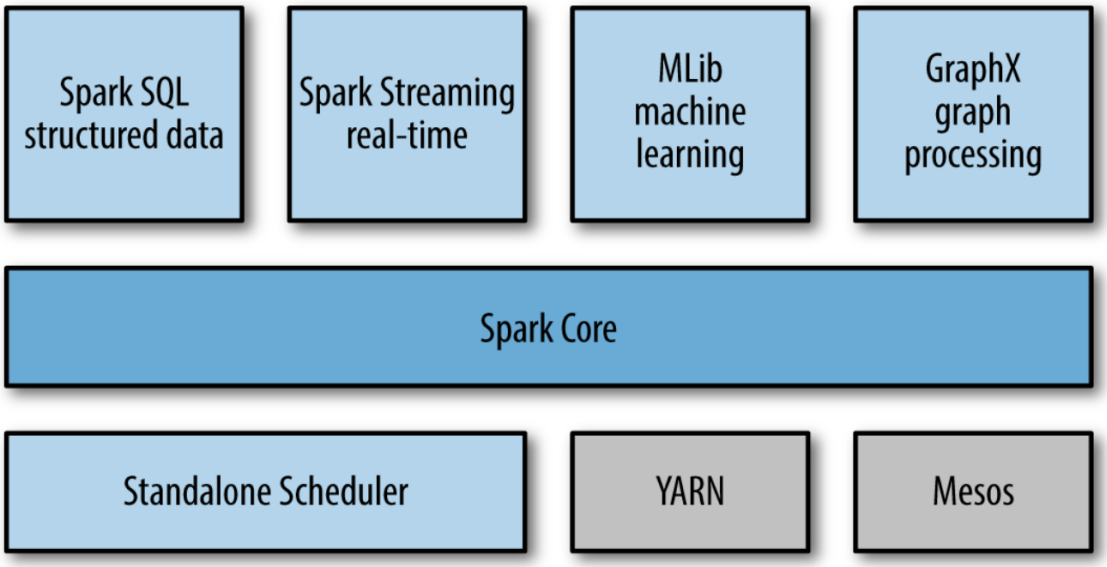


Figura 4: Arquitectura Spark

3.1.4. Apache Hadoop / HDFS

La librería de Apache Hadoop es un *framework* que permite el procesamiento distribuido de conjuntos de datos muy grandes a través de unos *clusters* de ordenadores usando simples modelos de programación. Está diseñado para escalar de un solo servidor a miles de máquinas, cada una ofreciendo su potencia de cómputo y su almacenamiento.

Hadoop Distributed File System (HDFS) es un sistema de ficheros distribuido di-

señado para ejecutar en el *hardware* más básico. Trabaja bien con grandes volúmenes de datos, proporciona un alto rendimiento en el acceso y escritura de los datos, así como ofrece alta disponibilidad y tolerancia a fallos debido a su replicación de datos.

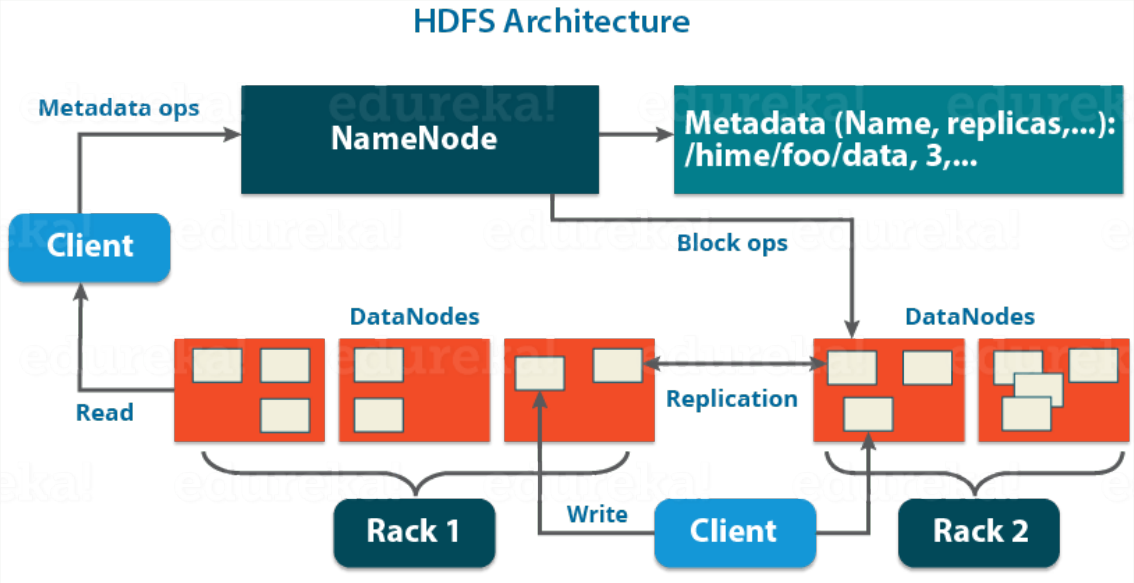


Figura 5: Arquitectur HDFs

3.1.5. Stack ELK

Es un *stack* compuesto por las siguientes tecnologías:

