Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

ANÁLISIS PREDICTIVO DE EVENTOS EN SISTEMAS DE EVACUACIÓN DE EMERGENCIA

TRABAJO FIN DE MÁSTER

José Andrés Muñoz Arcentales

2017

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

ANÁLISIS PREDICTIVO DE EVENTOS EN SISTEMAS DE EVACUACIÓN DE EMERGENCIA

Autor
José Andrés Muñoz Arcentales

Director
Joaquín Salvachúa Rodríguez

Departamento de Ingeniería de Sistemas Telemáticos

2017

Resumen

Lo que se presenta en este proyecto es la implementación de un módulo de análisis predictivo de eventos aplicado a resolver parte de la problemática planteada en los sistemas de evacuación de emergencia. Para ello se hace uso de los distintos valores que nos proveen los sensores utilizados en este tipo de sistemas, siendo estos ingresados como parámetros de entrada para el entrenamiento de de un modelo de aprendizaje automático, que permite predecir la ocurrencia de algún tipo de evento, que pueda llegar a desencadenar una situación de emergencia dentro de la infraestructura monitorizada.

El desarrollo del presente trabajo se divide en varias etapas, en las que se definen cada uno de los elementos que conforman la realización del módulo planteado. La primera etapa esta definida como la etapa de clasificación y adquisición de los datos provenientes de los nodos sensores. Esto se lo realiza haciendo uso de tecnologías como Apache Kafka y Zookeeper . Posteriormente en la siguiente etapa se procesa el flujo de los datos por medio de técnologías de Big Data, utilizando herramientas como: Apache Spark and Mlllib, entre otros. Todos los datos procesados en esta, etapa pasan a ser datos de entrenamiento para un modelo de aprendizaje automático basado en un algoritmo de clasificación de aprendizaje supervisado obteniendo como salida los valor de predicción que de ser necesario, enviarán una trama de alerta para activar el envío de las rutas de evacuación. Finalmente, todo los datos de salida del modelo predictivo de eventos serán almacenados y presentados visualmente por medio de una interfaz web de usuario definida en Apache Zeppelin.

Con todo lo anteriormente descrito, se realizan pruebas para determinar el correcto funcionamiento del módulo planteado y poder de esta forma cumplir con el objetivo de poder manejar un mayor margen de tiempo de evacuación, contribuyendo de esta forma el poder precautelar la vida de las personas con mayor probabilidad de éxito, ademas de contribuir con los servicios de socorro al proveerles una mayor capacidad de acción frente a la ocurrencia de una emergencia.

Abstract

This project shows the implementation of a predictive analysis module applied to the dynamic evacuation systems. This module uses as data input the values gather by the sensors usually deployed in this type of systems, these values are sent to a machine learning model able to to predict the occurrence of an event that could generate an emergency situation inside of the monitored infrastructure

This development is divided in various stages, in where is defined each element which is part of the whole module. The first stage is of acquisition and classification of the data that's coming from the sensors, using technologies as Apache Kafka and Zookeeper. The next stage process the data flow using big data technologies as Apache Spark and SparkSQL. All of the data processed in the previous stage is used for the training of a machine learning model based on a classification algorithm of supervised learning, as a result of that, a data output with the predicted value is obtained and depending of the case, an alarm is generated. Finally, all of the data output of the predictive model is presented in a web interface defined in Apache Zeppelin.

With all of the information described in the previous paragraphs, the corresponding test is developed in order to determine the correct execution of the module development, and accomplish the goal of manage a greater evacuation time, for people in emergency cases. Also provide the helping teams of a greater capacity of intervention in case of emergency

Índice General

Índice de Figuras

List of Figures

1 Introducción

Actualmente, el plan de evacuación de una edificación viene predefinido de forma estática y su funcionalidad se basa en la presencia de una señalización clara y el mantenimiento adecuado de las salidas de emergencia. Sin embargo, la experiencia demuestra que, en una situación de emergencia, la mayoría de las víctimas no son provocadas por la causa directa de la catástrofe (incendios, inundaciones, gases tóxicos), sino por el efecto de una multitud presa del pánico, corriendo hacia puertas de salida a través de rutas mal mantenidos o señalizadas de forma confusa. Este resultado no significa necesariamente que el diseño original del sistema de emergencia fuese negligente, sino que los cambios en las estructuras o simplemente la presencia de escombros pueden hacer a este plan inaplicable y obsoleto.

Un diseño alternativo de plan de evacuación debe basarse en un procedimiento dinámico, capaz de adaptarse a las diferentes situaciones y, por tanto, de "recalcular" vías de escape a través de las lecturas obtenidas a partir de una variedad de sensores (sustancias tóxicas, espacios de inundación, humo y el nivel de la obstrucción o la ocupación de una ruta, los recursos de asistencia disponibles...).

Un sistema de evacuación de emergencia consiste en múltiples elementos como, adquisición de datos, procesamiento de video, detección de patrones, redes heterogeneas, entre otras, que al interactuar entre ellas, proveen información precisa acerca de los eventos que se estén desenvolviendo dentro del edificio, y que puedan generar algún tipo de alerta. [1]

Este proyecto se define como un complemento al modelo de “Adaptive evacuation management system using monitoring techniques”[1], el sistema antes mencionado plantea únicamente la asignación de rutas de emergencia en caso de que alguna emergencia haya sucedido, es decir actúa ́de forma reactiva. Lo que aquí se plantea es que por medio del análisis de datos se pueda llegar a definir un modelo de aprendizaje automático que pueda realizar la predicción de una emergencia, por medio de esto se lograría manejar una evhtana de tienpo suficiente para realizar una evacuación, así como una acción mas rápida de los equipos de socorro.

Para ello se emplean para la adquisición de datos técnicas de Big Data para poder manejar y utilizar la información proveniente de los nodos sensores, además determinar y definir un algoritmo de aprendizaje automático que

permita realizar el análisis predictivo de los datos y poder enviar los parámetros de salida a un sistema de definición de rutas de evacuación. Finalmente se presentará una interfaz gráfica que permita ver de forma visual la ejecución y despliegue del desarrollo planteado.

