Universidad ORT Uruguay

Facultad de Ingeniería

Sistema de tarifado dinámico en near-to-real-time para Smart Parkings

Entregado como requisito para la obtención del título de Maestría en Big Data

Pedro J. Bonillo - 265075

José Díaz - 230253

Juan D. Mattos - 262316

Tutor de Tesis:

Alexis Quintana

2022

Declaración de autoría

Quien suscribe, Pedro J. Bonillo, José Díaz y Juan D. Mattos declaramos que el presente trabajo es de nuestra autoría.

Podemos asegurar que:

- El trabajo fue producido en su totalidad mientras realizaba el Máster en Big Data de la Universidad ORT Uruguay.

- En aquellas secciones de este trabajo que se presentaron previamente para otra actividad o calificación de la universidad u otra institución, se han realizado las aclaraciones correspondientes.

- Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad.

- Cuando citamos obras de otros, hemos indicado las fuentes. Con excepción de las citas, la obra es enteramente nuestra.

- En el trabajo, hemos acusado recibo de las ayudas recibidas.

- Ninguna parte de este trabajo ha sido publicada previamente a su entrega.

José Díaz

Juan D. Mattos

Pedro J. Bonillo

Dedicatoria y agradecimientos

Dedicamos la realización de esta Tesis a nuestras familias y agradecemos el esfuerzo y apoyo incondicional brindado durante toda nuestra etapa académica. Seguramente, sin ellos, nada de lo conseguido podría haber sido posible.

Resumen

Cuando se habla de una ciudad inteligente (Smart City), nos enfocamos en el cómo una ciudad administra y trata sus asuntos que tienen que ver con el agua, con la energía, con la movilidad, con sus edificios-casas y con la basura. La tesis se centra en el factor movilidad y más precisamente en sus estacionamientos.

El parque automotor de las zonas urbanas del mundo y del cual Uruguay no queda al margen, ha aumentado exponencialmente en los últimos años. Este proyecto nace con el objetivo de optimizar las búsquedas de lugares para estacionar y poder reducir todas estas problemáticas que este proceso conlleva.

La tesis presentada tendrá como meta principal diseñar e implementar una arquitectura con elementos de IoT, Big Data y Machine Learning, capaz de procesar un gran volumen de datos estructurados relacionados a parkings inteligentes y poder ofrecer una solución real que ahorre los tiempos del proceso de estacionar un vehículo al mismo tiempo que sea una solución tentadora para la Gerencia del parking que pueda incrementar sus ingresos mediante tarifas dinámicas.

El procesamiento será en near-to-real-time y tendrá una interfaz de usuario capaz de desplegar distintas formas de visualizar información de interés, ya sea capacidad total y disponibilidad de los parkings, características de los distintos estacionamientos y tarifado dinámico.

Los atributos de calidad manejados por la arquitectura implementada serán la escalabilidad, tolerancia a fallos, performance y disponibilidad.

Palabras clave: Big Data, IoT, Parking Systems, Open Data

Abstract

Whenever we're discussing a Smart City, the factors that come into play are usually how the city manages all its issues related to water supply, energy production, mobility, construction and garbage disposal.

This thesis's main focus is mobility, and inside this category, parking lots. The fleet of vehicles in urban areas of the world (from which Uruguay is not excluded) has increased exponentially in recent years. This project was born with the main objective of optimizing parking lot searches and to be able to reduce all the problems that this process entails.

In order to achieve this, we will design and implement a big data architecture with elements of IoT (internet of things) and Machine Learning, capable of processing a really large volume of structured data related to parkings.

The architecture will be able to offer a real solution to vehicle parking issues, in order to improve urban transportation and make cities more livable, efficient and also environmentally friendly. It will also be a tempting solution for all parking owners that implement the system, since they will be able to increase their income through dynamic rates and a more efficient parking space management.

Data processing will be in near-to-real-time. The architecture will consist of the main gathering and processing data system and a user-friendly interface, capable of displaying relevant information to vehicle and parking owners, such as total capacity, available spots and dynamic pricing.

The main quality attributes managed by the implemented architecture will be scalability, fault tolerance, performance and availability.

Keywords: Big Data, IoT, Parking Systems, Open Data.

Contenido

Índice general

Índice de figuras

Capítulo 1

INTRODUCCIÓN

1.1 Problema

Uruguay y sobre todo su capital Montevideo presenta problemas de congestionamiento vehicular como consecuencia del crecimiento demográfico, así como de su centralización poblacional en la capital. La dificultad a la hora de buscar estacionamiento trajo consigo una gran cantidad de problemas como el incremento del estrés al volante, aumento de congestiones en las calles y mayor polución emitida al ambiente, entre otros.

Las dirigencias políticas ya empezaron a considerar la problemática. En 2016 el Poder Ejecutivo otorgó a través del Decreto 110/16 beneficios fiscales a las actividades de construcción, ampliación y explotación de estacionamientos en las zonas decretadas de carácter prioritario, para incentivar la inversión privada en dicho sector. Se consideran zonas prioritarias según dicho decreto las comprendidas dentro de los siguientes límites en el departamento de Montevideo: calle Hipólito Yrigoyen, Avenida Italia, Avenida Dr. Luis Alberto de Herrera, Avenida General Flores, Bulevar General Artigas, Rambla Baltasar Brum, Rambla Edison, Rambla Sud América, Rambla Franklin D. Roosevelt, Rambla 25 de Agosto de 1825, Rambla Ingeniero Monteverde, Rambla Francia, Rambla Gran Bretaña, Rambla República Helénica, Rambla República Argentina, Rambla Presidente Wilson, Rambla Mahatma Gandhi, Rambla República del Perú, Rambla Armenia, Rambla República de Chile, en todos los casos de ambas aceras.

La ciudad de Montevideo cuenta a su vez con una zona de estacionamiento tarifado gestionada por la Intendencia de Montevideo, con el apoyo logístico para la venta de tiempo de la red de cobranza descentralizada y de las empresas de telefonía celular. Los sistemas de estacionamiento tarifado surgieron por la necesidad de

generar espacios para estacionar en sectores de la ciudad donde hay importante demanda.

A pesar de los esfuerzos en resolver la problemática, lo cierto es que la misma se encuentra lejos de resolverse. El alcance del proyecto se limitará a brindar soluciones en los estacionamientos privados y no situados en la vía pública.

1.2 Objetivos

1.2.1 Objetivo general

Diseñar una Arquitectura de Big Data (Smart Parking System) para la implementación de tarifas dinámicas en near to real time para estacionamientos inteligentes.

1.2.2 Objetivos específicos

i. Integrar arquitecturas y tecnologías vista en el Master de Big Data, para proponer un Sistema de Parkings Inteligentes.
ii. Procesar una gran cantidad de datos que se generan a gran velocidad previendo la variabilidad de datos.
iii. Aplicar una arquitectura de streaming para el análisis de los datos que permita implementar el tarifado dinámico.
iv. Realizar un modelo orientado a Machine Learning para predecir el estado de ocupación de un área del parking
v. Plantear las bases a través del modelo de Machine Learning que permita las recomendaciones y reservas a los usuarios del parking.
vi. Proponer una solución alternativa a la existente en el mercado uruguayo que mejore problemas asociados a la alta demanda de estacionamientos en la actualidad.
vii. Diseñar una solución visual e intuitiva (prueba de concepto) para que el usuario interactúe con el sistema.

1.3 Justificación

Nuestra Tesis se centra en resolver la problemática presente en nuestro país a la hora de encontrar los slots libres en estacionamientos, sobre todo en las capitales y ciudades de mayor densidad demográfica en donde el crecimiento ha generado que los parkings se saturen con mayor frecuencia y no sea posible aparcar en los lugares disponibles ya sea por no detectarlos o por los atascos en la circulación de los automóviles dentro del parking.

El proyecto tiene como innovación en nuestro país, permitir al usuario poder decidir ni bien ingresa a un estacionamiento inteligente, a qué sector del estacionamiento se va a dirigir. Esto se logra a través de una pantalla (WebApp) que muestra el estacionamiento fraccionado en sectores, con su disponibilidad de lugares y su precio por hora dinámico. A su vez, va a permitir mediante cualquier dispositivo móvil, ver en tiempo real esta información y poder generar en un futuro reservas.

Usuario:

El usuario va a poder optar por aquellos slots con un valor hora más bajo, por su porcentaje de ocupación, pero que posiblemente se encuentre alejado a la entrada o en pisos superiores o bien, optar por abonar un precio mayor en sectores más demandados y de baja disponibilidad.

El usuario también se ve beneficiado con la opción de ver en cualquier momento el nivel de ocupación de un estacionamiento desde su dispositivo móvil y evitar concurrir al mismo si este se encuentra a tope, generando mayores demoras, estrés y conflictos en la circulación de los vehículos.

A todo esto, se le agrega la función de poder reservar slots que actualmente no está contemplada en el mercado de los estacionamientos uruguayos.

Gerencia:

La gerencia del estacionamiento se va a ver beneficiada, en primer lugar, por una mayor satisfacción de los usuarios gracias a los beneficios de nuestra solución, lo que se va a ver reflejado en mayores ventas y por ende mayores ingresos.

Pero en mayor medida, la gerencia va a contar en nuestra solución como un aliado a lo hora de controlar los precios. Pongamos un ejemplo sencillo de cómo funciona una estrategia de aumento en los precios de los slots, sin impactar negativamente en los usuarios finales. Si se quiere aumentar los ingresos mediante un aumento de precios, una buena estrategia sería aumentar el costo de aquellos slots más demandados (en color rojo en la pantalla y con baja disponibilidad) y reducir levemente el costo de los menos demandados (en color verde y de alta disponibilidad). El usuario siempre va a tener la opción de elegir un slot más barato, aunque ello conlleve trasladarse a un piso superior.

La solución con el tarifado dinámico posibilita un flujo armónico de los vehículos a todos los sectores del estacionamiento evitando que esté la mayoría de los vehículos “dando vueltas” en la entrada o en el piso principal.

1.4 Metodología

En este proyecto se implementará un tablero en Trello el cual nos permite el uso de una metodología del tipo Kanban, es muy usada en ambientes de coordinación de equipos para el desarrollo de software, ingeniería y productos, que mediante la incorporación de diferentes tableros y tarjetas (tareas a realizar) nos brinda una forma sencilla de visualizar el trabajo para poder coordinar quién hace qué, las diferentes etapas este proyecto y cuáles son los vencimientos de cada etapa. En la actualidad, los tableros Kanban se representan en su mayoría como tableros virtuales con columnas que representan cada etapa del trabajo.

A la metodología Kanban, se le agregan conceptos de scrum y desarrollo ágil, con el trabajo dividido en sprints de una/dos semanas y teniendo coordinaciones semanales

entre el equipo para coordinar entre nosotros qué se hizo, qué se va a hacer y si hay algún punto de preocupación a tener en cuenta. Si bien Kanban hace focus en la mejora de los procesos, Scrum se centra en ayudar a los equipos a culminar más trabajos y con mayor agilidad.

Adicionalmente, se hará uso de la herramienta de GitHub, que nos permite mantener un orden a la hora de realizar diferentes desarrollos a lo largo de la arquitectura, su uso se deriva a las funciones colaborativas que ayudan al desarrollo simultáneo entre los integrantes del proyecto.

Por último, a través de la plataforma de Dropbox, se gestionará la documentación de la Tesis y se elaborará el informe.

1.5 Alcance y limitaciones

En este trabajo de posgrado se plantea la implementación de una arquitectura de Big Data que sea capaz de poder procesar de forma distribuida un gran volumen de datos (+1 Millón de registros) tanto a nivel de streaming como en batch, de tal manera de realizar un modelo basado en las 5 capas que representa un proyecto de manejo de grandes volúmenes de datos. Su función principal se centra en poder ejecutar y procesar datos con la finalidad de poder realizar en tiempo real un sistema que permita ajustar precios de forma dinámica a los puestos de los parkings de la ciudad de Montevideo.

Cabe aclarar que en la presente Tesis, se considerarán los términos “tiempo real” y “near-to-real-time” como equivalentes, aunque en todos los casos se refieren a un proceso cercano al tiempo real.

Para el desarrollo de este proyecto existen una diversidad de retos técnicos a abordar, los cuales se presentan en diferentes secciones de procesamiento de datos de la plataforma. Por otro lado, los datos que se pasan en la arquitectura son simulados ya que no se cuenta con la oportunidad de poder obtener datos reales de Parkings de la ciudad de Montevideo. Del mismo modo, se busca construir una arquitectura que

cumpla con las bases fundamentales para un proyecto de Big Data cuyo foco se centra en el soporte de las denominadas 5 V's, que son: Velocidad, Volumen, Veracidad, Variabilidad y Valor.

Dado que el objetivo principal es establecer una arquitectura de Big Data, este trabajo no estará centrado en el área de aplicar modelos predictivos basados en Machine Learning. Sin embargo, se aplicará un modelo simple para demostrar su integración dentro de la arquitectura. Esto sentará las bases para implementar futuras funcionalidades, por ejemplo, reservas y recomendaciones en tiempo real.

Adicionalmente, un punto importante a destacar es que el hardware a usar es de bajo recursos ya que no se cuenta con la posibilidad de adquirir un servidor que genere el valor necesario para la investigación y finalmente, otro criterio limitante para la investigación es que se usaron en su mayoría sistemas open-source lo cual genera que al momento de hacer integraciones se tome más tiempo de desarrollo.

Capítulo 2

MARCO TEÓRICO

2.1 Internet de las Cosas (IoT)

Continuamente avanzamos hacia una manera de entender el mundo de una manera absolutamente conectada a la red. Las cosas de nuestra vida cotidiana están conectadas y se comunican entre sí. La innovación en Internet de las Cosas tiene el objetivo de conectar esas cosas cotidianas al internet, aproximando cada vez el mundo tangible al digital. El Internet de las Cosas cuenta con las siguientes características:

Conectividad: Las cosas se conectan a la red para interrelacionarse con los usuarios y otros dispositivos o sistemas mediante WI-FI como tipo de acceso, por ejemplo.

Sensibilidad: A través de sensores se puede detectar movimiento, temperaturas, comportamientos y otros patrones o parámetros.

Comunicación: Las personas, los dispositivos y el mundo físico permanentemente interaccionan entre ellos, por lo que es importante conocer la relación existente entre los mismos.

Seguridad: Los dispositivos deben contar con medios de seguridad para asegurar que se proteja la integridad y privacidad de los datos que se van transmitiendo.

2.2 Arquitecturas de Big Data

El procesamiento de los datos se puede realizar de dos maneras, la primera en modo batch, y la segunda en modo stream o near-to-real-time.

Se puede diferenciar ambos tipos de procesamiento de datos principalmente en los tiempos de procesamiento, siendo el procesamiento batch aquel que nos permite procesar volúmenes de datos por ejemplo en horas o días, mientras que el modo

stream nos posibilita procesar los datos casi al instante en que estos son generados, como por ejemplo cada segundo o milisegundos.

Es aquí en donde nace la Arquitectura Lambda como una forma de implementar sistemas de procesamiento de datos que combinan ambas modalidades de procesamiento de datos: batch y stream.

La Arquitectura Lambda permite equilibrar la latencia, el rendimiento y la tolerancia a fallos a partir del procesamiento por lotes, mientras que al mismo tiempo utiliza el procesamiento de datos en tiempo real para proporcionar vistas dinámicas.

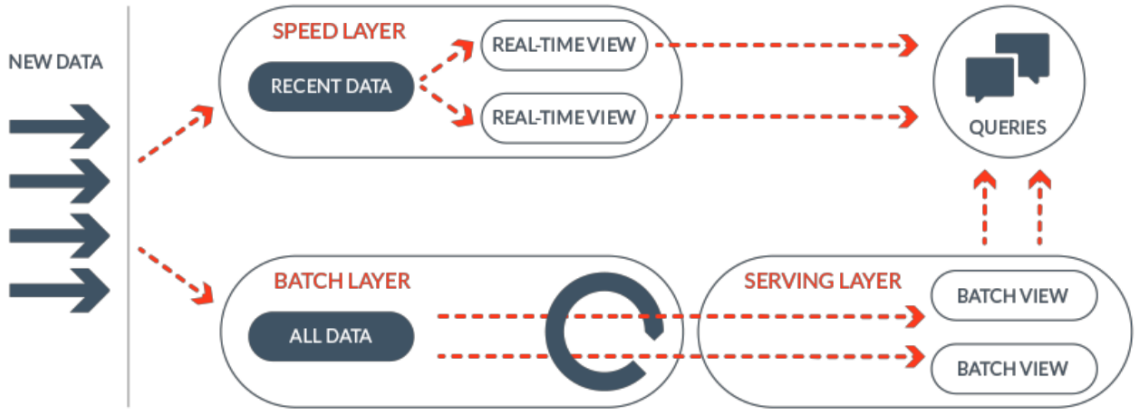


Figura 1.- Arquitectura Lambda

Por otra parte, nos encontramos con la Arquitectura Kappa, que tiene por idea fundamental eliminar el concepto de sistema manejadores de base de datos. Se tiene una tecnología de computación distribuida que permite procesar un flujo constante de datos en tiempo real. Todo debe estar orientado a ser procesado en real time y en streaming y debemos olvidarnos de la capa batch. La ingesta de datos se realiza en su totalidad a través de streaming.

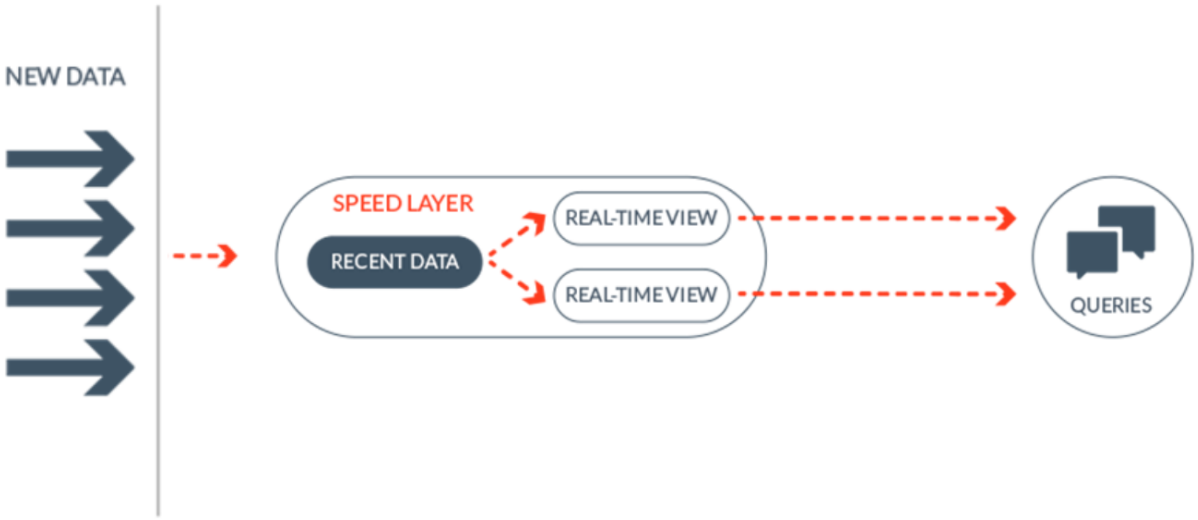


Figura 2.- Arquitectura Kappa

Para nuestro caso de estudio, la data seleccionada corresponde a eventos del pasado (aunque hayan sido generados) para poder predecir el porcentaje de ocupación de cada área de un estacionamiento y de esta forma predecir el precio futuro y generar una reserva, por lo que nos decidimos por la Arquitectura Lambda. Esta arquitectura nos otorga lo mejor de dos mundos, ya que el modo batch nos brinda un alcance completo y confiable mientras que el modo stream nos da los datos en línea para decisiones instantáneas.