- **Logstash**: preprocesa los datos antes de ser enviados a ElasticSearch (*output*). Tiene varios posibles *inputs*, filtros, *plugins*, etc.
- **ElasticSearch**: base de datos distribuida. Distribuye la información en todos los nodos, por tanto es tolerante a fallos y tiene alta disponibilidad
- **Kibana**: permite la visualización y creación de consultas sobre los datos indexados en ElasticSearch

Posee las siguientes características:

- Es un gestor de datos
- Tiene un almacenamiento distribuido en ElasticSearch
- Procesamiento de la información con Logstash.
- Consultas y visualizaciones en tiempo real con Kibana

Intenta resolver la falta de consistencia, como el formato de tiempo, es descentralizado y te abstrae de la complejidad del formato de *logs*.

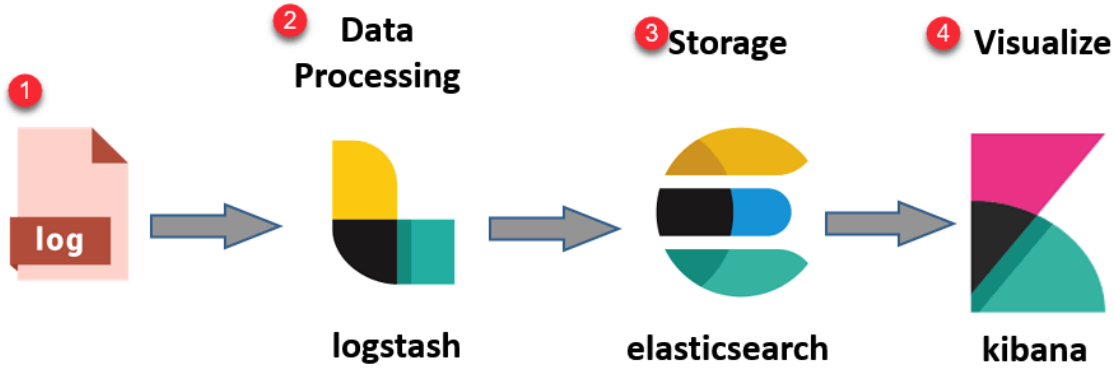


Figura 6: Arquitectura ELK

3.1.6. Jupyter Notebook

Aplicación cliente servidor que proporciona un entorno de trabajo con múltiples posibilidades a los desarrolladores de Python y R. El documento está formado por bloques que pueden albergar diferentes elementos (texto, *markdown*, LaTeX, R, Python, vídeos e imágenes, entre otros).