1.1 Objetivo General

Definir un esquema de aprendizaje automático para la la realización de análisis predictivo de eventos, que permita hacer el procesamiento de datos medidos por sensores y que genere las salidas correspondientes para identificar si se debe generar una alerta, para realizar una evacuación den sistemas dinámicos de evacuación de emergencia.

1.2 Objetivos Específicos

- Desarrollar un programa que permita simular nodos de adqusición de datos por medio de sensores para las etapas de adquisición del dataset y pruebas.
- Definir una arquitectura para el manejo de eventos asíncronos provenientes de nodos sensores
- Estudiar, determinar y probar algortimos de aprendizaje automático para realizar el análisis predictivo de eventos
- Definir una plataforma de despliegue de servicios que permita el manejo de altos volúmenes de datos
- Estudiar, implementar y probar Spark juntos con sus componentes de Streaming, Sql y Mllib para definir un programa que procese los datos en tiempo real y genere una respuesta a los eventos de entrada

2 Estado del Arte

En esta sección se describe el estado del arte de cada una de las tecnologías que se utilizan para el despliegue e implementación del sistema planteado. Básicamente, lo que se muestra es una breve descripción de cada tecnología, sus características más importantes y finalmente la arquitectura que maneja.

2.1 Docker

Docker es un proyecto de código libre que empaqueta, distribuye y ejecuta aplicaciones bajo la funcionalidad de contenedores ligeros. Con esto lo que se logra es proporcionar una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo [2].

Docker es una forma de ejecutar procesos de forma aislada pero también se compone de herramientas para construir imágenes y un repositorio para compartirlas.Al contrario de la virtualización Docker no emula o virtualiza una máquina y su sistema operativo con lo que los procesos son mucho más ligeros y hace que el hardware pueda ser aprovechado más al poder aumentar la densidad de los servicios en una misma máquina. Los contenedores y servicios incluidos en ellos inician muy rápidamente, en pocos segundos. Además, no es necesario el sistema de archivos completo del sistema operativo invitado con lo que docker usa una fracción de espacio de almacenamiento necesario en la virtualización. [3].

Los contenedores de docker son agnosticos al hardware y a la plataforma. En otras palabras, esto permite que su pueda desplegar desde una computadora personal hasta un gran sistema de computacion en la nube. Este tipo de tecnología no requiere el uso de ningun framework o sistema de empaquetamiento , por lo que es una de las grandes ventajas para el despligue y escalado de aplicaciones web , bases de datos, backend services, entre otros [4].

2.1.1 Características

Docker tiene varias características que definen justmente el potencial de este tipo de tecnología, dentro de todas las características podemos nombrar las mas relevantes como:

- **Portabilidad**, puesto que las imagenes, pueden ser desplegadas entre diferentes plataformas, con el único requerimiento de que en el sistema huesped se encuentre disponible el servicio de docker[5].
- **Autosuficiencia**, Los contenedores son autosuficientes (aunque pueden depender de otros contenedores, por ejemplo, un wordpress que necesita una base de datos mysql) no necesitando nada más que la imagen del contenedor para que funcionen los servicios que ofrece[5].
- **Ligereza**, debido a que no hay virtualización aprovechándose mejor el hardware y únicamente necesitando el sistema de archivos mínimo para que funcionen los servicios[3].
- **Seguridad**, pudiendo hacer que los contenedores se comuniquen por un túnel solo disponible para ellos, los contenedores están aislados en el sistema mediante namespaces y control groups[3].

Finalmente se preseta algunas caracteristicas adicionales:

- Aplicaciones libres de las dependencias instaladas en el sistema anfitrión.
- Capacidad para desplegar multitud de contenedores en un mismo equipo físico.
- Puesta en marcha de los servicios en un abrir y cerrar de ojos.
- Contenedores muy livianos que facilitan su almacenaje, transporte y despliegue.
- Capacidad para ejecutar una amplia gama de aplicaciones (prácticamente cualquier cosa que se nos ocurra podrá ejecutarse en un contenedor Docker).
- Compatibilidad Multi-Sistema, podremos desplegar nuestros contenedores en multitud de plataformas.
- La aplicación base de Docker gestionará los recursos existentes para asignarlos responsablemente entre los contenedores desplegados.
- Podremos establecer una base desde la que comenzar nuestros proyectos, lo que nos ahorrará el tiempo de preparar el entorno para cada uno de ellos.
- Podremos compartir nuestros contenedores para aumentar los repositorios de Docker así como beneficiarnos de los que compartan los demás.

2.1.2 Arquitectura

Para lograr identificar las ventajas del uso de contenedores, es necesario hacer una breve descripción de la arquitectura que maneja esta tecnología, ademas de mostrar las principales diferencias entre virtualizacion de servicios y el despligue de servicios en contenedores.

Virtualización. En un esquema de virtualización, por cada aplicación virtualizada se incluye, además de la aplicación por si misma, una serie de librerías y archivos binarios para el uso de la aplicación, esto normalmente genera un consumo adicional de recursos. Pero además incluye el un sistemas operativo completo en donde se aloja la aplicación, generando de esta forma un consumo aun mayor de los recursos del equipo, una descripción gráfica de los descrito en este párrafo se muestra en la Fig 1.