2.3 Marco de referencia

Dentro de este proyecto de grado, su principal fundamento se centra en la aplicación de un modelo de Big Data que cumpla con las 5 diferentes “V's” que habilitan a diseñar una solución para el manejo de grandes volúmenes de datos. Estas últimas se describen de la siguiente manera:

- Volumen**:** Para las consideraciones de un proyecto de Big Data, esta característica es la más importante ya que como su nombre lo indica, se piden a

los arquitectos de datos que el sistema a implementar pueda contar con la posibilidad de procesar más de 1 millón de registros.

- Variabilidad**:** Este punto hace referencia a que al tener presencia de datos del tipo no estructurados y/o semiestructurados, se necesita una arquitectura de diferentes frameworks que puedan ser capaces de soportar datos variables dentro de la solución.
- Veracidad**:** Respecto a este punto, su foco se centra en el planteamiento de la calidad de los datos, saber qué tan veraz es un lote de datos y denominar si los mismos realizan un aporte al desarrollo de la arquitectura para posteriores análisis.
- Velocidad**:** Como su nombre lo indica, es necesaria la integración de datos en distintos períodos de tiempo, haciendo uso de diferentes técnicas que permitan el procesamiento de la data relevante en real time, near-to-real-time o incluso en batch, denominando en estos tres diferentes tipos de rangos de tiempo en segundos, minutos o incluso horas de procesamiento.
- Valor**:** Por último, todo proyecto relacionado a datos hace necesaria la existencia de presentar algún valor interesante al negocio y al cliente final que consuma los recursos de la plataforma.

El marco de referencia elegido es el modelo general de una arquitectura de software para hacer streaming en tiempo real de Big Data (es un modelo de referencia y por lo tanto está envuelto en una arquitectura).

Esta se caracteriza por describir su funcionamiento en cinco capas diferentes en su procesamiento y en su descripción a nivel de software, formando una continuidad y un flujo, el cual permite realizar correctamente el tratado de los datos en diferentes procesos de una solución.

La primera capa es dónde se realiza la ingesta de los datos (Collection tier) y en dónde se pueden utilizar productos de software como por ejemplo Amazon Kinesis, Flink y Mosquitto. Estas herramientas entre otras, permiten que se ejecute la colección de los

datos en tiempo real para luego pasarlos a un componente de cola de mensajes en lo que sería la segunda capa (Message queuing tier).

La tercera capa es la de análisis de los datos (Analysis tier), con el fin de hacer sumarizaciones, agregaciones o computar algo. Por lo general se va a querer computar algo y realizar la comparación con el pasado (computar la historia y lo inmediato). Esta capa de análisis de datos tiene dos formas de trabajar; estructurado y no estructurado, con persistencia y sin persistencia. Cuando en la capa de análisis se realiza un streaming en tiempo real no estructurado, significa que se admite la pérdida de datos, y se encontraría en lo que se denomina streaming suave o near to real time. Por otro lado se tiene en la capa de análisis, las arquitecturas de analítica en tiempo real duras (estructurado) que no admiten fallas. Estas últimas, como no admiten fallas, tienen que tener un control desde el inicio hasta el fin de la transacción y a su vez se tiene que poder recuperar la transacción en cualquiera de sus etapas, por lo que se va a requerir un mecanismo de persistencia (se guarda un control de lo que está sucediendo en un área especial de memoria).

Una vez que se procesaron los datos en la capa de análisis, estos se envían a un sitio específico de memoria (esta sería la capa de almacenamiento) en donde se puede acceder rápidamente a esa data para presentarla en un teléfono o en un dashboard (capa de acceso). Los valores de lo que está ocurriendo son de uso para el equipo de data scientists que necesitan que éstos sean almacenados en las variables que alimentan el modelo (features store).

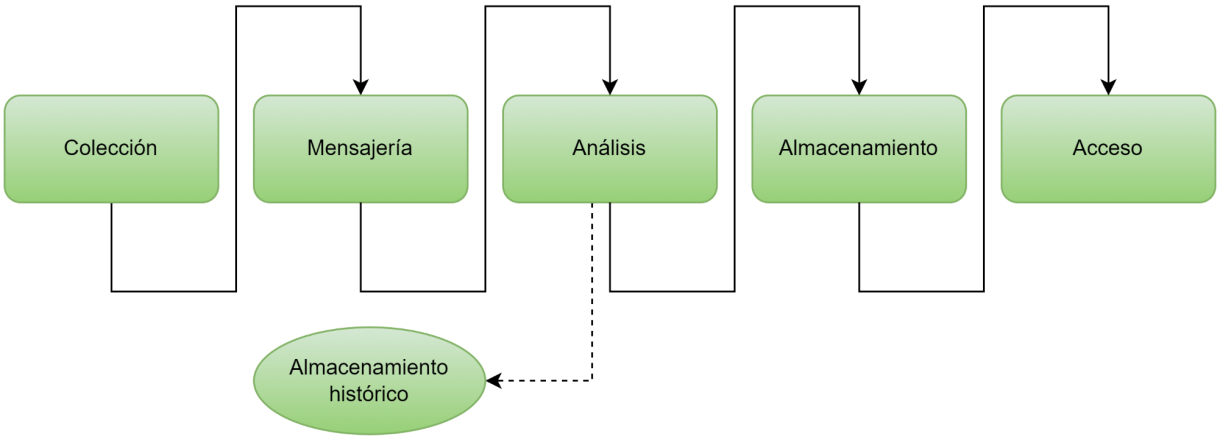


Figura 3.- Marco de referencia

2.4 Principales tecnologías a utilizar

2.4.1 Hadoop

El proyecto Apache™ Hadoop® desarrolla software de código abierto para una computación confiable, escalable y distribuida.

La biblioteca de software Apache Hadoop es un marco que permite el procesamiento distribuido de grandes conjuntos de datos a través de clústeres de computadoras utilizando modelos de programación simples. Está diseñado para escalar desde servidores individuales a miles de máquinas, cada una de las cuales ofrece computación y almacenamiento locales.

En lugar de depender del hardware para ofrecer alta disponibilidad, la biblioteca en sí está diseñada para detectar y manejar fallas en la capa de aplicación, por lo que ofrece un servicio de alta disponibilidad sobre un clúster de computadoras, cada una de las cuales puede ser propensa a fallas.

2.4.2 MapReduce

MapReduce es una herramienta, con un modelo concreto de programación, usado en Big Data ya que permite una escalabilidad masiva en cientos o miles de servidores en

un clúster de Hadoop. En sus inicios, MapReduce era un sistema usado por Google para analizar los resultados de búsqueda, pero con el correr del tiempo, MapReduce, componente de procesamiento, se transformó en el corazón de Apache Hadoop . El término "MapReduce" se refiere a 2 procesos separados y distintos que realizan los programas Hadoop. Uno es la función Map, que toma los datos de entrada y los divide en bloques más pequeños. Después, a cada bloque se le asigna un mapper (un servidor de Hadoop que ejecuta las funciones de MapReduce) para poder procesarlo.

La función Reduce toma la salida de la tarea de Map como entrada y combina esas tuplas de datos en un conjunto más pequeño de tuplas. Reduce procesa los datos de tal forma que se simplifican y se leen de manera secuencial, con lo cual ejecuta un archivo de salida por cada una de las tareas procesadas. Como implica la secuencia del nombre MapReduce, el trabajo de reducción siempre se realiza después del trabajo del algoritmo de mapeo.

Beneficios:

Escalabilidad**:**. Permite procesar petabytes de datos almacenados en el sistema de archivos distribuido de Hadoop.

Flexibilidad: Acceso más fácil a múltiples fuentes y tipos de datos.

Velocidad: Rápido procesamiento de cantidades masivas de datos.

Simple: Permite a los desarrolladores escribir código en una gran variedad de lenguajes, tales como, Java, C ++ y Python.

2.4.3 Apache HIVE

Hive es un software que trabaja sobre clústeres de Hadoop creando una capa que permite al desarrollador gestionar grandes de ficheros HDFS y MapReduce y realizar consultas de datos basadas en SQL, con el lenguaje HiveQL.

Con Hive se pueden realizar consultas que no sean demasiadas complejas y al proporcionar un lenguaje similar al SQL de las bases de datos relacionales para

trabajar con grandes cantidades de datos este software es muy adecuado para entornos de data warehouse y analítica.

Hive fue inicialmente desarrollado por Facebook para luego evolucionar como proyecto open source de Apache, dentro del ecosistema de Hadoop. Actualmente lo utilizan grandes compañías como Netflix o Amazon en Amazon Elastic MapReduce o AWS.

Una ventaja de Hive es que es similar a trabajar con bases de datos tradicionales. Con Hive debemos estructurar los datos agrupándolos en tablas, con sus columnas y tipos de datos asociados. Seleccionar los datos de estas tablas, procesarlos y finalmente analizarlos, sin olvidar que Hive no es un motor de bases de datos.

Una desventaja que presenta Hive es que, dado que requiere procesar la consulta y traducirla a lenguaje Java para crear el MapReduce, la latencia de respuesta es alta. Otra desventaja es que no soporta todo el lenguaje SQL, transacciones ni índices.

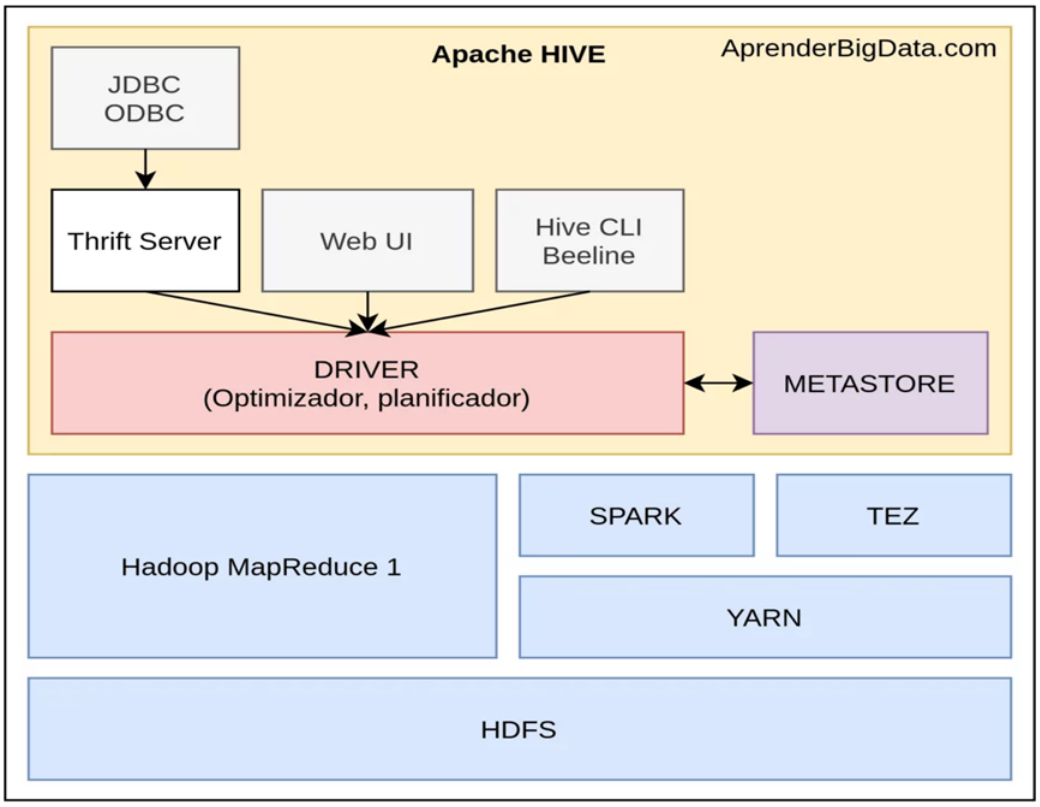


Figura 4.- Modelo de arquitectura de Apache Hive.

2.4.4 HUE

Hue (Hadoop User Experience) es un sistema de interfaz de usuario Apache Hadoop siendo este open source. Evolucionó a partir de Cloudera Desktop, y finalmente Cloudera lo aportó a la comunidad Hadoop de la Fundación Apache. Se implementa en base al framework Python Web Django. Hue es un marco de integración de interfaz amigable que puede integrar una gran cantidad de marcos de software de sistemas de Big Data, y todos los marcos se pueden ver y ejecutar a través de una interfaz.

Hue permite interactuar con el clúster de Hadoop en la consola web para analizar y procesar datos, como datos operativos en HDFS, ejecución de MapReduce Job, ejecución de sentencias Hive SQL, exploración de la base de datos HBase, etc.

Características:

- Acceso a la API de Hadoop
- Presencia del navegador de archivos HDFS
- Navegador y diseñador de trabajos
- Interfaz de administración de usuario
- Editor de consultas de Hive
- Editor de consultas de Pig
- Acceso a Hadoop Shell
- Los flujos de trabajo pueden acceder a la interfaz de Oozie
- Las búsquedas SOLR pueden obtener una interfaz separada

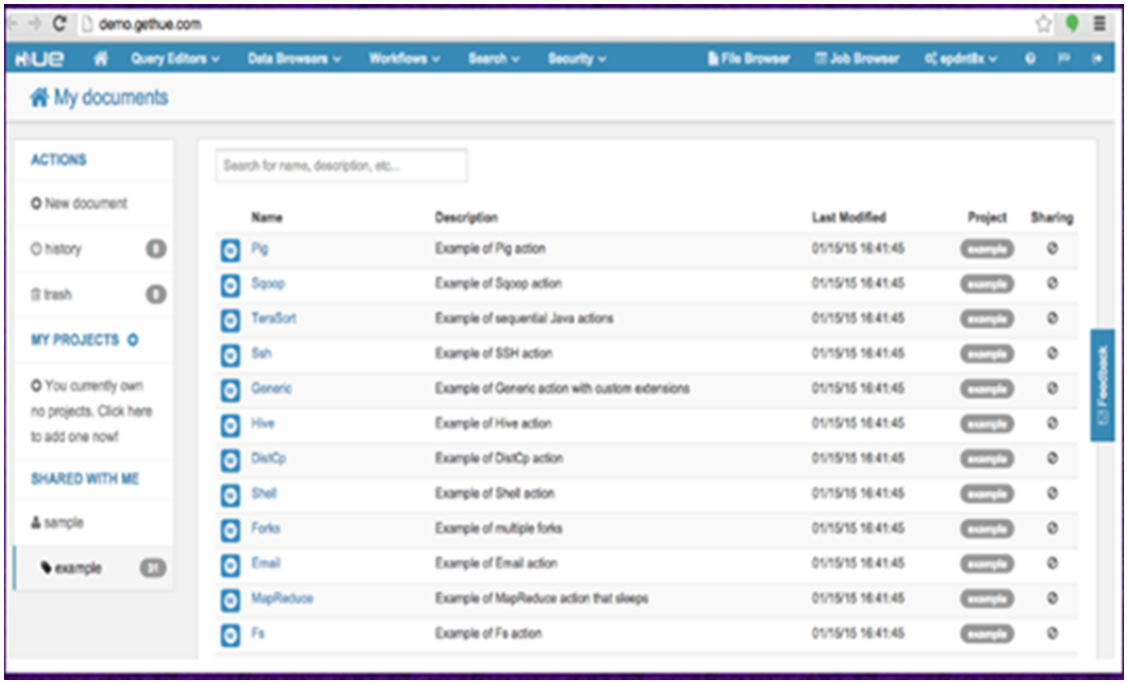


Figura 5.- Hue

2.4.5 Python

Python es un lenguaje de programación interpretado de tipado dinámico cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma y disponible en varias plataformas, con soporte en programación orientada a objetos, imperativa y funcional. Fue utilizado para desarrollar aplicaciones de todo tipo, ejemplos: Instagram, Netflix, Spotify entre otros.

Es Administrado por Python Software Foundation, posee una licencia de código abierto, denominada Python Software Foundation License. Presenta constantes actualizaciones siendo a la fecha de presentación de la Tesis la versión Python 3.11.0b5, su última actualización.

Python se clasifica permanentemente como uno de los lenguajes de programación más populares en la actualidad.

2.4.6 Jupyter Notebook

Jupyter Notebook es una aplicación web de código abierto cliente-servidor lanzada en 2015 por Proyecto Jupyter. Cada desarrollador puede dividir el código en partes y trabajar en ellas sin importar el orden: escribir, probar funciones, cargar un archivo en la memoria y procesar el contenido.

Permite crear y compartir documentos web en formato JSON que siguen un esquema versionado y una lista ordenada de celdas de entrada y de salida. Estas celdas albergan, entre otras cosas, código, texto (en formato Markdown), fórmulas matemáticas y ecuaciones, o también contenido multimedia (Rich Media).

Jupyter Notebook es un entorno de desarrollo interactivo con el live code. Jupyter muestra una ejecución del código a través del navegador web. Si un desarrollador quiere visualizar un gráfico o una fórmula, escribe el comando deseado en la celda correspondiente. Este enfoque ahorra tiempo y ayuda a evitar errores. Los documentos creados en Jupyter pueden exportarse, entre otros formatos, a HTML, PDF, Markdown o Python.

Características:

- Admite lenguajes de programación de ciencia de datos populares como Python, R, Julia y Scala.
- Permite compartir cuadernos con otros colaboradores
- Integración de Big Data con Spark

2.4.7 Apache Kafka

Kafka se distribuye bajo la licencia Open Source de la Apache Software Foundation y con el tiempo se ha convertido en una de las tecnologías principales para el procesamiento de datos en tiempo real.

Se puede conceptualizar como un sistema de mensajes que permite el intercambio de datos entre procesos, aplicaciones y servidores, el cual utiliza el modelo de publicación – suscripción.

Componentes:

1. Productor: escriben los mensajes de Kafka.
2. Consumidores o Suscriptores: leen estos mensajes y los procesan.
3. Brokers: son los nodos que integran el clúster de Kafka y almacenan y distribuyen los datos

Ventajas:

1. Altamente escalable.
2. Actúa como un sistema de mensaje
3. Posee gran tolerancia a fallos.
4. Cada registro almacenado tiene un identificador.
5. El almacenamiento de mensajes tiene un período de retención.