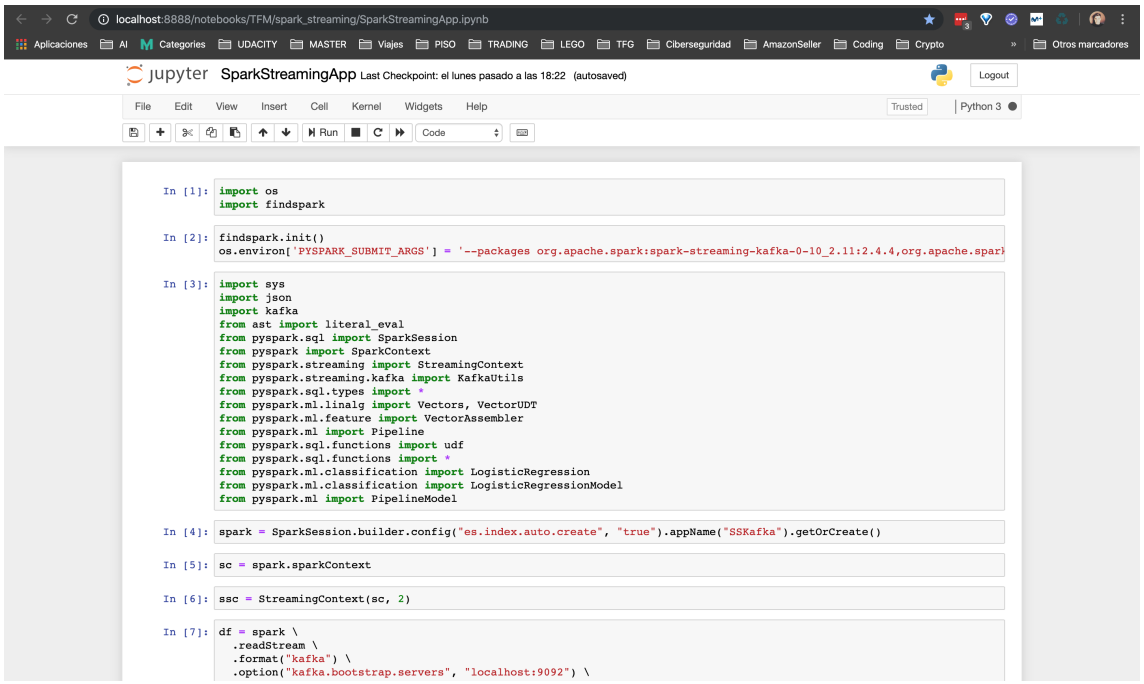


Figura 7: Uso de Jupyter Notebook en Local

3.2. Diseño

La arquitectura en *streaming* permite el procesado y visualización de los datos inyectados en el sistema y consiste en su mayoría en *containers* de Docker que se comunican a través de la publicación/suscripción al *broker* de mensajes. Esto hace que todos los módulos no solo se puedan lanzar en cualquier sistema sino que también desacopla la lógica entre ellos, pudiendo así escalar más fácil el sistema con unos simple ajustes.

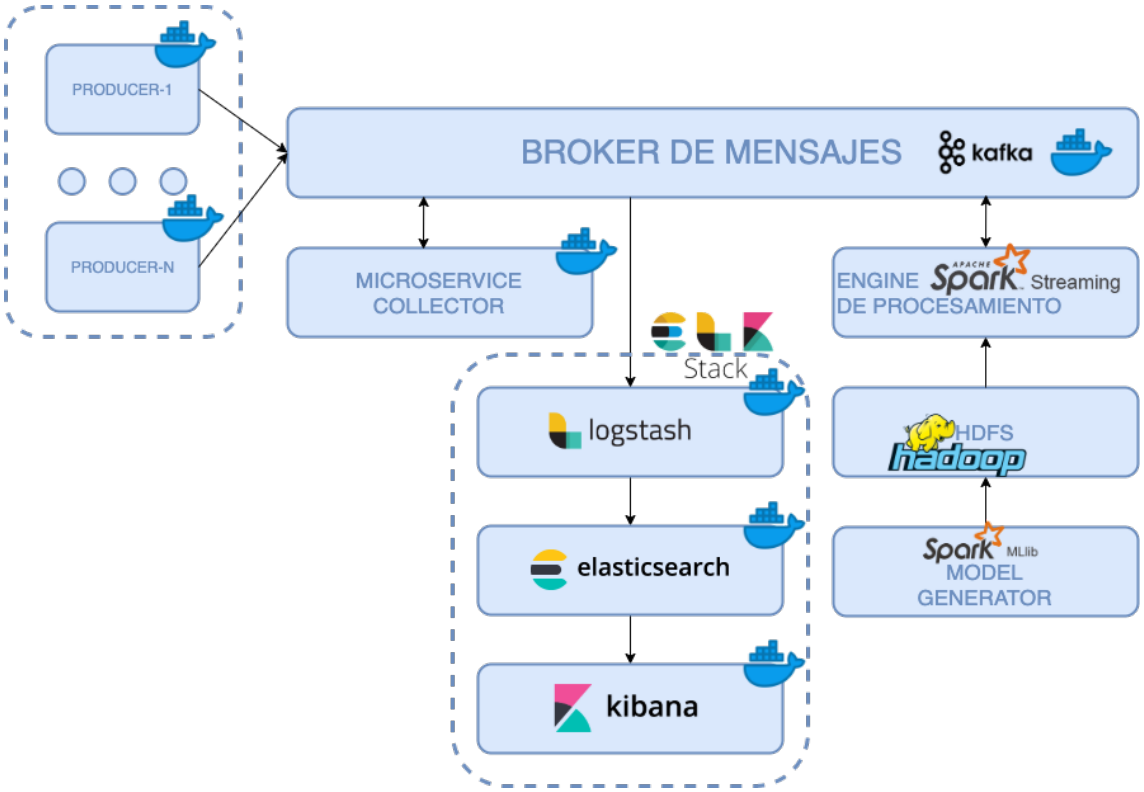


Figura 8: Diagrama de Arquitectura en *streaming* del sistema

3.3. Módulos

El sistema está compuesto de los siguientes módulos:

3.3.1. Producer

Es el módulo que se encarga de enviar los datos para emular la emisión de datos de *stock* en tiempo real. Este módulo dockerizado tiene un *script* de Python que se encarga de leer el *dataset* línea a línea, creando un json con las siguientes propiedades [*stock*, *price* y *date*] y enviándolo al *topic* "test"del *cluster* de Kafka.