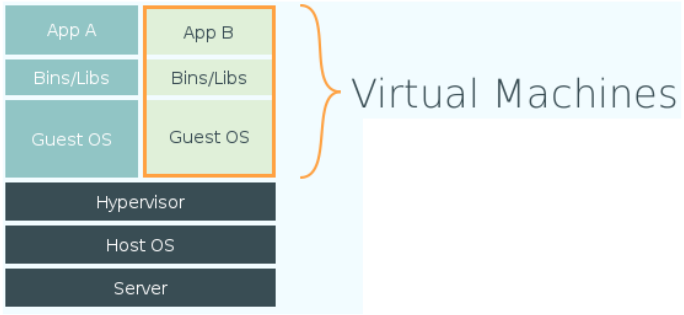

Figure 1: Vitualización

Docker. A diferencia del esquema de virtualización, Docker por medio del uso de Docker Engine container, únicamente contiene la aplicación y sus dependencias. Ejecutándose como un proceso aislado dentro del espacio del usuario del sistema operativo host, por lo que se presenta como un servicio mucho mas portable y eficiente en el uso de los recursos. en la Fig 2 se presenta el esquema de Docker.

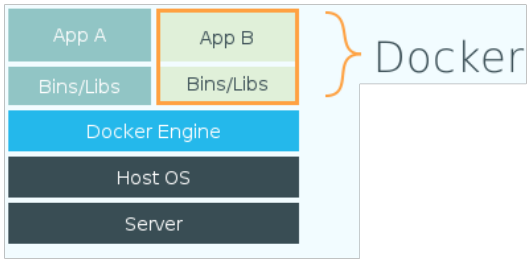

Figure 2: Esquema Docker

Docker proporciona la capacidad de empaquetar y ejecutar una aplicación en un entorno suelto llamado un contenedor. El aislamiento y la seguridad le permiten ejecutar muchos contenedores simultáneamente en un host determinado. Debido a la naturaleza ligera de los contenedores, que se ejecutan sin la carga adicional de un hipervisor, puede ejecutar más contenedores en una combinación de hardware dada que si estuviera utilizando máquinas virtuales.

Docker proporciona herramientas y una plataforma para administrar el ciclo de vida de sus contenedores:

Encapsule sus aplicaciones (y componentes de soporte) en contenedores Docker Distribuya y envíe los contenedores a sus equipos para su posterior desarrollo y pruebas. Implementar esas aplicaciones en su entorno de producción, ya sea en un centro de datos local o en el Cloud.

El componente principal de Docker es el Docker Engine que es una aplicación cliente-servidor con estos componentes principales, tal y como se aprecia el la Fig 3:

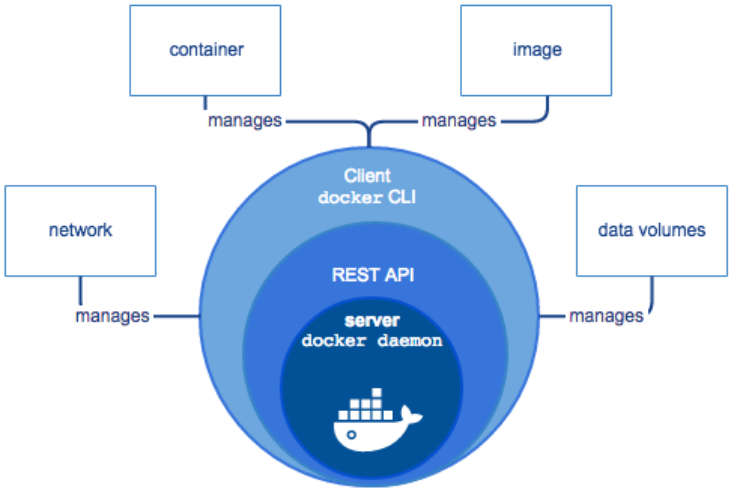

Figure 3: Arquitectura de Docker

Un servidor que es un tipo de programa de larga ejecución llamado proceso de daemon.Una API REST que especifica las interfaces que los programas pueden usar para hablar con el daemon e instruirlo qué hacer. Un cliente de interfaz de línea de comandos (CLI). La CLI utiliza la API Docker REST para controlar o interactuar con el demonio Docker a través de scripts o comandos directos de CLI. Muchas otras aplicaciones de Docker utilizan la API y la CLI

subyacentes. El daemon crea y administra objetos de Docker, como imágenes, contenedores, redes y volúmenes de datos [6].

2.1.3 Componentes

En la sección anterior se describió la arquitecctura de Docker y se mostro como docker engine es el que gestiona la creación de contenedores. Con lo anteriormente descrito se logró comprender que Docker utiliza una arquitectura cliente - servidor, el “Docker daemon” es el encargado de realizar el trabajo "construir, ejecutar y distribuir" los contenedores, el cliente permite interactuar con el servidor “Docker daemon”, para lograr ese objetivo, Docker hace uso de tres componentes :

- **Docker images.** Las imágenes docker son una especie de plantillas de solo lectura, las cuales contiene la información necesaria para ejecutar un contendedor. Una imagen contiene el sistema operativo, y servicios adicionales para ejecutar algún tipo de servicio por ejemplo un servidor web apache con módulos de php, y una base de datos. Cada imagen inicia desde una imagen base, por ejemplo "Centos image", "Red Debian image". A esta imagen base se agrega otros componentes o servicios porejemplo una base de datos NoSQL "neo4j", node.js, etc.
- **Docker registries.** Los registros docker son repositorios que contienen las imágenes de los cuales se pueden descargar, agregar o actualizar. Estos permiten compartir las imágenes y ademas se pueden tener registros públicos o privados.
- **Docker containers.** Un contenedor es creado a partir de una imagen, estos contenedores pueden ser ejecutados, detenidos, iniciados o eliminados. Además un contenedor contiene todo lo necesario para que una aplicación pueda ejecutarse “Sistema operativo, aplicaciones, archivos, metadatos, procesos, etc.”

Según la web de oficial de documentación de Docker , existen una serie de componentes que permiten el manejo y despliegue de contenedores con diferentes tipos de servicios[7], entre ellos tenemos :

- **Docker Hub** Docker Hub es un servicio de registro basado en la nube que permite enlazar a repositorios de código, construir imágenes y probarlas, en el se almacenan manualmente imágenes y enlaces a Docker

Cloud para que se puedan desplegar imágenes en los diferentes hosts. Proporciona un recurso centralizado para el descubrimiento de imágenes de contenedores, la distribución y la gestión del cambio, la colaboración de usuarios y equipos y la automatización del flujo de trabajo en toda la cadena de desarrollo.