Por estas razones en la actualidad, Apache Kafka es muy utilizado por las empresas en arquitecturas de Big Data, dado que consigue obtener una baja latencia, escalabilidad horizontal y absorber los picos de carga que pueden ocurrir en el sistema.

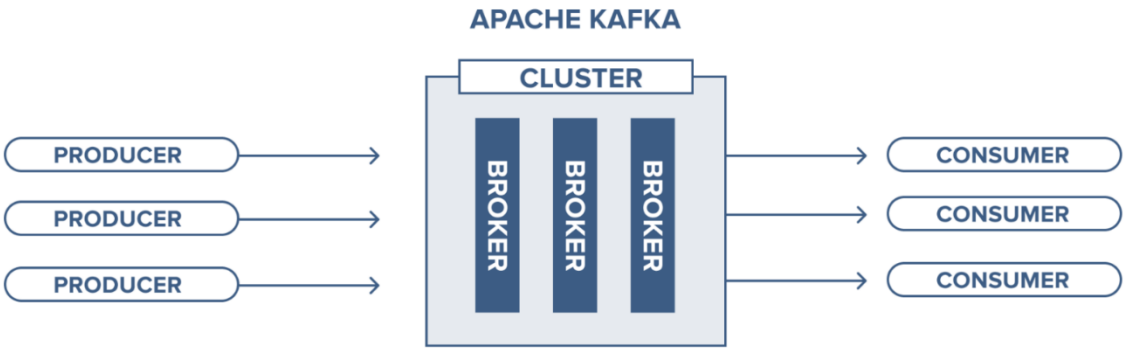


Figura 6.- Modelo de publish-subscriber

2.4.8 Patrón publicador - suscriptor

Para entender cómo funciona Apache Kafka debemos entender primero el concepto de patrón publicador – suscriptor.

El patrón publicador – suscriptor se utiliza para comunicar aplicaciones a través de mensajes. Es un sistema de eventos distribuidos en el que un suscriptor tiene interés por ciertos eventos. Por otra parte, el publicador es el que genera los eventos, que posteriormente serán enviados a los suscriptores interesados.

El suscriptor puede incorporar un filtro mediante tópicos para consumir solo los mensajes que le interesen.

2.4.9 Apache Zookeeper

Apache ZooKeeper es un proyecto de software libre de la Apache Software Foundation, que ofrece un servicio para la coordinación de procesos distribuido y altamente confiable que da soluciones a varios problemas de coordinación para grandes sistemas distribuidos. Se utiliza en sistemas distribuidos para la sincronización de servicios y como registro de nombres.

Cuando se trabaja con Apache Kafka, Zookeeper se utiliza principalmente para rastrear el estado de los nodos en el clúster de Kafka y mantener una lista de tópicos y mensajes de Kafka.

Características:

- Permite la coordinación entre procesos distribuidos mediante un namespace jerárquico que se organiza de manera similar a un file system. El namespace consiste en registros (znodes) similares a ficheros o directorios. En contraposición con el típico file system, ZooKeeper mantiene la información en memoria permitiendo obtener latencias bajas y rendimientos altos.
- Permite las réplicas instaladas en múltiples hosts, llamados conjuntos o agrupaciones. Los servidores que conforman el servicio ZooKeeper deben

conocerse todos entre ellos. Estos mantienen una imagen en memoria del estado, completado con un log de transacciones y snapshots almacenados en un store persistente. Mientras la mayoría de servicios estén disponibles, ZooKeeper se mantendrá disponible.

- ZooKeeper se encarga de marcar cada petición con un número que refleja su orden entre todas las transacciones de manera que permite mantener por completo la trazabilidad de las operaciones realizadas.

- Rápido sobre todo en entornos en los que predominen las operaciones de lectura.

2.4.10 Spark Streaming

Para desarrollar el trabajo se utilizó Spark, programado en Python, porque permite realizar cómputo distribuido en memoria lo cual brinda gran velocidad a las operaciones y tiene varios módulos de compatibilidad entre los cuales destaca SparkSQL y MLlib, este último módulo o biblioteca cuenta con varios algoritmos clásicos de Machine Learning para poder llevar a cabo un análisis predictivo en un conjunto de datos. MLlib tiene librerías para clasificación, regresión, agrupación en clústeres, recomendaciones, comprobación de hipótesis o cálculo de estadísticas entre otras.

Se puede definir Spark como un framework para el procesamiento de grandes volúmenes de datos de forma distribuida y a Apache Spark Streaming como una extensión de la API core de Spark, que da respuesta al procesamiento de datos en tiempo real de forma escalable, con alto rendimiento y tolerancia a fallos.

Spark Streaming proporciona diferentes medios para poder levantar la data en tiempo real ya sea mediante un dataframe (para datos estructurados), como también en un RDD (para datos no estructurados). Su finalidad es generar un procesamiento de datos distribuidos de forma rápida y segura que permita realizar diferentes operaciones en simultáneo dentro de un clúster mediante Spark. Es posible realizar la integración de datos mediante eventos ya sean en tiempo real (real time), cercanos a tiempo real

(near-to-real-time), micro-batch o incluso en procesamiento del tipo batch hablando ya en eventos de datos orientados a los minutos u horas de captura.

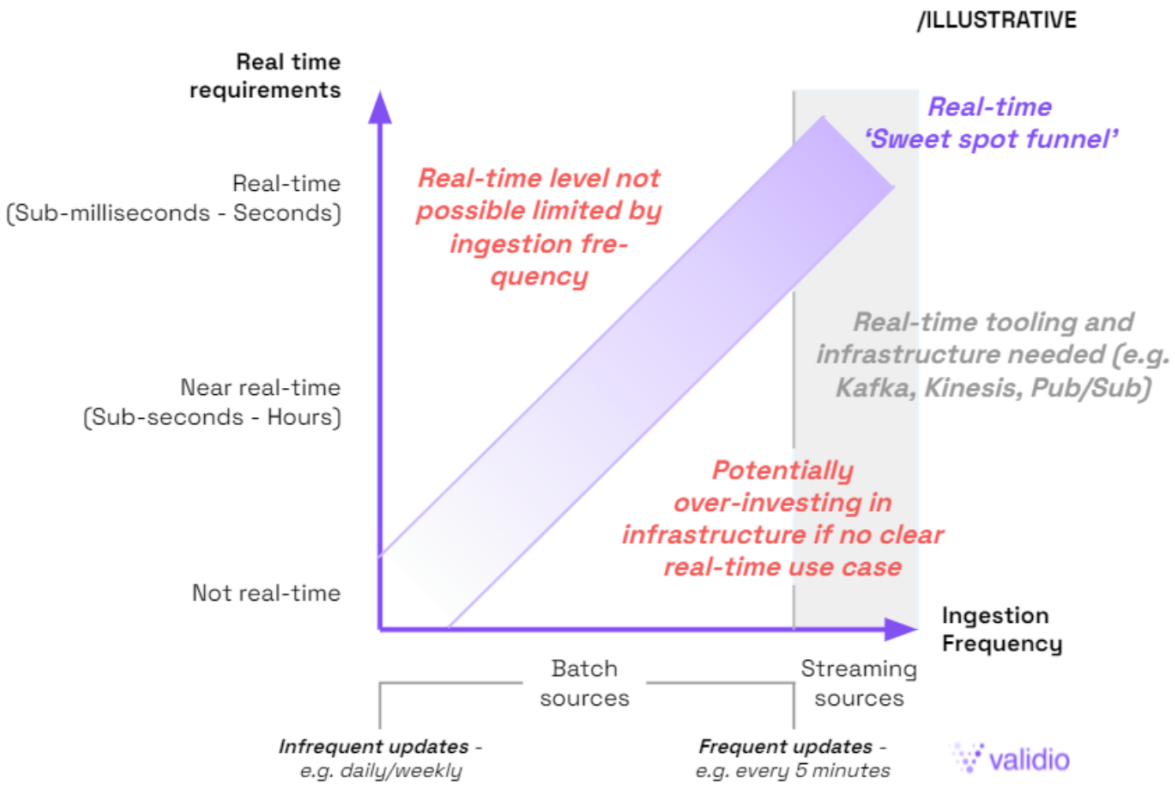


Figura 7.- Tabla de diferencias entre tipo de eventos.

Spark hace que los procesos de MapReduce nativos de Hadoop sean ejecutados de forma más rápida y eficiente, ya que su arquitectura cuenta con un contexto un poco diferente al realizado por Hadoop File System. Su clúster se basa en una orientación del tipo DAG (Directed Acyclic Diagrams), haciendo que los pasos a ejecutar en las tareas de mapeo sean por una parte compartidos entre sus workers, generalizando y optimizando sus tareas de reduce. La figura 8, muestra una Arquitectura de Apache Spark.

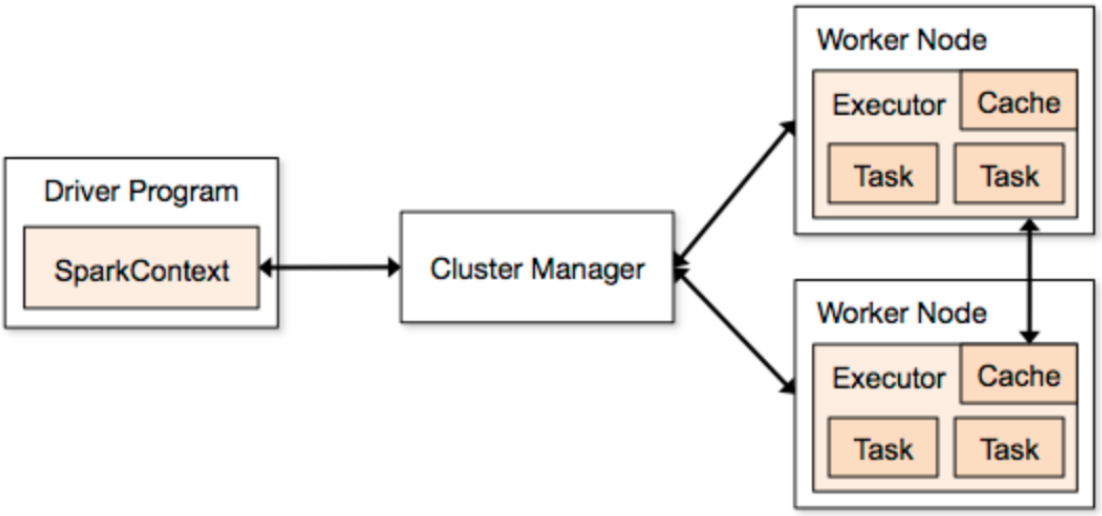


Figura 8.- Arquitectura de Apache Spark

Spark Streaming proporciona un marco que soporta completamente la Arquitectura Lambda, la cual nos permite trabajar simultáneamente el procesamiento de datos en batch y streaming. Spark Streaming puede ingerir datos de multitud de fuentes, tales como Kafka, Flume o Kinesis.

Ventajas

Dentro de las ventajas encontramos que Spark Streaming es una tecnología madura y de amplia aceptación, lo cual se ve reflejado por su gran comunidad de desarrolladores. Soporta diversos lenguajes de programación tales como; Java, Scala, Python, R y SQL, permitiendo utilizar librerías de Machine Learning y grafos. Es tolerante a fallas debido a su naturaleza micro batch.

Desventajas

Permite multitud de parámetros de configuración, por lo que incrementa su dificultad de uso, no es Streaming real por lo que no es ideal cuando se busca bajas latencias

menores a los 0,5 segundos y en muchas opciones avanzadas se queda por debajo de Flink.

2.4.11 Application Programming Interface

API o *Application Programming Interface* es un conjunto de funciones y procedimientos que permite integrar sistemas. Mediante esta interfaz, es posible reutilizar funcionalidad y aplicarla a otras aplicaciones o software. Su principal función es intercambiar datos entre diferentes sistemas, normalmente en formato JSON (JavaScript Object Notation) que es un formato ligero de intercambio de datos muy utilizado. Más que nada, es una forma estandarizada de formatear datos de texto en un formato legible por máquina y por humanos.

2.4.12 WebApp

WebApp es una aplicación normalmente implementada con lenguajes propios del desarrollo web, como HTML, CSS o JavaScript. La principal ventaja de las WebApp es que no hace falta desarrollar una app distinta para cada sistema operativo, sino que las mismas se ejecutan en un servidor web y se accede a ellas desde un navegador. Por esta razón, los usuarios finales pueden utilizarlas sin necesidad de instalar nada en sus dispositivos. Todo esto claramente se da siempre y cuando los mismos tengan acceso a internet.

Normalmente las WebApp tienen tres partes principales:

- Base de datos: Es donde se va a almacenar la información ya procesada proveniente del sistema de Big Data.
- Backend: Donde se desarrolla la API que consumirá tanto el sistema de Big Data (para enviar información) como la interfaz de usuario (para recibir información). Es la parte de la WebApp con acceso a las bases de datos.
- Frontend: La interfaz de usuario propiamente dicha. Solo interactúa con su backend y despliega la información que recibe del mismo.

2.4.13 Git y Github

Para poder trabajar distribuidamente, mantener distintas “fotos” del avance del proyecto y dejar el proyecto listo para ser deployado a un Host, se trabajó con Git y Github. Git es un proyecto open source cuya principal función es actuar como un sistema de control de versiones distribuido. Es sin dudas el sistema de control más utilizado del mundo. Github es un portal creado para alojar el código versionado en git de las aplicaciones de cualquier desarrollador. Es muy utilizado en la actualidad, principalmente desde que Microsoft lo compró en 2018 y le dio un soporte que lo distingue de otros portales similares. No solo permite alojar código y distribuirlo en equipos de desarrollo, sino que facilita cualquier forma de hosteo.

2.4.14 Heroku

Heroku es una plataforma en la nube como servicio (PaaS) con el objetivo de ayudar a los usuarios a construir, ejecutar y operar aplicaciones en la nube. Heroku hace que los procesos de implementación, configuración, escalado, ajuste y administración de aplicaciones sean lo más simples y directos posible. Permite el uso de los lenguajes de mayor uso en las aplicaciones, tales como, Javascript, Ruby, Java, Scala, PHP y más.

Características:

- Soporta diferentes lenguajes de programación: Javascript, Ruby, Java, Clojure, Scala, Go, Python, PHP
- Tiene una versión gratuita fácil de usar
- Ejecuta las aplicaciones a través de sus contenedores, también conocidos como Dynos.
- Tiene Dynos que pueden ser de tres tipos: web, worker o cron.
- Ofrece más de 200 complementos con los que ampliar las aplicaciones al instante

- Ofrece varias características de seguridad, incluyendo SSL, autenticación y cumplimiento de PCI

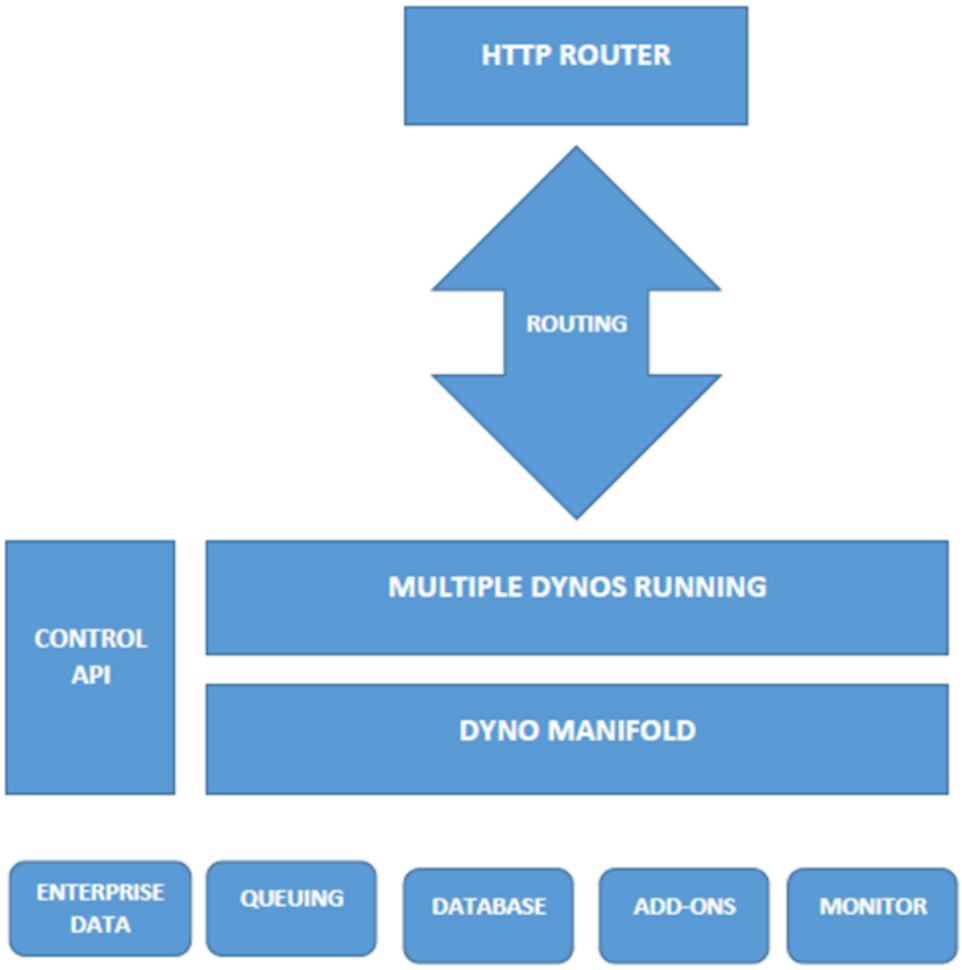


Figura 9.- Componentes arquitectónicos de Heroku

2.4.15 Microsoft Power BI

Power BI es un conjunto de poderosas herramientas de Business Intelligence, en las cuales se puede centralizar diferentes fuentes de datos para la creación de paneles e informes interactivos. De esta forma, la aplicación BI de Microsoft permite visualizar las interacciones y comportamientos de los datos, facilitando su análisis y la toma de decisiones informadas.

Power BI permite conectar a cientos de orígenes de datos en la nube o entorno local, creando informes con objetos integrados o creando objetos personalizados. Power BI, nos permite analizar los datos y obtener patrones “poco visibles” y que ayuden a llegar a conclusiones y la toma de decisiones. A su vez, brinda un potente lenguaje de fórmulas de DAX, que nos proporciona un control total sobre el modelo.