Figura 9: *Topic* test

3.3.2. Microservice Collector

Es el módulo que se encarga de la preparación del dato enviado por el Producer para el módulo de Predicción en *Streaming*. Este módulo dockerizado tiene un *script* de Pyhton que lee del *topic* generado por el *producer*, agregando una ventana de los últimos cinco precios por *stock*, y envía un json al *topic* "test_agg"de Kafka con esta estructura [*stock*, *price*, *date* (fecha del precio más reciente)].

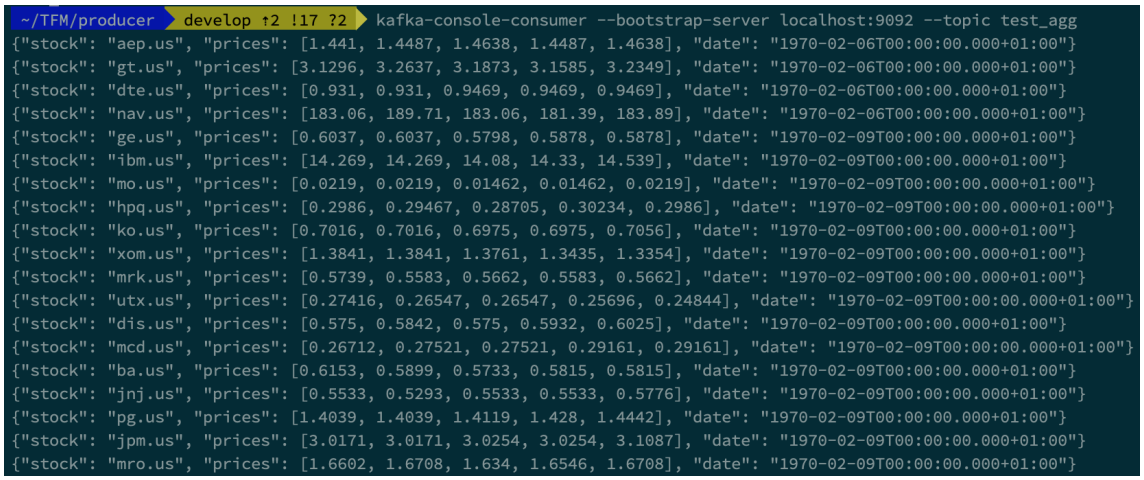


Figura 10: *Topic* test_agg

3.3.3. Spark Streaming

Es el módulo que se encarga de predecir los datos enviados por el Microservice Collector. Este módulo no se ha dockerizado, está en local, pero se podría hacer sin ninguna dificultad. Se encarga de leer el *topic* de agregación de precios, carga el modelo de *machine learning* previamente entrenado y realiza una predicción sobre él añadiendo unos datos, como la predicción y su probabilidad. Al igual que en los casos anteriores, todo esto se envía al *topic* de ”testoutput”.

Figura 11: *Topic* testoutput

3.3.4. ELK

Este módulo se encarga de escuchar el *topic* ”testoutput”(generado por el módulo de Predicción en *Streaming*), procesarlo, indexarlo y realizar visualizaciones en base a los

datos de los *stocks* y sus predicciones.