- **Docker Trusted Registry** Docker Trusted Registry (DTR) provee una funcionalidad similar a la de Docker Hub pero DTR es la solución de almacenamiento de imágenes de calidad empresarial de Docker. Normalmente este componente se instala detrás del firewall empresarial para el almacenamiento y administración con seguridad de las imágenes de Docker que usan en las aplicaciones dentro de la empresa.

- **Docker Compose** Compose es una herramienta para definir y ejecutar aplicaciones Docker de contenedores múltiples. Docker Compose, utiliza un archivo de Compose para configurar los servicios de la aplicación a ser ejecutada. Luego, con un solo comando, crea e inicia todos los servicios de su configuración.

- **Docker Universal Control Plane** Docker Universal Control Plane (UCP) es la solución de gestión de clúster de grado empresarial de Docker. Puede ser instalado localmente o en una nube privada virtual y brinda ayuda con la administración sel clúster y las aplicaciones de Docker de forma centralizada.

- **Docker Cloud** Docker Cloud ofrece un servicio de registro con instalaciones de compilación y pruebas para imágenes de aplicación Dockerized; Además de herramientas que ayudan a la configuración y administración de la infraestructura del host; con esto lo que se logra es manejar todas las características del ciclo de vida de la aplicación para automatizar la implementación (y redistribución) de los servicios creados a partir de imágenes.

En resumen, Docker es una tecnología que nos permite empaquetar desplegar y ejecutar servicios dentro de contenedores, presentando multiples ventajas frente a los sistemas de virtualización. Docker permite hacer un mejor uso y distribución de los recursos y esta enfocado desde ejecuciones unitarias hasta grandes despligues, sus componentes principales son los Docker Images, Docker registry y Docker container y el motor de gestion de los contenedores es Docker Engine.

2.2 Apache Spark

Apache Spark es un sistema de computación de clúster rápido y de propósito general. Proporciona APIs de alto nivel en Java, Scala, Python y R, y un motor optimizado que soporta grafos de ejecución general. También soporta un amplio conjunto de herramientas de alto nivel, como Spark SQL para SQL y procesamiento de datos estructurados, MLlib para el aprendizaje automático, GraphX para procesamiento de grafos y Spark Streaming [8].

Apache Spark proporciona a los programadores una interfaz de programación de aplicaciones centrada en una estructura de datos denominada conjunto de datos distribuidos resilientes (RDD), un multiset de sólo lectura de elementos de datos distribuidos en un grupo de máquinas, que se mantiene de forma tolerante a fallos [9].

Se desarrolló en respuesta a las limitaciones en el paradigma de computación de cluster MapReduce, que obliga a una estructura de flujo de datos lineal particular en programas distribuidos: MapReduce programas de lectura de datos de entrada de disco, mapa de una función a través de los datos, reducir los resultados del mapa, Resultados en disco. Los RDDs de Spark funcionan como un conjunto de trabajo para programas distribuidos que ofrece una forma deliberadamente restringida de memoria compartida distribuida [10].

2.2.1 Características

Spark posee múltiples configuraciones, y dependiendo de la aplicación y los componentes que se utilicen para ella, pero se pueden destacar tres características generales [11]:

- **Rapidez** Spark ayuda a ejecutar una aplicación en el clúster Hadoop, hasta 100 veces más rápido en la memoria, y 10 veces más rápido cuando se ejecuta en el disco. Esto es posible reduciendo el número de operaciones de lectura-escritura en el disco. Almacena los datos de procesamiento intermedios en la memoria.
- **Multilenguaje** Spark proporciona APIs integradas en Java, Scala o Python. Por lo tanto, puede escribir aplicaciones en diferentes idiomas. Spark cuenta con 80 operadores de alto nivel para la consulta interactiva.

- **Analítica Avanzada** Spark no sólo admite 'Map' y 'reduce'. También es compatible con consultas SQL, datos de flujo continuo, aprendizaje automático (ML) y algoritmos de grafos.

2.2.2 Arquitectura

La arquitectura de Apache Spark requiere un administrador de clúster y un sistema de almacenamiento distribuido. Para la administración de clústeres, Spark soporta (cluster Spark nativo), Hadoop YARN o Apache Mesos. Para el almacenamiento distribuido, Spark puede interactuar con una amplia variedad, incluyendo Hadoop Distributed File System (HDFS), Sistema de archivos MapR (MapR-FS), Cassandra, OpenStack Swift, Amazon S3, Kudu o Se puede implementar una solución personalizada. Spark también admite un modo local pseudo-distribuido, usualmente utilizado sólo para propósitos de desarrollo o prueba, donde el almacenamiento distribuido no es necesario y se puede usar el sistema de archivos local; En tal escenario, Spark se ejecuta en una sola máquina con un ejecutor por núcleo de CPU [12]. Un esquema típico de la estructura de cluster que maneja Spark se puede apreciar en la Fig 4

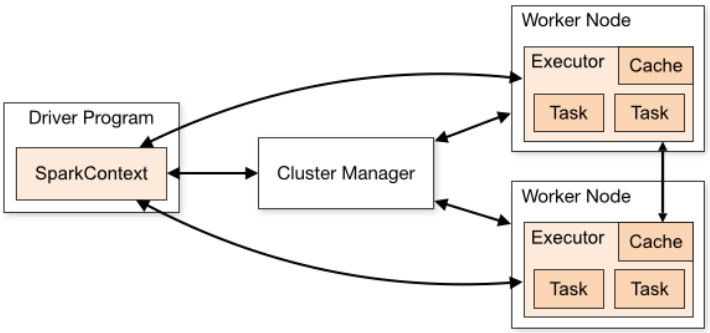

Figure 4: Estructura de Cluster Spark

Adicionalmente Spark presenta tres tipos diferentes de configuración, tal y como se aprecia en la Fig 5 [11]

- **Standalone** El despliegue independiente significa que Spark ocupa el lugar en la parte superior de HDFS (Hadoop Distributed File System) y el espacio se asigna explícitamente para HDFS. Aquí, Spark y MapReduce se ejecutarán lado a lado para cubrir todos los trabajos de encendido en el clúster.