Capítulo 3

PUESTA EN MARCHA Y RESULTADOS

La arquitectura de Big Data seleccionada para este trabajo especial de grado sería del tipo Lambda, ya que la misma permite poder realizar integraciones tanto a nivel de streaming continuo como también tener la posibilidad de deployar ingestas en batch al mismo tiempo. A continuación, se modela un breve flujo el cual permite ver gráficamente lo anteriormente expuesto:

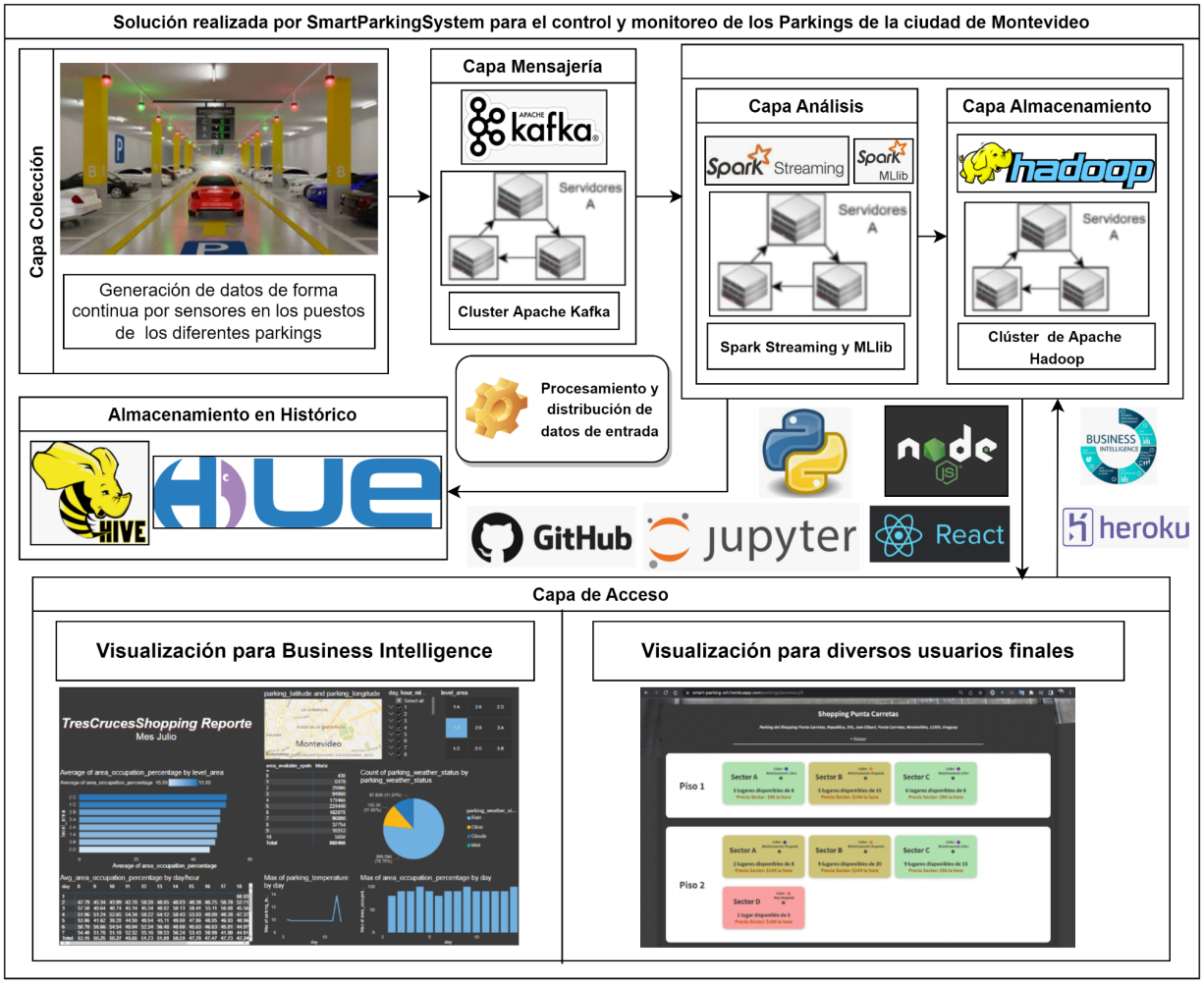


Figura 10.- Arquitectura planteada tesis

Analizando más a alto nivel lo anteriormente expuesto, se puede considerar describir los desarrollos realizados en las diversas capas en las subsecciones que se presentan a continuación y que de forma gráfica se puede observar el flujo de las mismas como lo visto en la figura 3 (página 20).

3.1 Capa colección de datos

Tecnologías:

La capa de colección se basa en la creación de diferentes scripts capaces de realizar acciones según una lógica definida que permita captar estados de diferentes eventos que normalmente acontecen en un Parking que cuenta con sensores digitales o analógicos.

Su arquitectura plantea el uso de diversos sensores que permitan la integración con diferentes módulos físicos que transportan la información de un punto de origen a un punto de concentración de datos. Su modelo se basa en el uso de IoT para establecer un marco de datos estables y confiables dentro de la futura arquitectura de Big Data.

Puesta en Marcha:

Esta capa se encuentra fundamentada en la generación de datos que se encuentran en flujo de la arquitectura de Big Data, en muchos casos estos se encuentran generados a través de sensores de diferentes clases, en el consumo de APIs o de los datos dentro de una red de diferentes aplicaciones de software.

Dentro del servidor las diferentes implementaciones se encuentran realizadas de el directorio /opt/smart-parking/, permitiendo así las diferentes integraciones de la topología.

Para el proyecto, debido a que no se cuenta con data real, se planteó de primera instancia una posible implementación a través del uso de sensores en una maqueta a baja escala, en la cual se pueda demostrar una posible ejecución de un sistema del estilo, para esto se desarrollaron scripts en Arduino IDE con la capacidad de leer los estados de sensores de IR ubicados en lo que serían los puestos del estacionamiento del parking, al tener recepción de los datos se configuró de forma adicional un dispositivo XBee S2C encargado de la transmisión de los datos mediante RF de baja frecuencia.

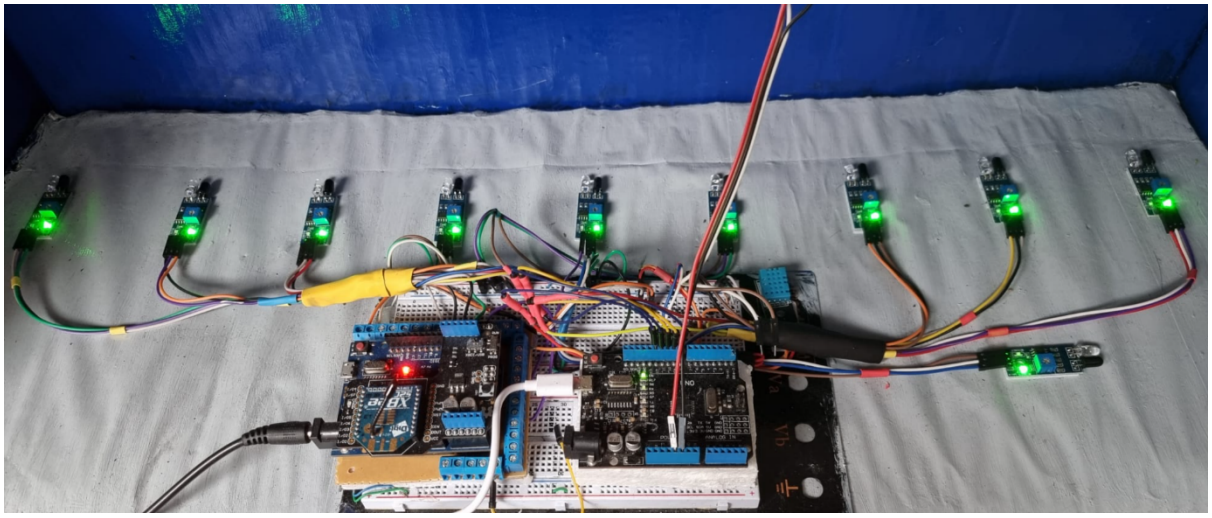


Figura 11.- Maqueta Arduino-XBee Smart Parking Model

Una vez los datos se presentan enviados, estos son recepcionados por un dispositivo de la misma especie conectado a un Raspberry Pi 4, el cual ejecuta un script en Python3.10 que permite la captura de los datos para el envío de los mismos a lo largo de la arquitectura en la capa de mensajería. A continuación, se presenta una imagen con la recepción de datos mediante el RaspberryPI.

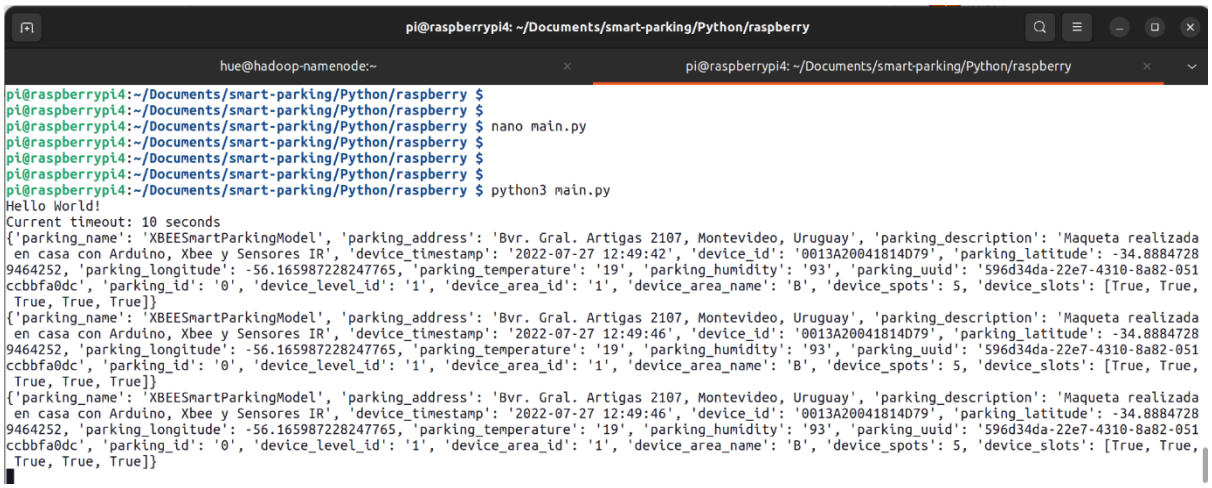


Figura 12.- Recepción de datos en el RaspberryPI 4

Debido a que para la maqueta era necesaria la interacción humana en la simulación de los datos y no se tenía una gran cantidad de puestos de estacionamiento, se realizó un Script en Python3.10 con la posibilidad de hacer la simulación de un Parking, el cual con diferentes secciones de configuración se puede llevar a cabo la integración de una amplia cantidad de puestos para un parking, variando sus datos ya sea desde la cantidad de pisos que este contenga hasta la cantidad de sectores a consumir que se tengan. Este desarrollo se encuentra siendo ejecutado en una máquina virtual que cuenta con Centos7 y Python3 en un virtual environment con las librerías necesarias para su ejecución.

Para pruebas se simularon los parkings de Tres Cruces Shopping y Punta Carretas Shopping como medida de integración en la arquitectura, cada uno contando con 3 pisos y diferentes sectores en su arquitectura, pero que en total en cada uno de ellos se simulan actualmente 90 puestos de estacionamiento.

La base fundamental de este desarrollo es la pensada en la maqueta, siendo cada dispositivo XBee un sector diferente en el envío de datos y los encargados de acumular los estados de los sensores del Smart Parking. Adicionalmente, se consumen diferentes librerías en Python que tienen la capacidad de indicar los días feriados en

Uruguay y arrojar datos relacionados al clima, los cuales permiten una posible alimentación sobre un modelo de Machine Learning (Capa de Análisis) con la capacidad de predecir el porcentaje de ocupación de un área y hacer posible el cálculo de una tarifa dinámica en su totalidad.

3.2 Capa de mensajería

Tecnologías:

En este proyecto estamos planteando el uso de una arquitectura del tipo Lambda, la cual es capaz de procesar la data en tiempo real y poder realizar también una integración de la misma en un contexto batch. De tal manera, lo anteriormente descrito se da a reflejar el uso de Apache Kafka como broker de mensajería, en la cual se pueden definir diferentes tópicos para que sea posible la integración y separación de los diferentes parkings que se consuman dentro del sistema.

Por lo general en el mundo del Big Data en la capa de mensajería se usan diferentes softwares los cuales sean capaces de poder utilizar una arquitectura del tipo publicación-suscriptor (publish/subscriber), su finalidad se centra en poder obtener un concentrador de información (tópico), en el cual diferentes workers de la topología puedan cumplir funciones de agregación de data al mismo tiempo (publicadores), para que otros conectores puedan obtener los resultados reflejados en el concentrador (suscriptor)

En tal sentido, el uso de esta tecnología tiene que cubrir con los estándares de que pueda contener alta disponibilidad que sea fácil a la hora de poder levantarse fácilmente en caso de pérdida de datos, Apache Kafka tiene la capacidad de guardar batches de información en memoria ubicados en el offset de cada tópico, esta es una medida la cual habilita a la arquitectura a ser tolerante a fallos.

Puesta en Marcha:

En nuestro proyecto se destaca el uso de Apache Kafka, como medida de integración entre la capa de colección y almacenamiento, ya que en este caso cada parking se

entiende como un tópico el cual permite obtener diferentes entradas de JSON, que da la recepción de datos desde los diferentes sectores del Smart Parking. Su manera de configuración es de forma sencilla, y no necesariamente debería existir interacción manual sobre Kafka para la creación del tópico.

Esto se debe a que cuando una aplicación tiene acceso al clúster de Kafka y se encuentran con configuraciones básicas estas son capaces de poder hacer la creación del tópico en caso de no existir en su listado. A continuación, se muestran los siguientes puntos de concentración del lado del servidor Smart Parking System:

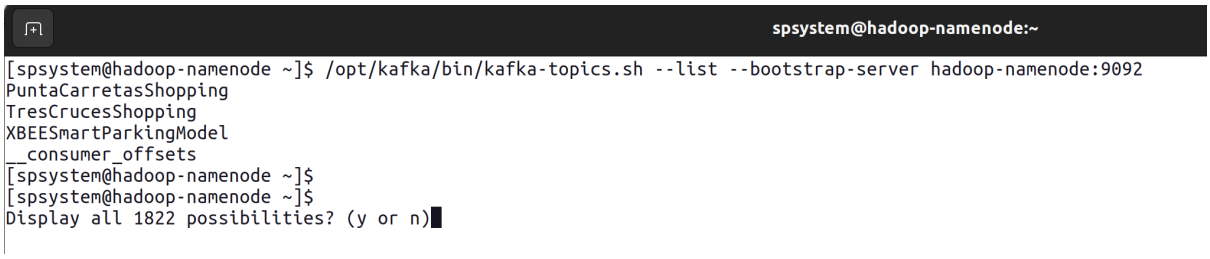


Figura 13.- Tópicos de Kafka

Una vez se encuentra recibiendo datos dentro del tópico, se puede obtener un JSON el cual está siendo transmitido desde la capa de colección de datos hasta el broker de Kafka, la siguiente sería la obtención de un JSON del siguiente estilo:

```
{
   "parking_name":"TresCrucesShopping",
   "parking_address":"La Pasiva, 1825, Bulevar General Artigas, Tres Cruces, Montevideo, 11600, Uruguay",
   "parking_description":"Parking ubicado en el Shopping Tres Cruces",
   "device_timestamp":"2022-07-25 22:32:20",
   "device_id":"FCB85470B7C0E0CC",
   "parking_latitude":-34.8943081,
   "parking_longitude":-56.1664375,
   "parking_temperature":9.73,
   "parking_humidity":66,
   "parking_uuid":"76c41438-3a97-4333-a037-f7858fbb0ba6",
   "parking_id":"1",
   "device_level_id":"3",
   "device_area_id":"2",
   "device_area_name":"B",
   "device_spots":"10",
   "device_slots":[
      false,
      false,
      true,
      false,
      true,
      false,
      false,
      true,
      true,
      false
   ],
   "parking_weather_status":"Clear",
   "parking_weather_status_detailed":"clear sky",
   "parking_wind_speed":4.12,
   "parking_holiday_status":false,
   "parking_holiday_description":null,
   "parking_holiday_type":"L",
   "parking_closed":true
}
```


Figura 14.- Formato de JSON a la entrada del sistema.

Hay que destacar que cada sector del Parking se encuentra enviando datos cada 1 segundo, esto quiere decir que dentro de la entrada de los mismos se puede reflejar un envío constante, generando una gran cantidad de datos por segundo por cada parking. A continuación, se muestra un ejemplo del consumo de uno de los tópicos de la arquitectura:

Figura 15.- Consumo de Tópico Parking Tres Cruces Shopping.

Estos datos al ser persistidos con Kafka en ciertos instantes de tiempo, permiten realizar diferentes aplicaciones que puedan consumir los datos de tópico para realizar acciones de almacenamiento en batch o en temas relacionados a sumarizaciones en tiempo real o análisis pertinentes para una futura visualización en la capa de la arquitectura correspondiente.

3.3 Capa almacenamiento

Tecnologías:

Esta parte de la arquitectura, como su nombre lo indica, se basa en el almacenamiento de los datos que se distribuyen dentro del sistema de Big Data, que para esta ocasión y debido a que necesitamos almacenamiento de grandes volúmenes de datos, además de que su modelo se plantea dentro de la recepción de datos del tipo clave-valor, hace que sea necesaria la integración del sistema de archivos distribuido de Hadoop (HDFS). Este, es capaz de realizar diferentes almacenamientos en un clúster pesado en el cual cuenta con diferentes secciones de la misma.

Hadoop es una estructura de software del tipo Open Source capaz de realizar grandes almacenamientos de cualquier tipo de datos y a su vez hace hábil la ejecución de diferentes aplicaciones dentro del clúster a través de YARN. Su modelo se centra en un nodo llamado Namenode el cual cumple funciones de gestión de la data distribuida en los diferentes Datanodes de un clúster.

Para tener un clúster confiable de Hadoop se aconseja realizar integraciones impares respecto a la cantidad de Datanodes y Namenodes. En nuestro caso debido a que tenemos un ambiente de estudio se implementa un Datanode y un Namenode por falta de disponibilidad de hardware.

El sistema de archivos distribuidos se hace tan importante debido a las siguientes características:

- Poder de cómputo distribuido: Esta propiedad permite que se puedan realizar diferentes procesamientos distribuidos de datos a gran velocidad dentro de un clúster.

- Tolerancia a fallas: Debido a que cuenta con la arquitectura basada en alta disponibilidad, en caso de presentar fallas en un nodo, el mismo automáticamente es capaz de poder redirigir todos los procesamientos a nodos activos del clúster, ya que los Datanodes están configurados para que generen múltiples copias de forma rápida y automática.

- Flexibilidad: A diferencia de las bases de datos relacionales, HDFS no tiene que procesar previamente los datos antes de almacenarlos. Puede almacenar tantos datos como desee y decidir cómo utilizarlos más tarde. Eso incluye datos no estructurados como texto, imágenes y videos.

- Bajo Costo: Ya que su implementación se centra en software de código abierto, hace que sea de fácil implementación dentro de un clúster.

- Escalabilidad: Es posible hacer crecer de forma fácil un sistema de archivos distribuidos generando la integración de grandes volúmenes de datos agregados al sistema.