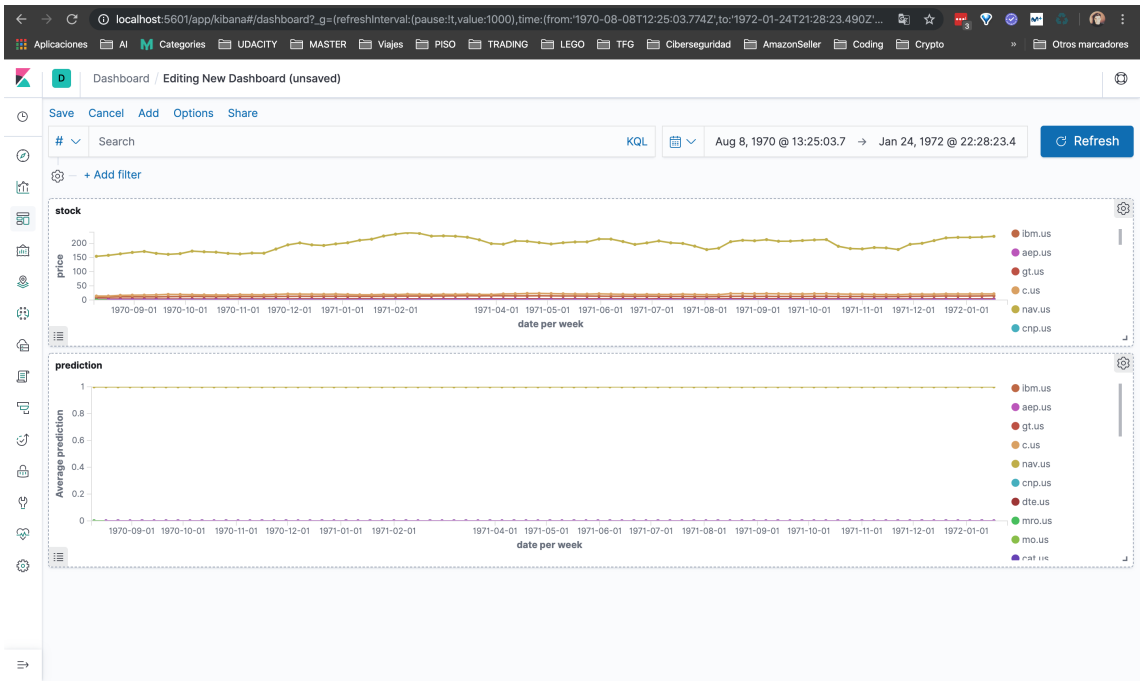


Figura 12: Visualización de *Stocks* con sus predicciones

3.3.5. HDFS

No está dockerizado, está en local y es donde se guarda el modelo previamente entrenado.

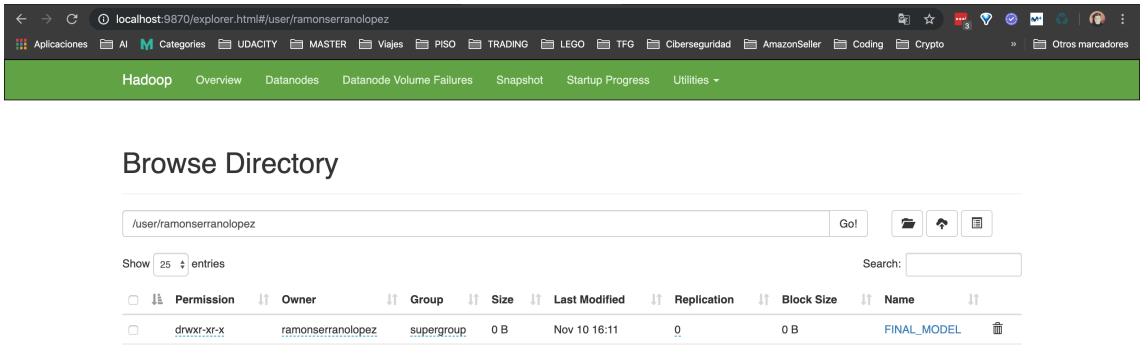
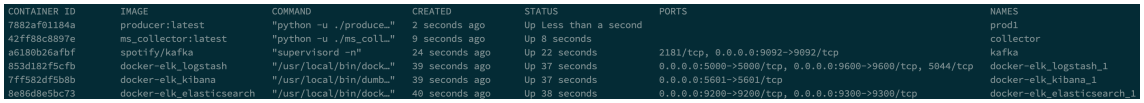


Figura 13: Modelo guardado en HDFS

3.4. Despliegue de infraestructura

Todos los componentes, exceptuando HDFS, Spark Streaming y el generador del modelo, están dockerizados en la misma red virtual creada en Docker llamada ”tfm”. Cada uno de los módulos posee un *hostname* para que sean más accesibles dentro de la red (kafka, producer, collector, elasticsearch, logstash y kinana).

La máquina que hostea todo el sistema es el portátil del alumno, que tiene 8 cores y 16GB de RAM.



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
7082af01184a	producer:latest	"python -u ./produce…"	2 seconds ago	Up Less than a second		prod1
42ff88c8897e	ms_collector:latest	"python -u ./ms_coll…"	9 seconds ago	Up 8 seconds		collector
a6180b26afbf	spotify/kafka	"supervisord -n"	24 seconds ago	Up 22 seconds	2181/tcp, 0.0.0.0:9092->9092/tcp	kafka
853d182f5cfb	docker-elk_logstash	"/usr/local/bin/dock…"	39 seconds ago	Up 37 seconds	0.0.0.0:5000->5000/tcp, 0.0.0.0:9600->9600/tcp, 5044/tcp	docker-elk_logstash_1
7ff582df5b8b	docker-elk_kibana	"/usr/local/bin/dumb…"	39 seconds ago	Up 37 seconds	0.0.0.0:5601->5601/tcp	docker-elk_kibana_1
8e86d8e5bc73	docker-elk_elasticsearch	"/usr/local/bin/dock…"	40 seconds ago	Up 38 seconds	0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp	docker-elk_elasticsearch_1

Figura 14: Docker containers

3.4.1. Producer y Microservice Collector

Se despliegan en contenedores de Docker y se encargan de consumir y/o producir a sus respectivos *topics* de Kafka (test y test_agg).