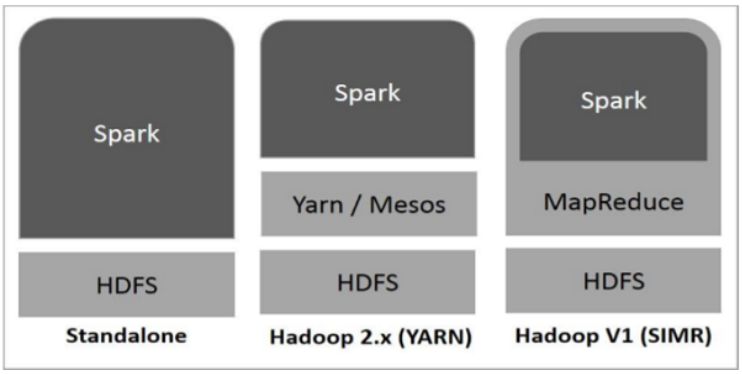

Figure 5: Tipos de Despliegue

- **Hadoop** El despliegue de hilo significa, simplemente, que la chispa se ejecuta en Hilados sin necesidad de pre-instalación o acceso root. Ayuda a integrar Spark en el ecosistema Hadoop o pila Hadoop. Permite que otros componentes se ejecuten en la parte superior de la pila.
- **Spark in MapReduce (SIMR)** Spark en MapReduce se utiliza para iniciar el trabajo de chispa, además de la implementación independiente. Con SIMR, el usuario puede iniciar Spark y utiliza su shell sin ningún acceso administrativo.

2.2.3 Componentes

Spark tiene como componente principal a Spark Core, pero se complementa con un conjunto de potentes bibliotecas de alto nivel que se pueden utilizar sin problemas en la misma aplicación. Estas bibliotecas incluyen actualmente SparkSQL, Spark Streaming, MLlib (para el aprendizaje automático) y GraphX, en la Fig 6 s emuestra la arquitectura de Spark con sus componentes [13].

- **Spark Core** es el motor base para procesamiento de datos paralelo y distribuido a gran escala. Es responsable de la gestión de memoria y recuperación de fallos, programación, distribución y supervisión de trabajos en un clúster e interactuando con sistemas de almacenamiento

  Spark introduce el concepto de un Conjunto de Datos Distribuidos Resilientes (RDD), una colección distribuida de objetos que pueden ser operados en paralelo, inmutable y tolerante a fallos. Un RDD puede contener cualquier tipo de objeto y se crea cargando un conjunto de

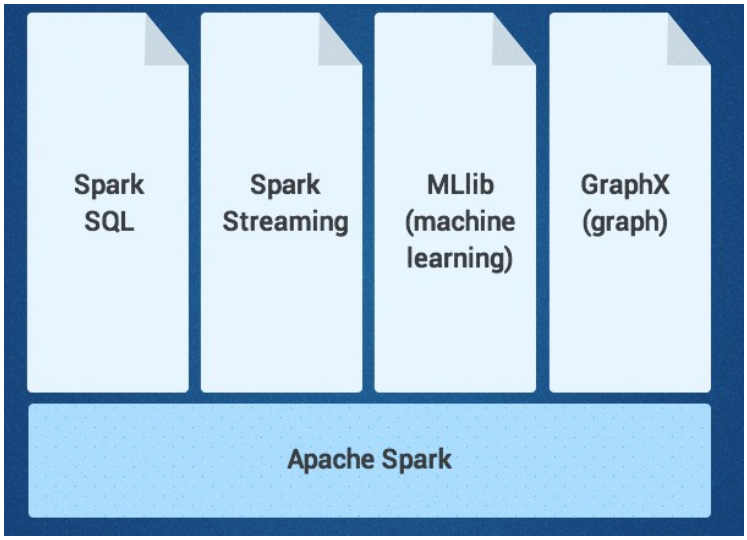

Figure 6: Arquitectura Spark

datos externo o distribuyendo una colección desde el programa de controladores.

Los RDD admiten dos tipos de operaciones:

Las transformaciones son operaciones (como mapa, filtro, unión, unión, etc.) que se realizan en un RDD y que producen un nuevo RDD que contiene el resultado.

Las acciones son operaciones (como reducir, contar, primero, etc.) que devuelven un valor después de ejecutar un cálculo en un RDD.

Las transformaciones en Spark son "perezosas", lo que significa que no calculan sus resultados de inmediato. En su lugar, simplemente "recuerdan" la operación a realizar y el conjunto de datos (por ejemplo, el archivo) al que se va a realizar la operación. Las transformaciones sólo se calculan cuando se llama a una acción y el resultado se devuelve al programa de controlador. Este diseño permite que Spark funcione de manera más eficiente. Por ejemplo, si un archivo grande se transformó de varias maneras y se pasó a la primera acción, Spark sólo procesaría y devolverá el resultado de la primera línea, en lugar de realizar el trabajo para todo el archivo.

- **SparkSQL**, es un componente Spark que admite la consulta de datos ya sea a través de SQL o mediante el lenguaje de consulta de colmena. Se originó como el puerto Apache Hive para correr en la parte superior de Spark (en lugar de MapReduce) y ahora está integrado con la pila Spark. Además de proporcionar soporte para diversas fuentes de datos,

permite integrar consultas SQL con transformaciones de código, lo que resulta en una herramienta muy potente.