Puesta en Marcha:

Para el proyecto de Smart Parking System, se realiza la integración de un clúster sencillo de Hadoop dentro de un servidor cuyo sistema operativo es Centos 7. Para esta oportunidad se integra un Namenode y un Datanode en el mismo host (en clústeres más robustos es necesaria la ejecución de más hosts con la función de Datanode y Nodemanager). Su configuración se basa en la instalación previa de diferentes dependencias, como lo es el caso de Java JDK (ya que se encuentra desarrollado en Java) y diversos paquetes para la compilación del código central del HDFS.

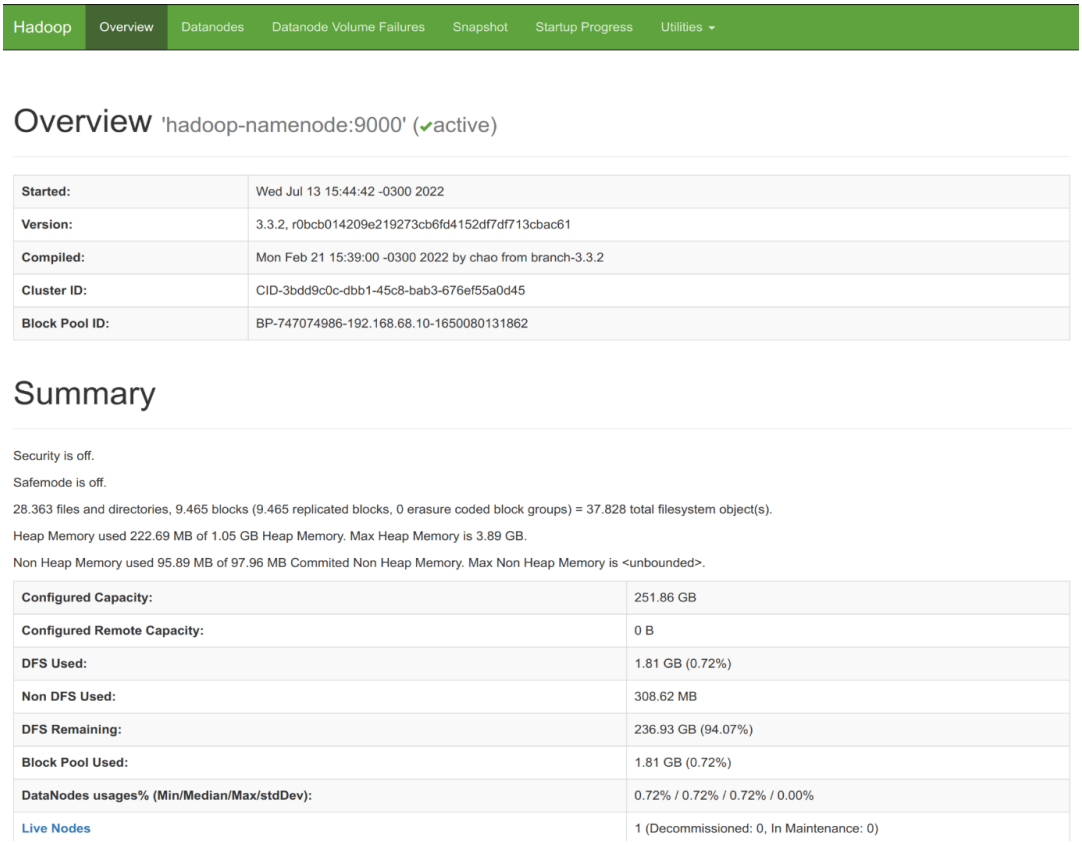


Figura 16.- HDFS

Una vez instaladas las instancias y siendo levantadas en el servidor, se puede comprobar accediendo al sistema de archivos distribuidos mediante el siguiente enlace: http://smartpsystem.tplinkdns.com:14200/explorer.html#/. Adicionalmente, si se desea acceder a la visualización de las tareas de MapReduce del clúster se puede acceder al siguiente enlace: http://smartpsystem.tplinkdns.com:14202/clúster.

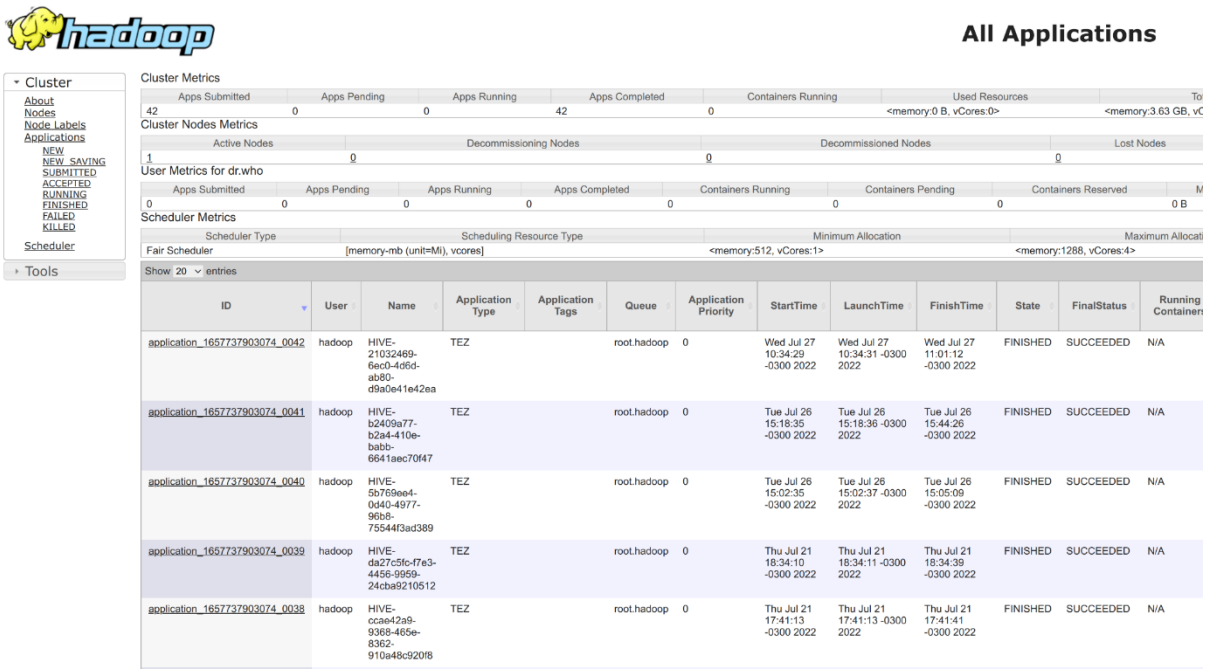


Figura 17. Yarn

Ahora, para la ejecución del almacenamiento e integración con la capa anterior, es necesario el uso de Spark Streaming para poder habilitar diferentes procesos de almacenamiento en batch que hagan control de los datos agregados dentro del sistema de archivos distribuidos. Esto se centra en el uso de algoritmos en python que permitan la conexión a los tópicos de los parkings de Kafka.

Este proceso cada 5 minutos acumula lotes de información con la finalidad de generar un Dataframe de Spark que sea posible describir mediante compresión del tipo parquet para mejorar la performance a la hora de almacenar. A continuación, se muestra una figura con los diferentes parquets almacenados en las diferentes particiones del HDFS.

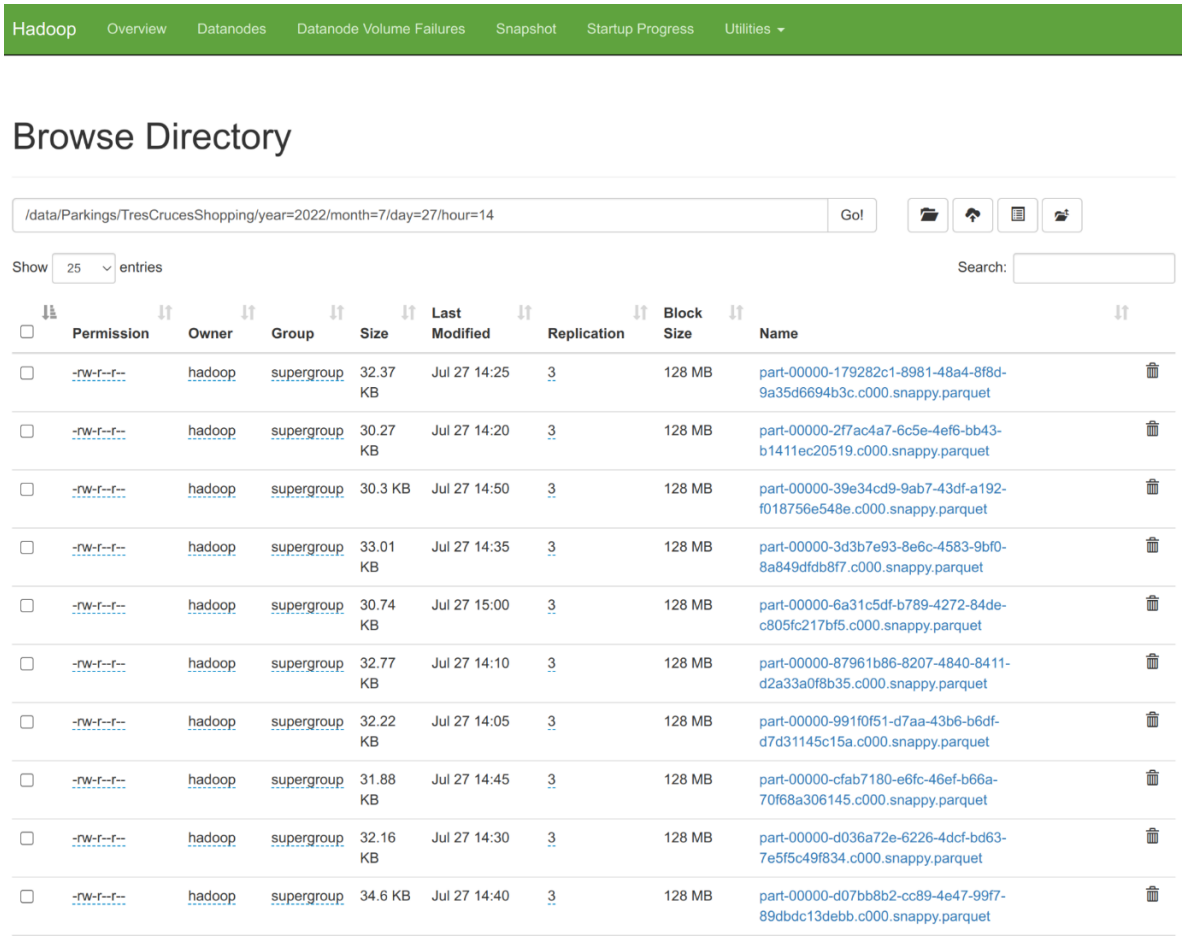


Figura 18. Archivos Parquets en la partición de datos de Hadoop.

A este último proceso se le hacen diversas agregaciones y sumarizaciones en tiempo real que permiten acumular más datos que los que se tienen en la entrada del sistema. Su finalidad es sumar diferentes datos usables para futuras implementaciones en modelos de Machine Learning dentro de la plataforma o incluso para mejoras en la generación de dashboard recreativos para los usuarios que consumen el sistema.

3.4 Capa de análisis

Tecnologías:

Las tecnologías usadas en la capa de análisis se centran en funcionamientos diversos para su tipo de estructura. De esta manera, las mismas pueden ser planteadas en su orientación en batch o incluso hacer diferentes sumarizaciones y agregaciones para

mostrar en tiempo real dentro de la capa de visualización. Esto último, se puede profundizar en los siguientes puntos:

1. Análisis en Batch:

Este análisis se centra en el uso de diferentes herramientas que permiten realizar reportes sobre los datos almacenados en Hadoop y en esta sección se plantea el uso de Apache Hive y Hue, herramientas que se encuentran orientadas dentro del ecosistema de Hadoop y que le dan vida a los datos distribuidos en el clúster.

Apache Hive se basa en un framework que permite realizar diferentes Querys sobre los datos que se encuentran almacenados en HDFS. Su uso radica en poder realizar consultas del tipo SQL pero usando el lenguaje HiveQL, el cual presenta algunas diferencias a la hora de ejecutar.

Estas consultas realizadas al HDFS, son traducidas internamente por el servicio HiveServer2 generando así la comunicación con el ResourceManager de Hadoop (YARN) en la obtención de recursos para la ejecución de tareas del tipo MapReduce.

Su arquitectura radica en poder generar un metastore de datos el cual a la hora de realizar consultas el resultado es cargado en una base del tipo relacional y junto a la generación de metadata se puede ir observando las agregaciones realizadas de la plataforma. Dicho esto, es importante mencionar que es necesaria la integración de una tabla (configurada en Hive pero cargada a nivel de MySQL) la cual mapea las columnas de la información almacenada en HDFS, indicando así que se encuentra creada como un “external table” que apunta a un directorio global del sistema de archivos distribuidos.

2. Análisis en Tiempo Real:

Para este punto, es posible la integración de scripts de python haciendo uso de Spark Streaming generando una posibilidad de agregaciones de valor a los datos obtenidos. Esta sección, se centra en la ejecución y construcción de un JSON que permite ser enviado en tiempo real al Backend de la WebApp del sistema generando una

visualización de datos en tiempo real mediante los cálculos realizados con Apache Spark.

Muchos de los cálculos garantizados en el proyecto se centran en la generación de porcentajes de ocupación por sector/área o piso del estacionamiento en el cual se encuentran los sensores (simulados en este caso). Adicionalmente, se generan estadísticas que son pertinentes para el estudio y agregaciones de variables que le dan vida al Frontend.

Puesta en marcha:

1. Análisis en Batch:

A nivel del análisis en batch, se realiza la creación con Apache Hive de una base de datos llamada “smartparkingsystem”, la cual contiene tablas relacionadas con los nombres de los estacionamientos del sistema y que cuenta con diferentes particiones correspondientes al año, mes, día y hora de los datos almacenados en HDFS.

Para realizar la creación de las tablas, se realizó la ejecución de las mismas a través de HUE (Hadoop User Experience) con otras secciones de la arquitectura de Smart Parking System. A continuación, en la figura 19 se detalla un ejemplo de la ejecución de este script:

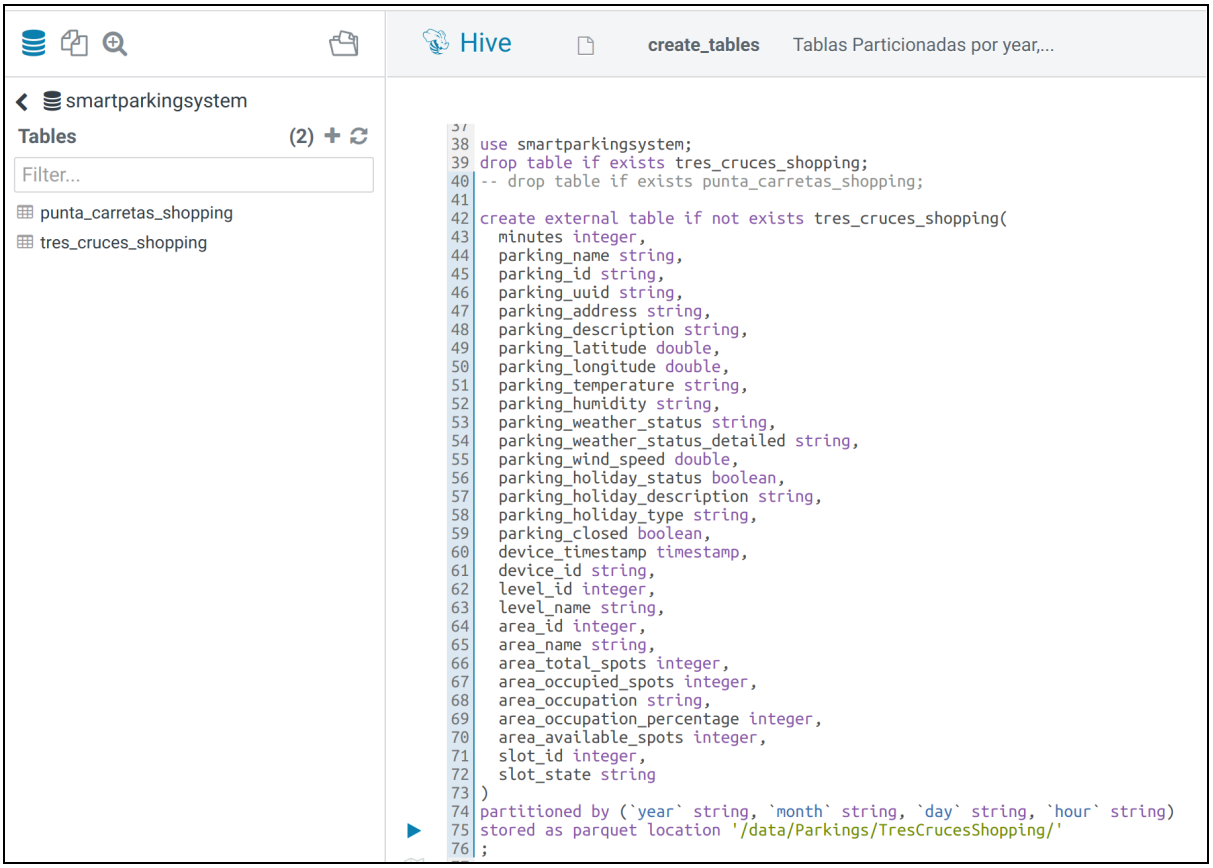


Figura 19.- Creación de tablas en metastore de Hive

HUE habilita poder realizar diferentes consultas mediante el lenguaje HiveSQL en una plataforma amigable para el usuario que la consume.

En caso de querer ingresar a la misma se puede acceder mediante el siguiente enlace http://smartpsystem.tplinkdns.com:14201/ con el usuario viewer y la password viewer. La figura 20 muestra un ejemplo de lo anteriormente mencionado.

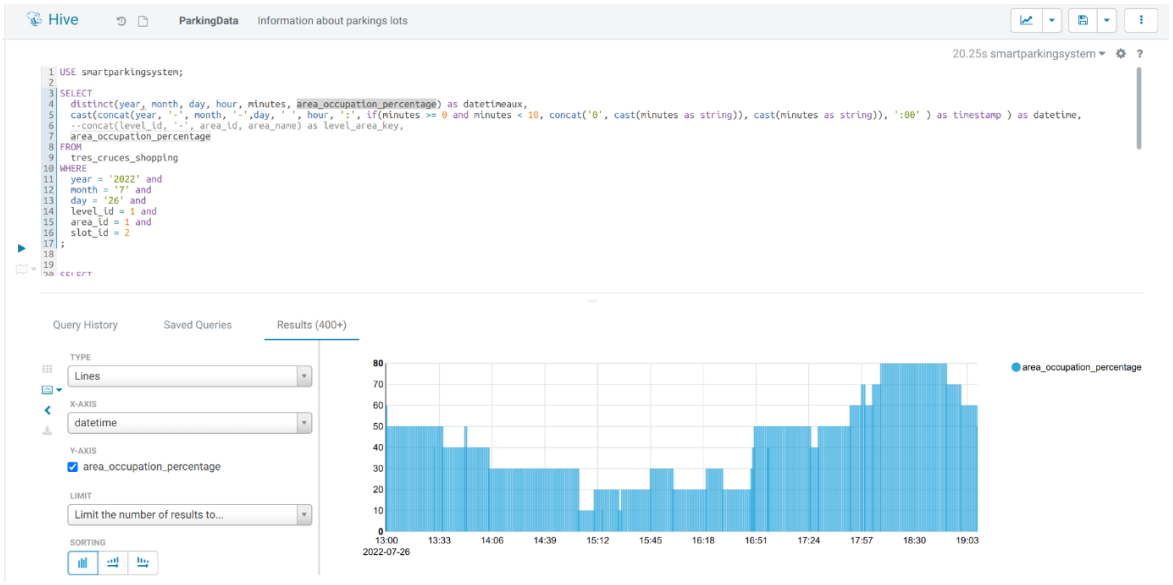


Figura 20.- Ejemplo de consulta en Hive a través de HUE.