3.4.2. Apache Kafka

Un único contenedor que posee un *broker* de Kafka y un *zookeper*. Se podría haber optado por dos contenedores para cada uno de estos, pero se ha tomado la decisión para poder simplificar más el sistema.

3.4.3. Apache Spark

Está en local y el desarrollo, tanto de la parte de Spark Streaming como la del modelo, se hace con PySpark en un Jupyter Notebook (Anaconda 3). La versión del sistema es la 2.4.4 para Apache Hadoop 2.7 en adelante.

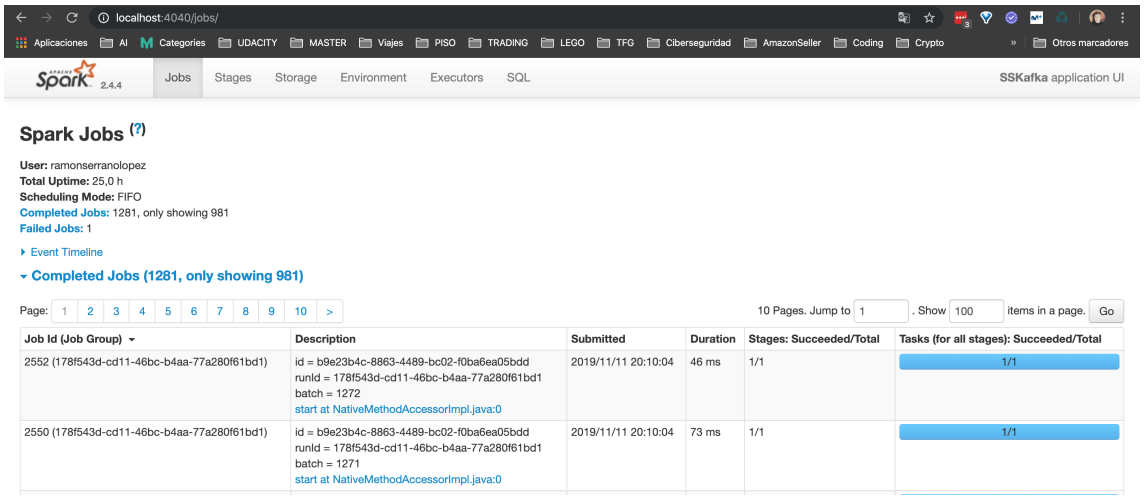


Figura 15: Spark UI

3.4.4. Apache Hadoop / HDFS

Instalado y configurado en local, un *datanode*, un *namenode* y un *secondary namenode*.