- **Spark Streaming**, Soporta el procesamiento en tiempo real de los datos de transmisión, como archivos de registro del servidor web de producción (por ejemplo, Apache Flume y HDFS / S3), medios sociales como Twitter y varias colas de mensajes como Kafka. Bajo el capó, Spark Streaming recibe los flujos de datos de entrada y divide los datos en lotes. A continuación, son procesados por el motor Spark y generan el flujo final de resultados en lotes, como se muestra a continuación en la Fig 7. La API de Spark Streaming coincide con la de Spark Core, facilitando así que los programadores trabajen en los mundos de los lotes y de los datos de transmisión.

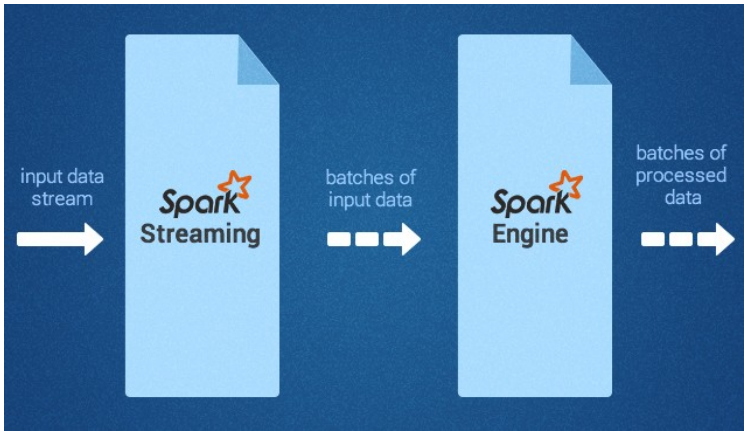

Figure 7: Arquitectura Spark

- **MLlib**, es una biblioteca de aprendizaje automático que proporciona varios algoritmos diseñados para escalar en un clúster para la clasificación, regresión, agrupación, filtrado colaborativo, etc. (consulte el artículo de Toptal sobre aprendizaje de máquinas para obtener más información sobre este tema). Algunos de estos algoritmos también funcionan con datos de flujo continuo, como la regresión lineal usando mínimos cuadrados ordinarios o agrupación en k-means (y más en el camino). Apache Mahout (una biblioteca de aprendizaje de máquinas para Hadoop) ya se ha alejado de MapReduce y unido fuerzas en Spark MLlib.

- **GraphX**, Es una biblioteca para manipular gráficas y realizar operaciones paralelas a gráficas. Proporciona una herramienta uniforme para ETL, análisis exploratorio y cálculos de gráficos iterativos. Aparte de las

operaciones incorporadas para la manipulación del gráfico, proporciona una biblioteca de algoritmos comunes del gráfico tales como PageRank.

En resumen, me diante el uso de Spark se simplifica la tarea de procesamiento intensivo de grandes volúmenes de datos en tiempo real o archivados, estructurados y no estructurados. En el caso específico de este proyecto permite mediante uso de Spark Streaming y Mlib permite la adquisición y procesamiento de datos en tiempo real, que permite manejar los eventos en sistemas de emergencia.

2.3 Apache Kafka

Apache Kafka es un proyecto de intermediación de mensajes de código abierto desarrollado por la Apache Software Foundation escrito en Scala [14]. Kafka es un servicio de registro distribuido, particionado, replicado desarrollado por LinkedIn y de código abierto en 2011. Básicamente es una cola de mensajes publish / subscriber ampliamente escalable diseñada como un registro de transacciones distribuidas.El proyecto tiene como objetivo proporcionar una plataforma unificada, de alto rendimiento y de baja latencia para la manipulación en tiempo real de fuentes de datos. Puede verse como una cola de mensajes, bajo el patrón publicación-suscripción, masivamente escalable concebida como un registro de transacciones distribuidas.[15]

2.3.1 Arquitectura

La arquitectura de kafka tiene varios componentes, que permiten manejar de forma eficaz los datos en tiempo real, estos componntes son: Producer, Consumer, Broker y Zookeeper, en la Fig 8 se puede apreciar el esquema general de la arquitectura de kafka, y cual es la interacción entre cada uno de los elementos que conforman la arquitectura.

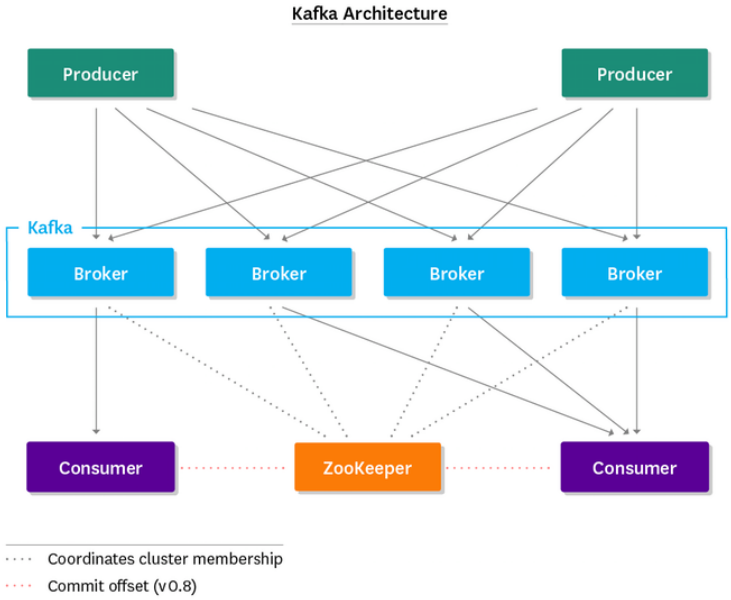

Figure 8: Arquitectura Kakfa

Los producers publican mensajes a los topics, y los consumers (ahem) con-

sumen mensajes de los topics. Los producers y consumers operan de manera push / pull, con los producers empujando mensajes a los Brokers (Kafka), y los consumers encuestando a los brokers para nuevos mensajes.