Adicionalmente, es posible la agregación de valor con Apache Spark para poder levantar la data de forma distribuida y ejecutar diferentes scripts que permitan analizar los datos almacenados en Hadoop.

Con los datos almacenados se pueden realizar análisis de correlación, ejecutando los mismos a través de Jupyter Notebook. Más adelante en la capa, se aborda la posibilidad de predecir el porcentaje de ocupación de un área del estacionamiento para así poder realizar el cálculo correspondiente a la tarifa dinámica.

2. Análisis en Tiempo Real:

Se realizó la ejecución de un script que en tiempo real permite poder actualizar y enviar al Backend de la WebApp un estado de la plataforma respecto al último estado de los puestos según entran los datos en Apache Kafka.

En principio, se conecta directamente al tópico del parking que se consume y realiza la ejecución de tareas y procesamiento por cada RDD que está entrando en Apache Kafka. Esto habilita a poder iniciar la creación de un JSON que continuamente se actualiza en memoria y con la finalidad de ser enviado a través del consumo de la API de la WebApp. Un ejemplo del JSON resultante se puede apreciar en el Anexo 1.

Machine Learning:

Si bien el foco principal de este proyecto, no se centra en la ejecución de modelos de Machine Learning, se ha realizado un modelo de regresión lineal básico basado en Pyspark MLlib.

La razón principal, es mostrar cómo cualquier proceso de Machine Learning puede ser fácilmente integrado en la arquitectura de Big Data desarrollada, con tal de agregar un valor sustantivo a la solución lograda.

Como se tienen los datos almacenados en Hadoop es necesaria una limpieza previa de los mismos para el procesamiento de los script de pyspark. De tal manera, se procede a realizar una eliminación de datos duplicados y de generar algunas variables necesarias para el modelo.

Para esto los datos son almacenados dentro el path "/machineLearning/Parkings/" del HDFS. En ese directorio se escriben los datos por estacionamiento de forma limpia en diferentes parquets particionados de forma automática por Hadoop. La siguiente figura ilustra algunas integraciones de lo comentado con anterioridad:

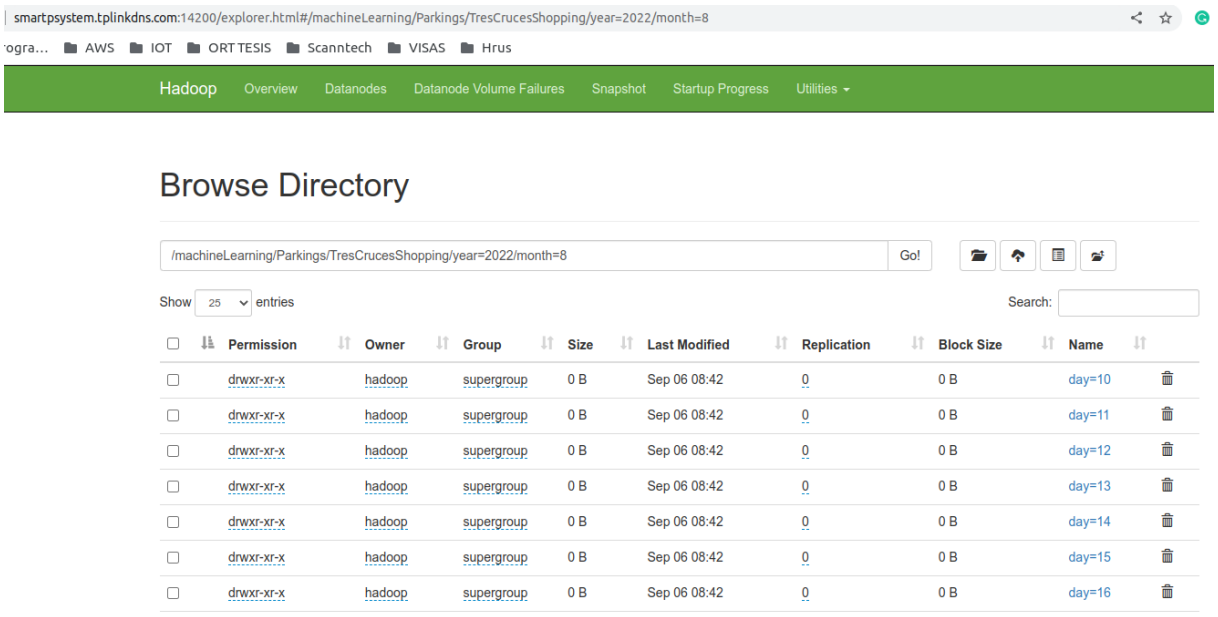


Figura 21.- Directorio del HDFS con datos limpios para futuro procesamiento del Machine Learning.

Luego, se desarrolló un script de python basado en MLlib, el cual con paquetes internos de la librería, permite ejecutar el entrenamiento y testeo del modelo de regresión dentro de la plataforma. Realizada esta ejecución, las predicciones basadas en el porcentaje de ocupación futuro que pueda mostrar cierta área y que permita el cálculo de una forma moderna del precio de los puestos de forma dinámica, se guardan dentro de un directorio del HDFS.

Este directorio se centra en las predicciones de los datos del mes anterior, con la data de todos los días, horas y minutos en los cuales los usuarios del estacionamiento hicieron uso del mismo. Su resultado se puede observar en el path "/predictions/Parkings/". Para una mejor referencia, la figura 22 que describe lo anteriormente expuesto:

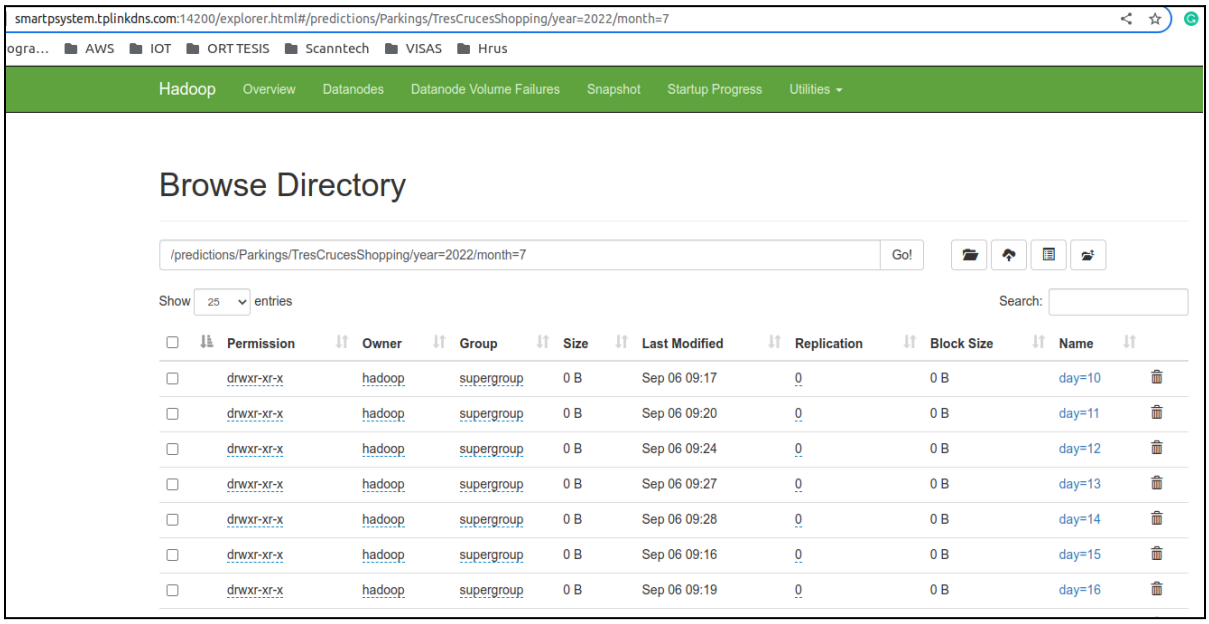


Figura 22.- Directorio del HDFS con las predicciones realizadas por la Regresión Lineal.

Todas estas ejecuciones que se describen con anterioridad, se encuentran automatizadas a nivel del Cron de Linux, permitiendo así su ejecución en diferentes momentos de tiempo para el estacionamiento que lo consuma. A continuación, se muestra el crontab ejecutado por Centos:

```
[spsystem@hadoop-namenode ~]$ sudo crontab -l
[sudo] password for spsystem:
0  *  *  *  *  /home/spsystem/Automatismo/clearcache.sh

# Data Preparation for Machine Learning in parkings

0  0  1  *  * spark-submit /opt/smart-parking/Python/MachineLearning/ml_datapreparation.py --parking TresCrucesShopping
0  0  1  *  * spark-submit /opt/smart-parking/Python/MachineLearning/ml_datapreparation.py --parking PuntaCarretasShopping

# MLLIB Linear Regression

0 1 1 * * spark-submit /opt/smart-parking/Python/MachineLearning/ml_mllib_linear_regression.py --parking TresCrucesShopping
0 1 1 * * spark-submit /opt/smart-parking/Python/MachineLearning/ml_mllib_linear_regression.py --parking PuntaCarretasShopping

# Generate area summary json
* 8-22 * * * python3.10 /opt/smart-parking/Python/MachineLearning/ml_generate_json.py --parking TresCrucesShopping
* 8-22 * * * python3.10 /opt/smart-parking/Python/MachineLearning/ml_generate_json.py --parking PuntaCarretasShopping
[spsystem@hadoop-namenode ~]$
```

Figura 23.- Linux Crontab para Machine Learning

3.5 Capa de acceso a los datos

La capa de acceso es la pieza final de cualquier arquitectura de Big Data. Su objetivo principal es ser un puente entre los usuarios finales y el sistema.

Una vez que se generan los datos, se ingestan, analizan y están listos para ser consumidos, necesitamos una forma de comunicar la información hacia los posibles sistemas periféricos que la necesiten.

Normalmente la comunicación es a través de un protocolo de API y los sistemas periféricos pueden ser una MobileApp, una WebApp o un Dashboard de control.

Para la capa de acceso del proyecto, se desarrolló una API capaz de consumir los datos del sistema, almacenarlos y dejarlos disponibles para una WebApp, que va a ser la responsable de desplegarlos en una interfaz de usuario lo más amigable posible.

Una vez finalizado el desarrollo de la interfaz, es importante plantearse cómo los usuarios finales van a poder acceder a la misma. Normalmente, esto se da mediante un proceso de hosting y dominio. El dominio es la dirección a donde el sistema va a publicar la información y también la forma de acceder a la interfaz. El hosting es el lugar físico donde el código va a ser ejecutado y el que permite que mediante un navegador se pueda correr la WebApp.

Tecnologías:

Es importante antes de justificar las tecnologías a utilizar en la capa de acceso, hacer un recuento de dónde se está parado en el sistema. Los datos del parking luego de ser ingresados, procesados y agregados con distintos procesos tienen que quedar en un formato listo para ser consumido.

El formato seleccionado por convención y porque es lo que más se utiliza en la práctica es un formato JSON con toda la información necesaria para cada estacionamiento. En la sección puesta en marcha se hace un desarrollo más específico de todos los componentes del mismo.

Luego de definir el formato de los datos, el siguiente paso es definir las tecnologías en las que se va a construir la WebApp.

Para la base de datos, se definió un sistema de datos relacional, ya que solo se va a almacenar la información procesada del sistema de Big Data. Se definió utilizar PostgreSQL ya que es un sistema de bases de datos relacionales open-source con más de 30 años, lo que le da una robustez, performance y confiabilidad muy importante. Para acceder a las distintas bases y tablas, se utilizó el software TablePlus, por su facilidad.

Para el Frontend y el Backend se utilizó el muy conocido PERN Stack por sus siglas en inglés. El PERN Stack es un conjunto de marcos/tecnologías utilizados para el desarrollo web de aplicaciones que consta de PostgreSQL, Express JS, React JS y Node JS como sus componentes.

React es una librería de JavaScript creada por Facebook que se utiliza para crear componentes de interfaz de usuario. Node es un entorno de ejecución para JavaScript que permite ejecutar JavaScript del lado del servidor (backend) y no en un navegador. Express es un Framework utilizado normalmente sobre Node y permite desarrollar cualquier backend de manera ágil y sencilla.

Las ventajas de utilizar este stack son: Permiten utilizar solo un lenguaje de programación, JavaScript, tanto en el front como en el back. Como este es el mismo lenguaje que aceptan los navegadores, no hay necesidad de compilaciones u otros conflictos al ponerlo en marcha. Esto garantiza una integración de todas las partes sin grandes problemas de configuración.

La justificación principal para usar el PERN Stack es la gran experiencia en el mismo que presenta el equipo y su extrema popularidad en el desarrollo de las WebApp actuales, factor que asegura soporte y mejoras continuas a lo largo del tiempo.

La última tecnología a mencionar es el proveedor de hosting elegido. Para esta tarea se utilizó Heroku. Heroku es un PaaS (platform as a service) que permite hostear el código en la nube y se encarga de toda la infraestructura que implica dejar el servicio web disponible. Se eligió Heroku porque es muy sencillo subir el código de la App, es muy robusto, es confiable ya que es propiedad de Salesforce, una de las empresas de software más importantes de Estados Unidos y tiene un plan gratis (al menos hasta la fecha de publicación del presente informe) que se ajusta mucho al alcance de la Tesis.

Si en el día de mañana se quisiera llevar el sistema al mundo real, Heroku también tiene un plan de pago muy interesante y económico para obtener un dominio propio y más procesamiento tanto para el Backend como el Frontend.

Por último, se consideró la herramienta de Microsoft Power Bi en su modalidad desktop para el desarrollo de paneles interactivos de business intelligence. Además, el reporte gerencial resultante fue subido a la nube de manera gratuita gracias a las funcionalidades de Power Bi Server.

Puesta en marcha

Lo primero que se definió es el formato JSON para la transferencia de datos del sistema de Big Data al Backend de la capa de acceso. Las características principales del JSON fueron las siguientes:

- Cada JSON corresponde a la información de un estacionamiento específico.

- Información del estacionamiento: nombre, dirección, si está cerrado, si el día es feriado, clima, ubicación.
- Si el tarifado dinámico aplica o no al estacionamiento.
- Los coeficientes por los que se multiplica el precio base dependiendo de qué tan lleno está cada área del estacionamiento.
- Estimación del porcentaje de ocupación de cada área del estacionamiento. Esta información es el resultado de un modelo de Machine Learning en la capa de análisis.
- Lista de niveles (pisos, por ejemplo) del estacionamiento con información de lugares libres/ocupados.
- Cada nivel tiene una lista de áreas del estacionamiento con la siguiente información:
    - Información de lugares libres/ocupados.
    - Nombre, color o distintivo, descripción.
    - Precio por día, precio base por hora de los lugares del área, precio por mes.
    - Si aplican o no horas gratis y cuáles son las mismas.
- Cada área tiene a su vez una lista de lugares:
    - Estos lugares son efectivamente, donde el auto está estacionado.
    - Cada lugar tiene un nombre y una referencia a su área.
    - También tiene un tipo, ya que puede ser para autos, camiones o bicicletas/motocicletas.

    - El precio del lugar está determinado por el porcentaje de ocupación del área.

    - La información más importante de cada lugar es si está ocupado o no.

Un ejemplo del JSON previamente mencionado se puede apreciar en el Anexo 1.

Con el JSON definido, se procedió a construir el Backend y la base de datos. Primero se trabajó localmente y cuando se llegó a una versión estable se envió el código a Heroku para dejarlo disponible tanto al Frontend como al sistema de base de datos.

Las rutas más importantes de la API creada en el Backend son las siguientes (Todas las rutas tienen el prefijo: “/api/v1/parkings”):

- POST

    - POST “/:externalId”: Esta es la ruta utilizada por el sistema de Big Data para enviar el JSON. Es la encargada de interpretar el mismo y almacenarlo en la base de datos. El parámetro “externalId” es un identificador de referencia para cada estacionamiento. En la figura 24 se puede apreciar la base de datos mediante el software TablePlus.

- GET

    - “/”: Utilizada por el Frontend para obtener una lista de todos los estacionamientos del sistema.

    - GET “/:parkingId”: Utilizada por el Frontend para obtener un solo estacionamiento con identificador igual a “parkingId”.

    - GET “/:parkingId/:levelId”: Utilizada por el Frontend para obtener un nivel con identificador “levelId” de un estacionamiento con identificador igual a “parkingId”.

- GET “/:parkingId/:levelId/:areaId”: Utilizada por el Frontend para obtener un área con identificador “areaId” de un nivel con identificador “levelId” de un estacionamiento con identificador igual a “parkingId”.

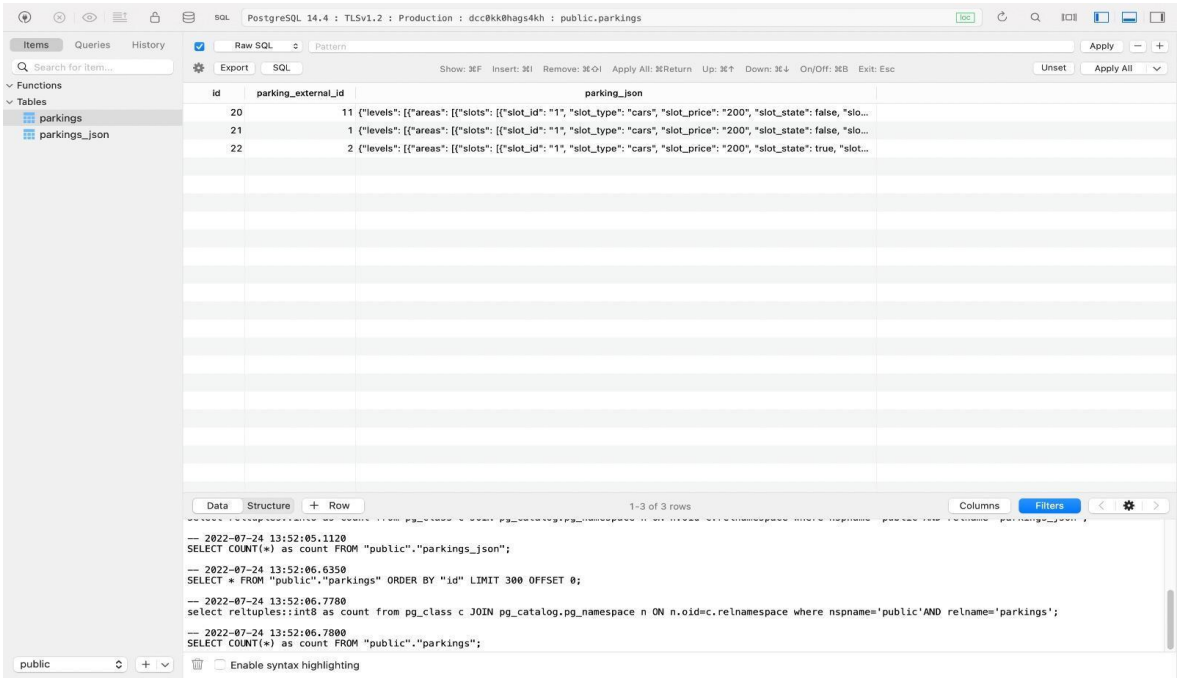


Figura 24 - Base de Datos Postgres (TablePlus)

Con el backend desarrollado, se inició el trabajo para la interfaz de usuario con React JS. De la misma forma que anteriormente, una vez finalizado se hosteó el código en Heroku. En la figura 25 se puede apreciar el panel de control de una de las Apps en heroku.