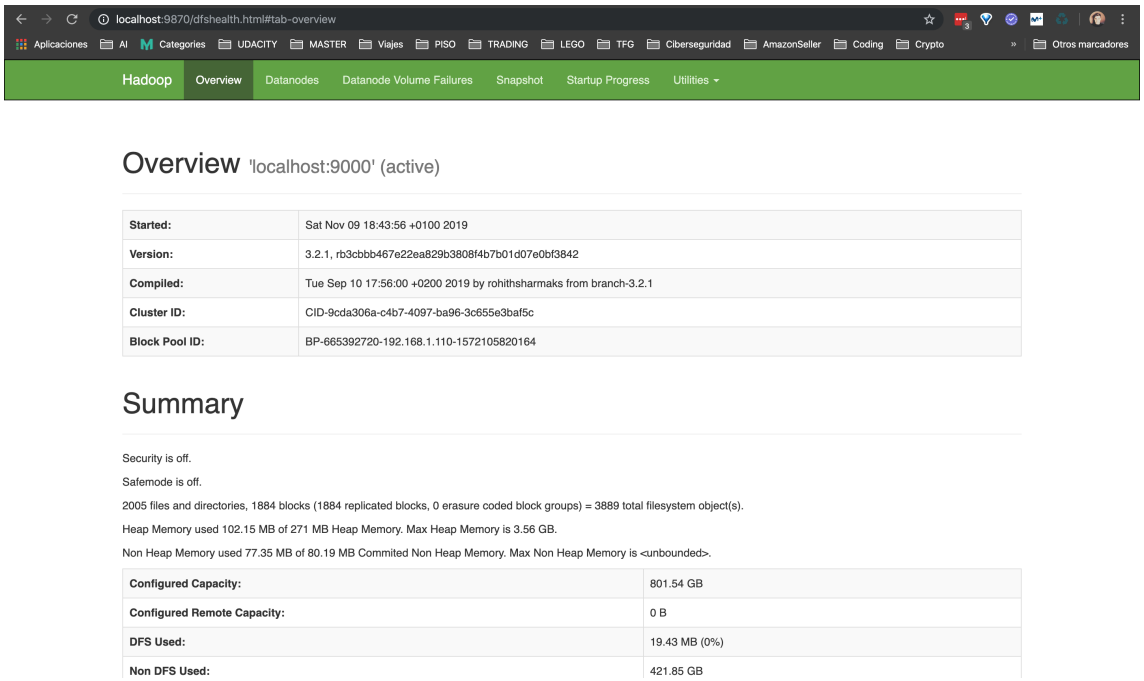


Figura 16: Interfaz web HDFS

3.4.5. Stack ELK

Docker *compose* que levanta estas tres tecnologías y la correspondiente configuración de Logstash para poder escuchar el *topic* ”testoutput”generado por el programa en Spark Streaming predictivo y explotar el *value* del mensaje en un json y así poder indexarlo en ElasticSearch, además de visualizar ese *index* en Kibana.

- Logstash

```
input {
  kafka {
        bootstrap_servers => "kafka:9092"
        topics => ["testoutput"]
  }
}

filter {
  json {
    source => "message"
  }
}

## Add your filters / logstash plugins configuration here

output {
  elasticsearch {
    hosts => "elasticsearch:9200"
    user => "elastic"
    password => "changeme"
    index => "data"
  }
}
```