Los Brokers son nodos de Kafka y actúan como intermediarios, almacenando los mensajes publicados para que los consumers hagan uso de los datos publicados a su propio ritmo. Esto significa que los brokers de Kafka no rastrean el consumo, dejando la eliminación de mensajes a una política de retención configurable.

Los mensajes consisten en una carga útil de bytes sin procesar, con el topic y la información de la partición codificados. Kafka agrupa los mensajes por topic y los consumers se suscriben a los temas que necesitan. Los mensajes en Kafka son ordenados por timestamp y son inmutables, con read como la única operación permitida.

Los topics se dividen en particiones y las particiones se asignan a los intermediarios. Por lo tanto, los temas imponen un fragmento de los datos en el nivel de intermediario. Cuanto mayor sea el número de particiones, un topic específico puede soportar mas consumers simultáneos.

Al configurar Kafka por primera vez, se debe tener cuidado de asignar un número suficiente de particiones por topic, y dividir las divisiones entre sus brokers. Con la replicación habilitada, cada partición se replica a través de varios intermediarios, con el número de intermediarios determinado por el factor de replicación configurado. Aunque pueden existir numerosas réplicas, Kafka sólo iniciará la escritura en el líder de una partición, elegida aleatoriamente del grupo de réplicas sincronizadas. Además, los consumers sólo leerán desde un líder de partición. Por lo tanto, las réplicas seguidoras sirven como fallback (siempre y cuando permanezcan sincronizadas) para mantener la alta disponibilidad en caso de fallo del corredor [15].

Por último, ningún despliegue de Kafka está completo sin ZooKeeper. ZooKeeper es el pegamento que lo sostiene todo junto, y es responsable de [15]:

La elección de un controlador (broker de Kafka que gestiona los líderes de partición) Registro de pertenencia a clúster Configuración del tema Cuotas Afiliación al grupo de consumidores

2.3.2 Componentes

En subsección anterior se mostraron los elementos que conforman la arquitectura de Kakfa, dando paso a que en este apartado se mencione brevemente los componentes de la que complementa la arquitectura mostrada y que permiten gestionar a nivel de configuración las diferentes características que nos provee este tipo de tecnología. En la Fig 9 se muestra los componente de kakfa y posteriormente se provee una breve descripción de a que corresponde cada uno de eso componentes [14].

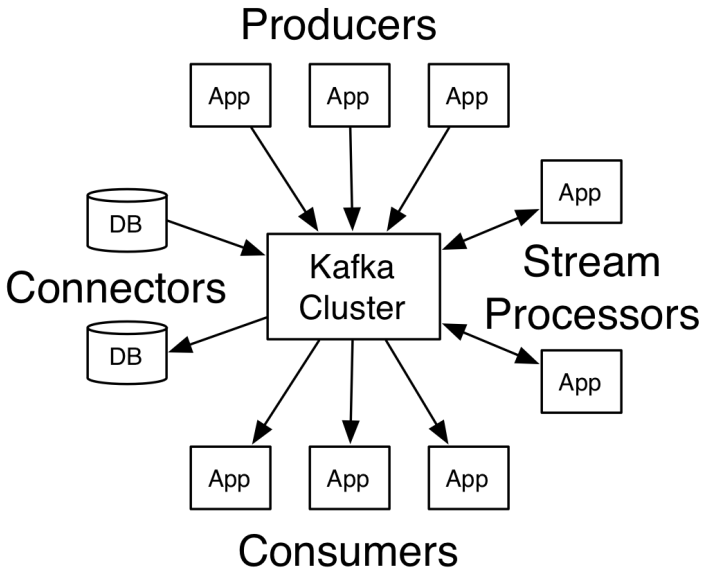

Figure 9: Componentes Kakfa

- El API de Producer permite que una aplicación publique un flujo de registros a uno o más temas de Kafka.
- El API de consumidor permite que una aplicación suscriba uno o más temas y procese el flujo de registros producido a ellos.
- El API de Streams permite que una aplicación actúe como un procesador de flujo, consumiendo una secuencia de entrada de uno o más temas y produciendo una secuencia de salida a uno o más temas de salida, transformando efectivamente los flujos de entrada a los flujos de salida.
- La API de conector permite crear y ejecutar productores reutilizables o consumidores que conectan temas de Kafka a aplicaciones o sistemas de datos existentes. Por ejemplo, un conector a una base de datos relacional puede capturar cada cambio en una tabla.

En resumen, Kafka nos permite manejar de forma ágil rápida y confiable la adquisición de datos o mensajes distribuidos, haciendo uso de el modelo de publish suscriber y utilizando uno o mas Broker para el manejo de los tópico. Finalmente el componente que mantiene interoperando todo junto es zookeeper quien se encarga de tareas tan complejas como elección del líder entre otras. Este tipo de tecnología se utiliza dentro de este trabajo para poder realizar la adquisición de los mensajes enviados por la red de sensores inalámbricos que se encuentra distribuida dentro del edificio monitorizado y se mostrará la aplicación del mismo en subsecuentes secciones.

2.4 Aprendizaje Automático

El aprendizaje automático o Machine Learning es el subcampo de la informática que permite a las máquinas la capacidad de aprender sin ser explícitamente programadas [16]. El aprendizaje de la máquina explora el estudio y la construcción de algoritmos que pueden aprender y hacer predicciones sobre los datos [17], dichos algoritmos superan las instrucciones estrictamente estáticas del programa mediante la fabricación de predicciones o decisiones impulsadas por datos, [18] a través de la construcción de un modelo a partir de los datos de la muestra. El aprendizaje automático se emplea en una serie de tareas informáticas en las que el diseño y la programación de algoritmos explícitos con un buen rendimiento es difícil o inviable.