El Backend y posteriormente el Frontend, van a estar hosteadas en distintas Apps de Heroku, a modo de mantener las lógicas separadas e intercambiables por otras tecnologías a futuro si esto fuera necesario (No solo a nivel de herramienta de hosteo, sino también a nivel de lenguaje de programación).

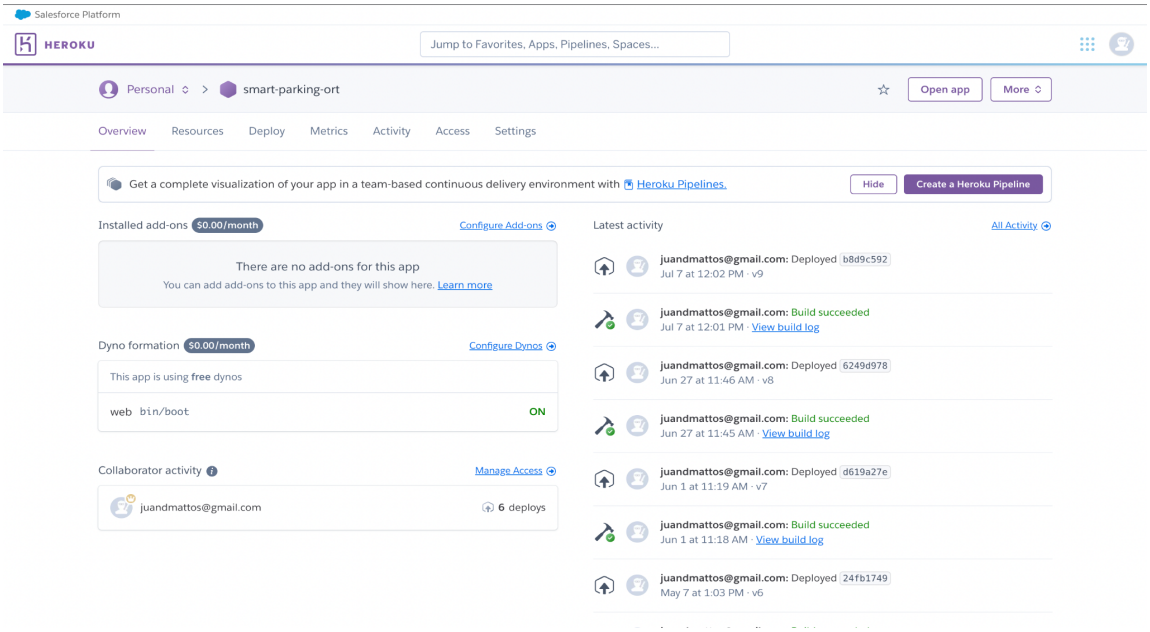


Figura 25 - Panel de control Heroku Frontend

Toda la información desplegada en el Frontend es near-to-real-time. El umbral utilizado es de 5 segundos. Esto quiere decir que cada 5 segundos la página se actualiza y con ella toda la información pertinente, en particular, el estado de ocupación de los estacionamientos y el precio de cada área si el tarifado dinámico está activado.

El Frontend consiste principalmente de las siguientes pantallas:

- Pantalla principal: Es la página con la que los usuarios empiezan a interactuar con la WebApp. Contiene información general, información del clima en tiempo real y una lista de los estacionamientos en el sistema. Cada estacionamiento despliega información pertinente como el nombre, la ubicación, el número de lugares y cuales están disponibles. También permite al usuario observar si el estacionamiento está cerrado. Esto se puede apreciar en la figura 26.

- Pantalla de control de cada estacionamiento: Esta página consiste en un resumen del estado de cada piso del estacionamiento, subdividido en áreas. Esto se puede visualizar mejor en la figura 27. Cada área muestra la siguiente información:

- Nombre del área.
- Distintivos del área.
- Información de tarifado.
    - Esto se despliega mediante un Tooltip.
    - Se menciona si tiene horas gratis, cuál es el precio mensual y si el día es feriado o no.
- Precio actual
    - Dinámico en base al porcentaje de ocupación y los coeficientes de tarifado de cada estacionamiento.
    - A modo de ejemplo con 4 niveles de coeficientes:
        - 0% a 20% área ocupada ⇒ Primer nivel de coeficiente (Casi el precio base).
        - 20% a 50% área ocupada ⇒ Segundo nivel de coeficiente.
        - 50% a 80% área ocupada ⇒ Tercer nivel de coeficiente.
        - 80% a 100% área ocupada ⇒ Último nivel de coeficiente (La cota máxima de precio dinámico posible por área).
- Lugares disponibles sobre lugares totales.
- Código de colores muy intuitivo para el usuario:
    - Verde: Área vacía o relativamente libre.
    - Amarillo: Área relativamente ocupada.
    - Rojo: Área muy ocupada o totalmente ocupada.

- Pantalla de área: Como muestra la figura 28, esta página es una forma visual de ver desplegados todos los lugares de un área en particular. Es una información interesante para desplegar en distintos puntos estratégicos de cada área del estacionamiento. Principalmente se despliega para cada lugar:
    - Nombre del lugar y área correspondiente.
    - Precio del área.
    - Tipo del lugar (auto, camión, bicicleta/motocicleta)
    - Estado del lugar (ocupado o libre)

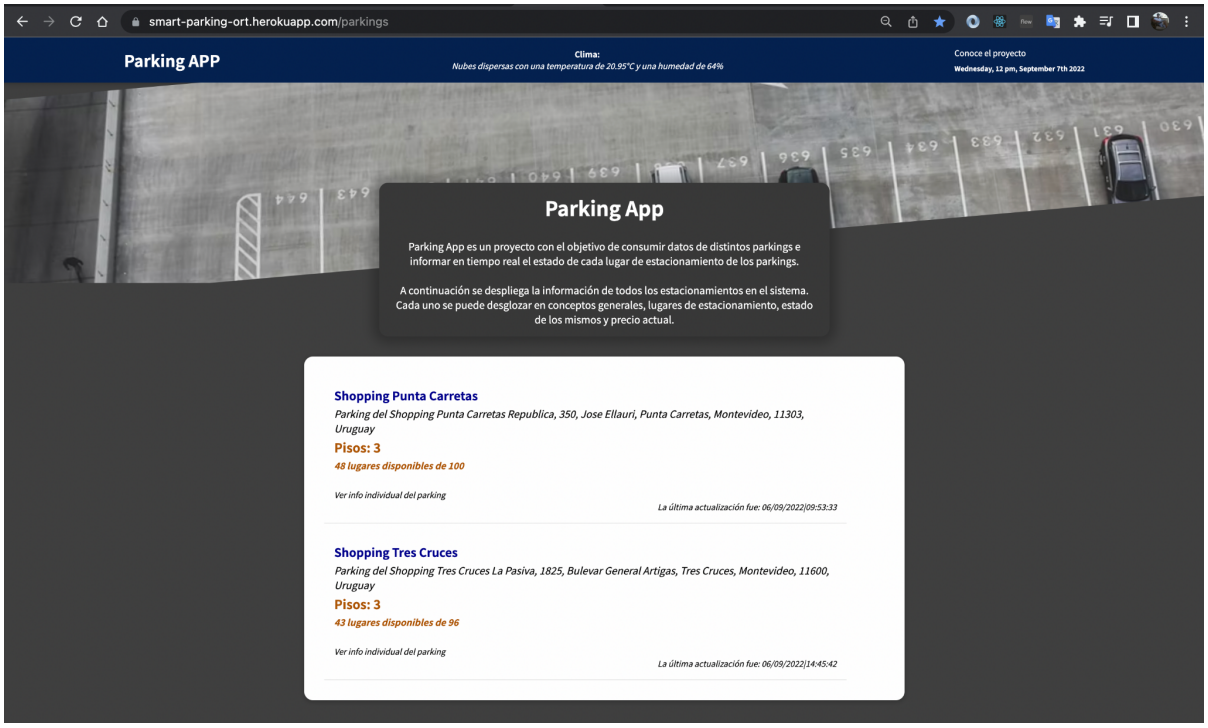


Figura 26 - Página principal de la WebApp

Figura 27 - Pantalla de control de un estacionamiento específico

Figura 28 - Pantalla de área de un estacionamiento específico

De la misma forma que el Backend, el Frontend creado con React JS también fue construido localmente y luego hosteado en Heroku.

En ambos, se priorizó la seguridad y es por esto que se implementó un protocolo HTTPS para que en la comunicación entre los usuarios y el sitio web sobre internet se pueda proteger principalmente la integridad y la confidencialidad de los datos.

URL de acceso: https://smart-parking-ort.herokuapp.com/parkings.

Para el reporte gerencial, como se mencionó anteriormente, la herramienta elegida fue Power BI de Microsoft. A su vez, se consideraron datos generados para un único shopping y por todo el mes de Julio 2022 importado desde una sola base de datos en formato parquet.

El reporte permite a la gerencia apreciar indicadores de ocupación (expresados en porcentajes y en cantidades), analizar la variación del clima y temperatura, ver cómo afectan a estos indicadores y situar el parking en el mapa según su geolocalización (latitud y longitud) entre otras informaciones que se desprenden del reporte.

Dado que Power BI es una herramienta de gestión de datos que permite el uso interactivo, se añadieron filtros por día, hora y por área del estacionamiento, permitiendo que todos estos indicadores mencionados en el párrafo anterior pudieran brindar una mayor información.

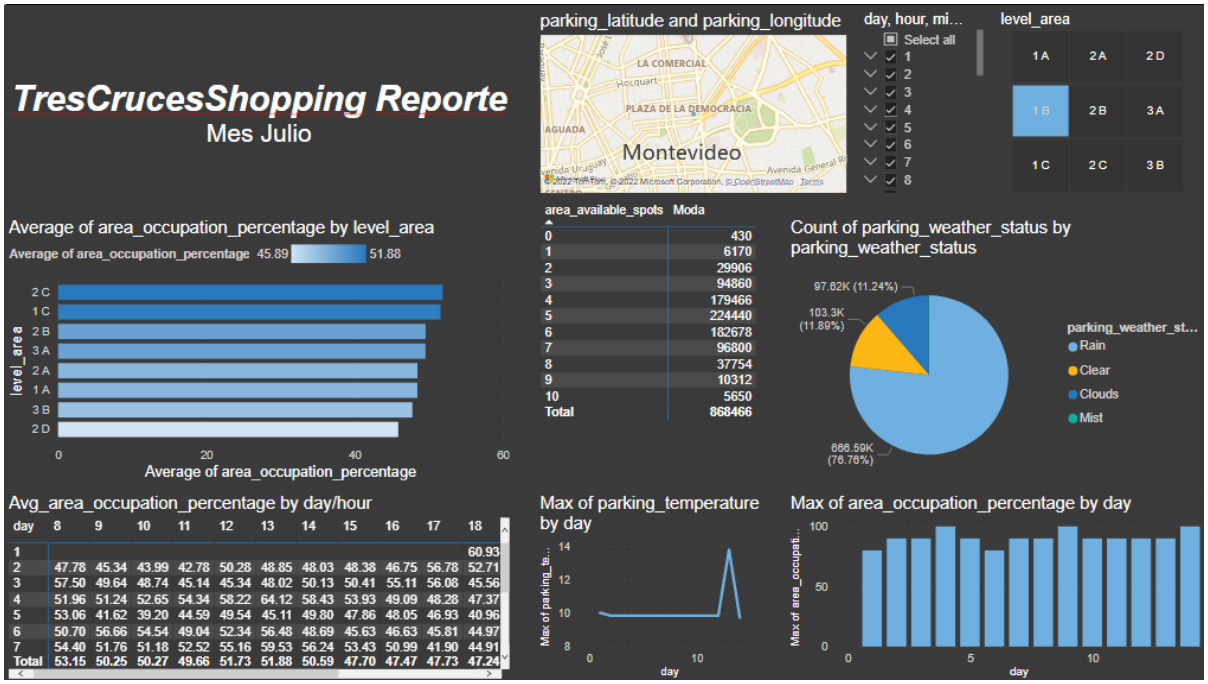


Figura 29 - Dashboard BI

URL de acceso:

https://app.powerbi.com/view?r=eyJrIjoiOGRhOTdlMDYtYTU3OC00NWMzLWFiNTMtYjRkMzFiMDc2OTNlIiwidCI6IjhiMTlkMGQ3LWY0NGYtNGJjYS1hZjM3LWEzY2JlNTAwOWIzOCJ9

Capítulo 4

CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

Para las capas de colección de datos y mensajería, la principal conclusión a la que se puede llegar es que se logró generar una gran cantidad de datos de estacionamientos y se pudo implementar Kafka para manejar la publicación y suscripción de los mismos hacia las otras capas.

Los datos fueron creados de la forma más orgánica posible, simulando las condiciones del mundo real de forma fidedigna.

En las capas de análisis y almacenamiento, se pudo no solo procesar los grandes volúmenes de datos generados en el sistema (aproximadamente 2 millones de registros por día), sino que también aplicarles transformaciones, un modelo de Machine Learning y enviarlos mediante una API a la capa de acceso.

También se puede destacar que se realizó el almacenamiento de los datos mediante Hadoop y el mismo quedó disponible para su consumo con herramientas tales como Hive o HUE.

Para la capa de acceso se puede acotar que toda la misma pudo ser desarrollada en tiempo y forma localmente, pudiendo versionarse con git y github y colocarse en producción mediante Heroku.

Como conclusiones generales, es que toda la arquitectura se implementó utilizando productos open source (con excepción de Power Bi), el plan libre de Heroku y utilizando como servidor principal una computadora personal y la nube. Es importante destacar esto para tomar dimensión de los recursos disponibles y el resultado final.

La aplicación final permite procesar la información proveniente del sistema de Big Data, almacenarla y actualizar todos los datos de los estacionamientos en el Frontend, para

que el usuario tenga una experiencia casi real time de actualización en la interfaz con la que interactúa con el sistema.

El tarifado dinámico no solo se da con información real time del grado de ocupación del estacionamiento, sino que se sentaron las bases para migrar a un modelo de Machine Learning capaz de predecir a futuro cuál va a ser el grado de ocupación y ajustar el precio en base a eso.

En general, se puede concluir que se desarrolló una arquitectura completa de Big Data con Machine Learning. Se pudieron implementar todas las capas de la arquitectura, una interfaz de usuario para consumir el sistema de forma simple e intuitiva y una interfaz de BI para la gerencia.

Todo el trabajo se pudo generar en tiempo y forma, siguiendo el cronograma que se propuso al principio del proyecto. Al ser un trabajo de tesis acotado en el tiempo, es muy positivo haber podido cumplir con todo el alcance propuesto.

Como última conclusión, se puede apreciar en el informe que todos los conocimientos adquiridos en el máster fueron utilizados en mayor o menor medida para el producto final.

4.2 Recomendaciones

La recomendación principal a la hora de continuar este proyecto sería insertarlo en el mundo real. Poder tomar los datos de un número acotado de estacionamientos de prueba, y comprobar que todo funciona tal cual fue construido.

Luego de esta prueba inicial, se debería proceder a migrar el sistema. Como mencionamos anteriormente y debido a los recursos disponibles, el hardware y software donde está construida nuestra solución final no es el óptimo.

Es por esto, que lo ideal sería montar esta misma arquitectura con herramientas basadas en nube con mayor poder de cómputo, con un hardware más a medida de la solución y por supuesto, con una mayor cantidad de estacionamientos.

Mientras la arquitectura se maneje en batch, sería importante mejorar el particionado dentro de Hadoop para lograr una mayor eficiencia a la hora de los procesos de lectura y escritura.

Esto a la larga optimizará y hará más rápidos los procesos de batch y generación de informes/datos que son el combustible para cualquier modelo de Machine Learning que se quiera generar a futuro y las próximas capas.

Para finalizar, las recomendaciones más pertinentes y alcanzables con lo ya construido a la hora de la funcionalidad e interfaz de usuario son las siguientes:

- Sistema de login y signup: Mediante un sistema de autenticación en el frontend, el sistema de Big Data no solo trabajaría con datos del estacionamiento, sino que agregaría a sus modelos información de usuario, como hábitos y preferencias.

- Funcionalidad de recomendación: Con la información de login/signup y una predicción de porcentajes de ocupación, el sistema podría recomendar estacionamientos y áreas de los mismos al usuario mediante notificaciones externas o intra-sistema.

- Funcionalidad de reserva: El usuario también tendría la capacidad de reservar un lugar y el sistema al tener capacidad de predecir en el futuro los porcentajes de ocupación, podría calcular el tarifado de cada área de estacionamiento para ayudar a decidir al usuario que lugar reservar.

- Realizar el dashboard de BI en tiempo real y no en batch.

- Implementar una MobileApp, de manera de brindar al usuario más opciones a la hora de utilizar e interactuar con el sistema.

- Asociación de sistema de pagos: Una mejora a futuro para el sistema, podría ser asociar un sistema de pagos, como Strapi, Paypal o Mercadopago por ejemplo, para facilitar el cobro del usuario y asegurar la reserva de lugares en el estacionamiento.

Capítulo 5

BIBLIOGRAFÍA Y ANEXOS

Bibliografía

Andrew G. Psaltis (2017). “Streaming Data: Understanding the real-time pipeline”. Shelter Island, NY, Manning Publications Co.

Byron Ellis (2014). “Real-Time Analytics”. Indianapolis, Indiana, John Wiley and Sons Inc.

Gowri Sankar Ramachandran, Jeremy Stout, Joyce J. Edson, Bhaskar Krishnamachari (2020). “ParkingJSON: An Open Standard Format for Parking Data in Smart Cities” – Paper.