Figura 17: Logstash Config

- ElasticSearch

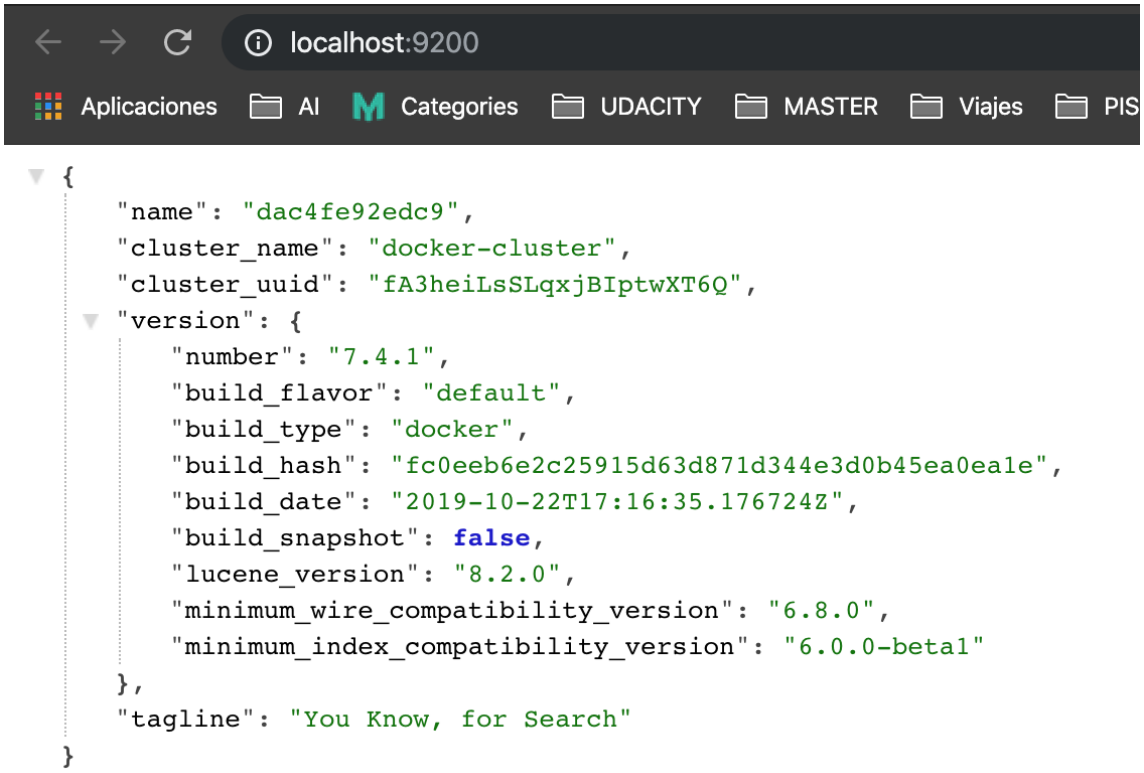


Figura 18: Elastic desplegado

- Kibana

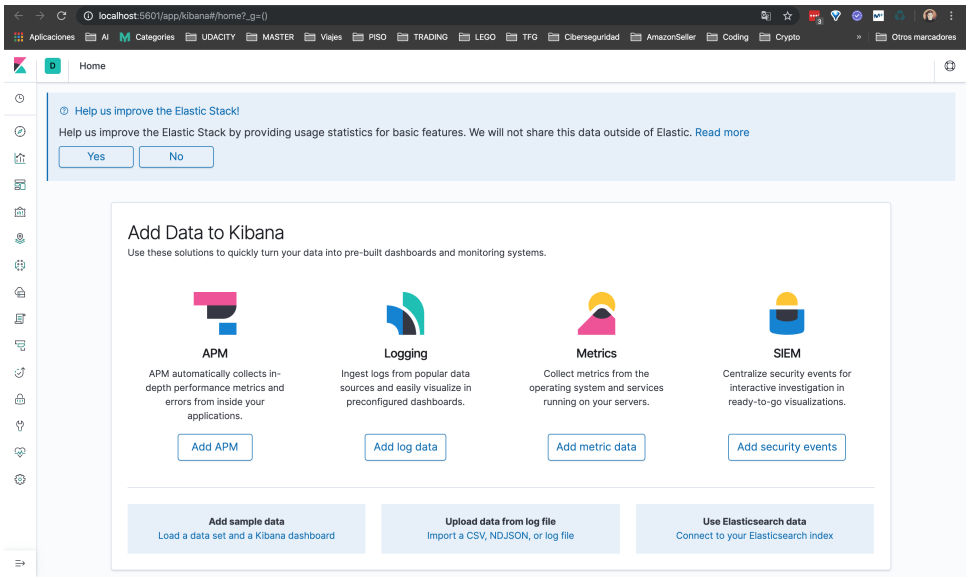


Figura 19: Kibana desplegado

4. Modelado

Esta sección no es el foco del proyecto, de ahí que no se obtengan los mejores resultados, pero permite su uso a nivel funcional para la demostración del caso de uso.

Las herramientas que se han usado para desarrollar esta sección han sido Jupyter Notebook con PySpark y la librería de ML de Spark MLlib.

El algoritmo escogido es la regresión logística, debido a la sencillez que supone usarlo, se podría hacer escogido un algoritmo de ML mejor o incluso una red neuronal recurrente (RNN) como la LSTM que van bien con las series temporales, pero no entraba dentro del scope del proyecto. Para el entrenamiento se ha escogido uno de los CSVs del *dataset* original ej. ïbm.us.txt". Tras la elección del *dataset* de entrenamiento, se han tenido que hacer una serie de transformaciones:

1. Con Python y la librería Pandas, se ha escogido la columna de precio de cierre y se ha ido troceando la columna de seis en seis elementos ordenados por tiempo para emular ventanas de 6 precios (n5, n4, n3, n2, n1 y n). Se ha creado una nueva columna comparando al instante n con n1: si n es mayor que n1, tiene valor 1.0 o True (significa que el precio de cierre va a subir) y 0.0 o False (significa que el precio de la acción va a bajar). Tras crear la columna "label", se borrará la columna "n", ya que no es necesaria.
2. Una vez transformado el *dataset*, se crea un Pipeline de *machine learning* que hará dos cosas:
    a) Agrupar las *features* con un vector en una sola columna con un VectorAssembler
    b) Hacer uso del nuevo vector de *features* y la columna *label* en la regresión logística para entrenarla.

Tras terminar el entrenamiento, el modelo creado se guarda en HDFS.

5. Conclusiones

Durante este proyecto, las mayores dificultades han estado sobre todo en la parte de la predicción de datos, ya que no era el foco principal de este trabajo y por motivos de tiempo y scope no se ha podido llegar a profundizar más en este aspecto.

Al sistema le faltan algunos módulos por dockerizar para así darle completa independencia. Pero con lo que se ha llegado a conseguir, se puede ver que es robusto y están bien planteados los módulos.

Una de las fortalezas del sistema propuesto es que a la hora de recibir datos reales en *streaming*, al estar tan desacoplados los módulos, no afectaría a este y los cambios para incorporar este nuevo flujo de datos serían mínimos.

Al ejecutar este sistema en el ordenador del alumno, se notan algunas carencias, pero esto se solventaría escalándolo de forma muy fácil por cómo está planteado, en un *cluster* de verdad, haciendo de la instalación de este muy fácil y sencilla.

El alumno ha mejorado bastante su conocimiento en base a estas tecnologías, y gracias a la incorporación de Docker ha sido capaz de desacoplar la primera versión que se tenía del proyecto en módulos bastante más sencillos y explicables.

6. Trabajos futuros

Tras la finalización del trabajo, el alumno ha querido plantear una lista con los objetivos que cumplían con el deseo inicial del alumno, pero que no han podido ser abordados debido al tiempo y alcance del proyecto disponibles:

- Dockerización completa del sistema
- Instalación en una plataforma de orquestación de contenedores (DC/OS, Kubernetes) en un *cluster* real
- Mejora del modelo de *machine learning* para poder hacer mejores predicciones
- Ofrecer varios tipos de visualizaciones que den más información para este caso de uso.

7. Código

Github Link To Repo