Faiz Ibrahim Shaikh, Pratik Nirnay Jadhav, Saideep Pradeep Bandarkar, Omkar Pradip Kulkarni, Nikhilkumar B. Shardoor (2016). “Smart Parking System Based on Embedded System and Sensor Network” - Paper

Martin Kleppmann (2016). “Designing Data-Intensive Applications”. California, O’Reilly Media Inc.

Milton Boos Junior (2020-2021). “A Cloud architecture to integrate a Multi-Agent Smart Parking system” – Tesis.

Neha Narkhede (2016). “Kafka: The Definitive Guide”. California, O’Reilly Media Inc.

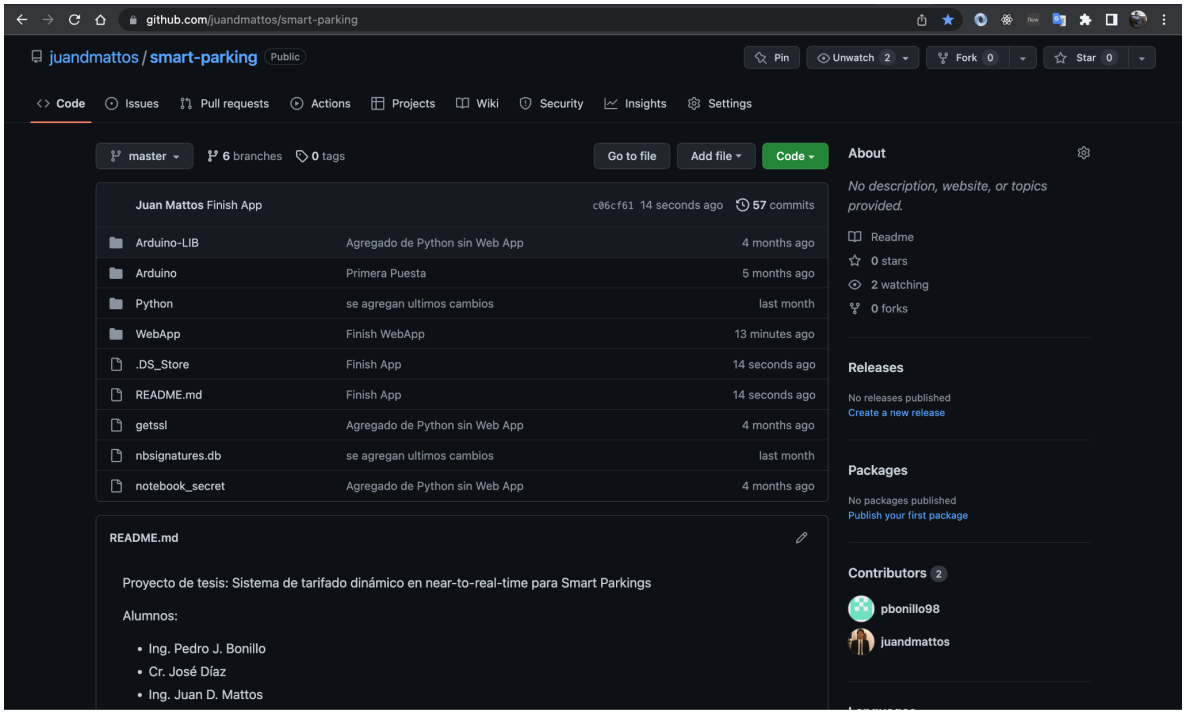
Shama Hoque (2020). “Full-Stack React Projects - Second Edition”. California, O’Reilly Media Inc.

Trista Lin, Hervé Rivano, Frédéric Le Mouël (2017). “A Survey of Smart Parking Solutions” - Paper

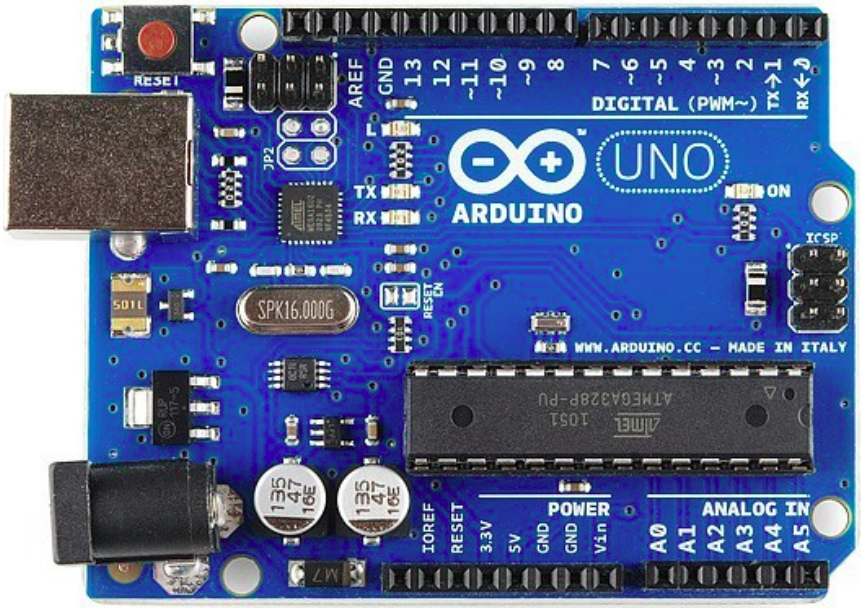
Anexos


```
{
  "parking_id": "2",
  "parking_name": "Shopping Punta Carretas",
  "parking_closed": false,
  "parking_address": "Jose Ellauri 11303, Uruguay",
  "parking_summary": {
    "levels": [
      {
        "areas": [
          {
            "area_id": "1",
            "area_occupation_percentage_target": "10"
          },
          {
            "area_id": "2",
            "area_occupation_percentage_target": "40"
          }
        ],
        "level_id": "1"
      }
    ],
    "coefficients": {
      "0to20Coef": "1.1",
      "20to40Coef": "1.2",
      "40to60Coef": "1.4",
      "60to80Coef": "1.8",
      "80to100Coef": "2"
    },
    "hasDynamicPrice": true
  },
  "parking_latitude": "-34.9240989",
  "parking_longitude": "-56.1588356",
  "parking_timestamp": "06/09/2022|09:53:33",
  "parking_wind_speed": 3.19,
  "parking_description": "Shopping Punta Carretas",
  "parking_holiday_type": "L",
  "parking_holiday_status": false,
  "parking_weather_status": "Clear",
  "parking_holiday_description": null,
  "parking_weather_status_detailed": "clear sky",
  "levels": [
    {
      "level_id": "1",
      "level_name": "Piso 1",
      "level_occupation": "Almost Empty",
      "level_total_spots": "5",
      "level_occupied_spots": "2",
      "level_available_spots": "3",
      "level_occupation_percentage": "40",
      "areas": [
        {
          "area_id": "1",
          "area_name": "A",
          "area_color": "#3244a8",
          "area_summary": {
            "dayFee": "300",
            "hourFee": "80",
            "monthFee": "2000",
            "hasFreeHours": true,
            "freeHourUntil": "2"
          },
          "area_occupation": "Empty",
          "area_description": "blue",
          "area_total_spots": "2",
          "area_average_price": "200",
          "area_occupied_spots": "0",
          "area_available_spots": "2",
          "area_occupation_percentage": "0",
          "slots": [
            {
              "slot_id": "1",
              "slot_type": "cars",
              "slot_price": "200",
              "slot_state": false,
              "slot_description": "1A"
            },
            {
              "slot_id": "2",
              "slot_type": "motorcycles",
              "slot_price": "200",
              "slot_state": false,
              "slot_description": "2A"
            }
          ]
        },
        {
          "area_id": "2",
          "area_name": "B",
          "area_color": "#cf852b",
          "area_summary": {
            "dayFee": "300",
            "hourFee": "80",
            "monthFee": "2000",
            "hasFreeHours": true,
            "freeHourUntil": "2"
          },
          "area_occupation": "Almost Full",
          "area_description": "orange",
          "area_total_spots": "3",
          "area_average_price": "200",
          "area_occupied_spots": "2",
          "area_available_spots": "1",
          "area_occupation_percentage": "66",
          "slots": [
            {
              "slot_id": "1",
              "slot_type": "trucks",
              "slot_price": "200",
              "slot_state": false,
              "slot_description": "1B"
            },
            {
              "slot_id": "2",
              "slot_type": "trucks",
              "slot_price": "200",
              "slot_state": true,
              "slot_description": "2B"
            },
            {
              "slot_id": "3",
              "slot_type": "cars",
              "slot_price": "200",
              "slot_state": true,
              "slot_description": "3B"
            }
          ]
        }
      ]
    }
  ]
}
```

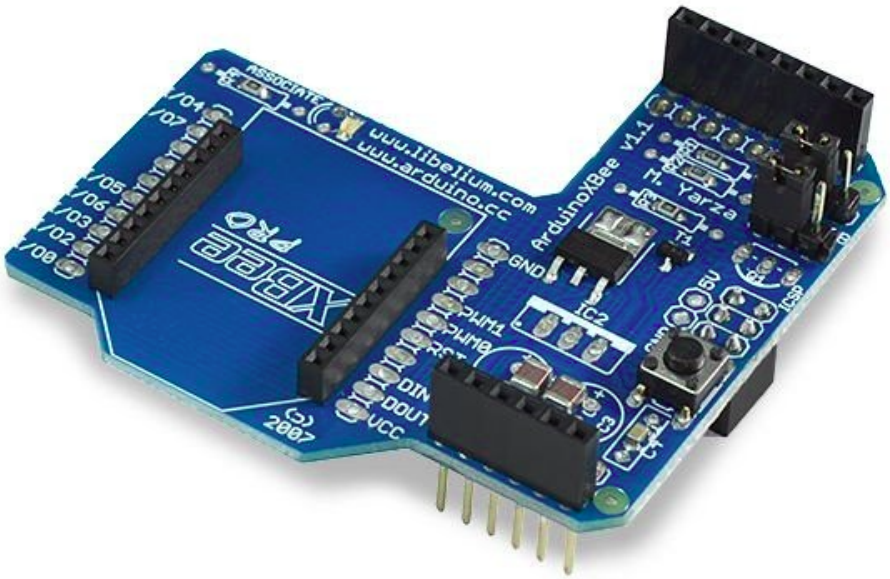

Anexo 1 - Ejemplo de formato JSON de sistema de Big Data a WebApp



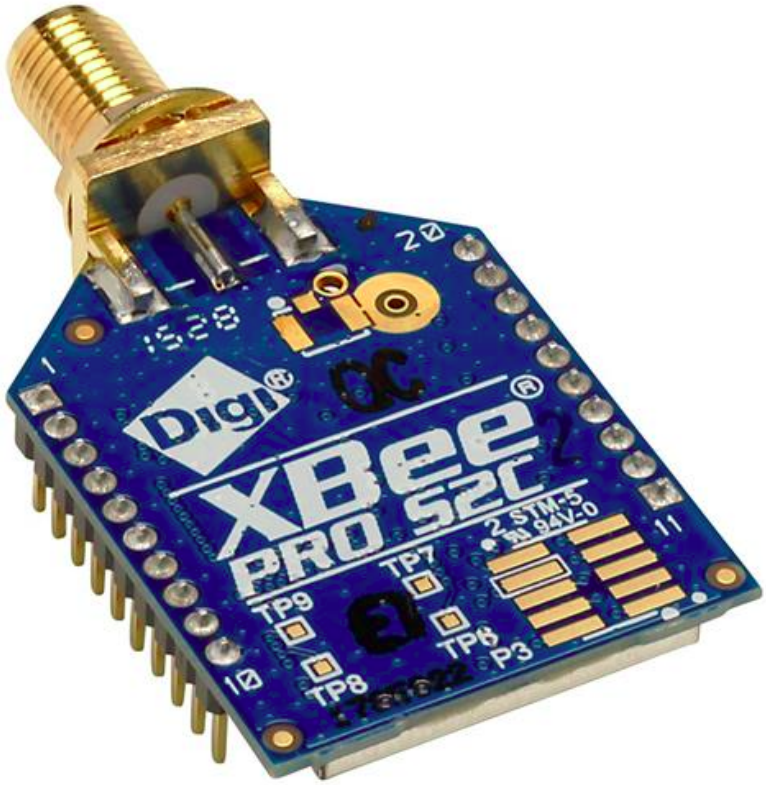
Anexo 2 - Repositorio Github con el proyecto - https://github.com/juandmattos/smart-parking



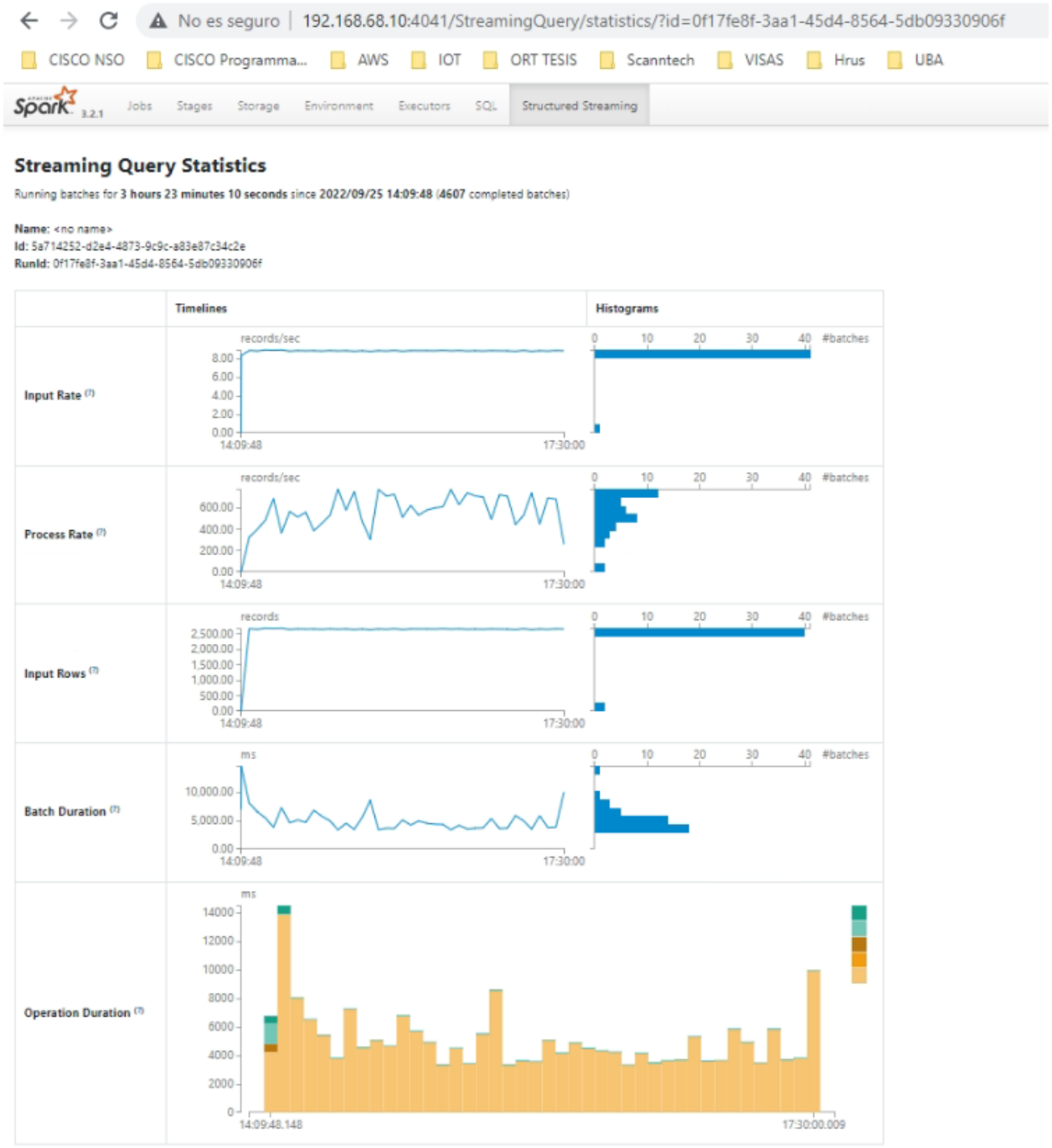
Anexo 3 - Arduino UNO R3 usado en Smart Parking System.



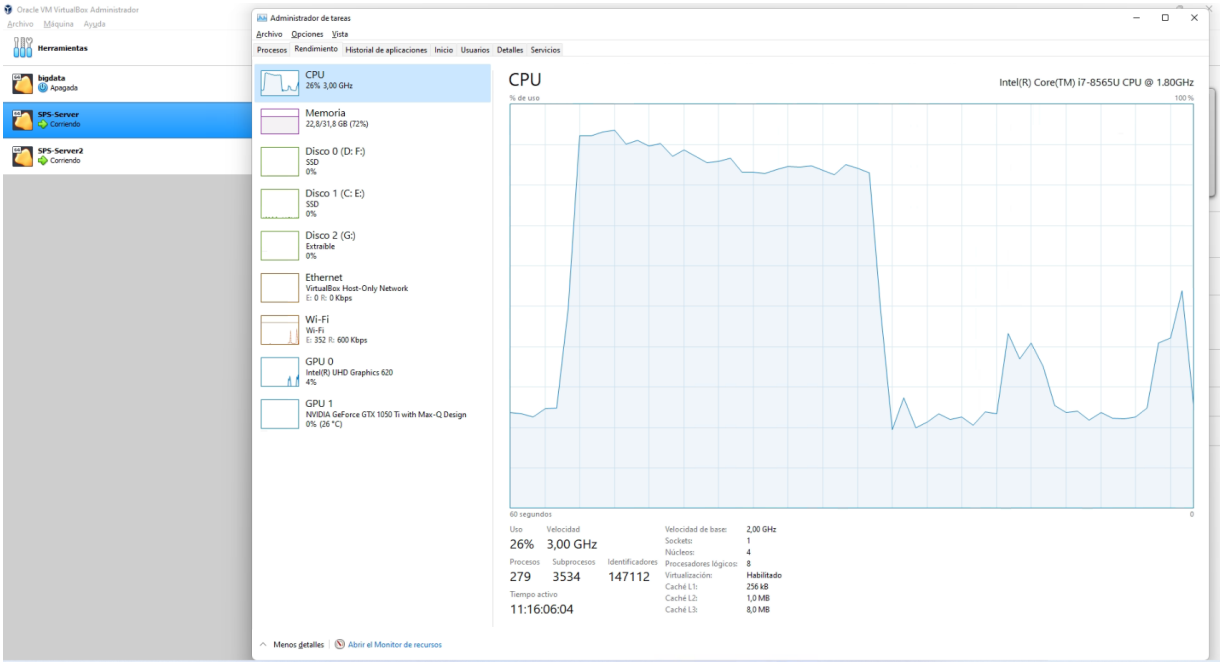
Anexo 4 - XBee Arduino Shield.



Anexo 5 - XBee S2C usado para Smart Parking System.



Anexo 6 - Estadísticas del Streaming estructurado.



Anexo 7 - Estado de los servidores de Smart Parking System.