Dentro del campo del análisis de datos, el aprendizaje automático es un método utilizado para diseñar modelos y algoritmos complejos orientados a realizar predicciones, ampliamente conocido como análisis predictivo. Estos modelos analíticos permiten a los investigadores, científicos de datos, ingenieros y analistas "producir decisiones y resultados fiables y repetibles" y descubrir "conocimientos ocultos" a través del aprendizaje de las relaciones históricas y las tendencias de los datos [19].

El aprendizaje automático maneja tres tipos principales de aprendizaje, supervisado, no supervisado y por refuerzo, cada uno de estos tipos se utilizan dependiendo de las características del data set que va a ser usado para crear el modelo, y basando en sus características se enmarcaría dentro de uno de los tres grupos antes mencionados. Cabe mencionar que dependiendo del tipo de aprendizaje que se vaya a realizar , se pueden utilizar una serie de algoritmos para determinar cual es el que genera un mejor modelo que se represente o procese los datos de manera precisa.

2.4.1 Aprendizaje Supervisado

En el aprendizaje supervisado, los algoritmos son entrenados usando ejemplos etiquetados, es decir el data set que se utiliza para el entrenamiento y prueba tiene clara mente identificadas las entradas y sus salidas deseadas. Por ejemplo, una pieza de equipo podría tener puntos de datos etiquetados como "F" (fallado) o "R" (se ejecuta). El algoritmo de aprendizaje recibe un conjunto de entradas junto con las correspondientes salidas correctas y el algoritmo aprende comparando su salida real con salidas correctas para encontrar er-

rores. Posteriormente, dependiendo de los errores encontrados se modifica el modelo para ajustarse mas a los datos de entrenamiento.Por medio de métodos como la clasificación, la regresión, la predicción y el aumento de gradiente, el aprendizaje supervisado utiliza patrones para predecir los valores de la etiqueta en los datos adicionales no etiquetados. El aprendizaje supervisado es comúnmente usado en aplicaciones donde los datos históricos predicen eventos futuros. Por ejemplo, puede prever cuándo es probable que las transacciones con tarjetas de crédito sean fraudulentas o que el cliente de seguros pueda presentar una reclamación [19].

Existe una variedad de algoritmos que típicamente se emplean en este tipo de arendizaje, pero cada uno de ellos presenta diferentes tipos de ventajas dependiendo de la aplicación, entre los mas comunes tenemos:

- k-NN (k Nearest Neighbors)
- Árboles de clasificación
- ID3/C4.5
- CART
- Clasificadores
- SGD y redes neuronales
- SVM

Es necesario mencionar que en los diferentes tipos de aprendizaje se debe tener especial consideración con identificar que el modelo no se sobre ajuste a los datos, puesto que de esta forma se daría paso a la creación de un modelo distorsionado y se perdería precisión en el mismo. Para poder identificar y corregir este tipo de problemas existen diferentes tipo de tecnicas y consideracciones, por ejemplo La evaluación no debe hacerse sobre los ejemplares que han permitido obtener la función de clasificación.

2.4.2 Aprendizaje no Supervisado

Este tipo de aprendizaje se utiliza contra datos que no tienen etiquetas históricas. El sistema no recibe la "respuesta correcta". El algoritmo debe averiguar qué se está mostrando. El objetivo es explorar los datos y encontrar alguna

estructura dentro. El aprendizaje no supervisado funciona bien en los datos transaccionales. Por ejemplo, puede identificar segmentos de clientes con atributos similares que pueden ser tratados de manera similar en campañas de marketing. O puede encontrar los principales atributos que separan segmentos de clientes entre sí. Las técnicas populares incluyen mapas de auto-organización, mapeo de vecinos más cercanos, agrupación de k-medios y descomposición de valores singulares. Estos algoritmos también se usan para segmentar temas de texto, recomendar ítems e identificar valores atípicos de datos [19]. Para poder resolver los distintos problemas que aborda el aprendizaje supervisado se pueden hacer uso de los siguiente algoritmos para su caso correspondiente:

- Agrupamiento: k-means
- Reglas de asociación: a-priori
- Extracción de características: PCA
- Máquina de Boltzmann restringida

2.4.3 Aprendizaje por refuerzo

Este tipo de aprendizaje tiene un enfoque mas centrado en la robótica, el juego y la navegación. Con el aprendizaje de refuerzo, el algoritmo descubre a través de ensayo y error cuáles acciones producen las mayores recompensas. Este tipo de aprendizaje tiene tres componentes principales: el agente (el aprendiz o el que toma decisiones), el entorno (todo lo que el agente interactúa) y las acciones (lo que el agente puede hacer). El objetivo es que el agente elija acciones que maximicen la recompensa esperada durante un período de tiempo determinado. El agente alcanzará la meta mucho más rápido siguiendo una buena política. Así que la meta en el aprendizaje de refuerzo es aprender la mejor política [19].

3 Arquitectura

Es esta sección se presenta la arquitectura utilizada para el desarrollo de este proyecto, presentando así cada uno de los componentes involucrados en el diseño de la misma y el por qué de si inclusión en este desarrollo. Esta arquitectura usa entro otras, las tecnologías descritas anteriormente en la sección del estado del arte y describe como la integración de ellas, permite el despliegue el escenario para la ejecución y pruebas del sistema.

3.1 Componentes

El sistema aquí planteado se compone de tres componentes principales, el primero denominado Adquisición y Clasificación que es el que permite recibir y clasificar los datos provenientes de los nodos de la red de sensores inalámbrica. El segundo componente se lo conoce como Procesamiento, y es el que se encarga de recibir la información proveniente de Adquisición y Almacenamiento y hacer las predicciones para determinar si debe generar o no una alerta y enviar este estado al generador de rutas. Finalmente el último componente es el de Almacenamiento y Presentación que es el encargado de almacenar la información generada por el componente de Procesamiento y presentar de forma visual los eventos que ocurren dentro del edificio y además, las rutas generadas a partir de dichos eventos. En la Fig 10 se puede apreciar la vista general de la arquitectura que compone el sistema presentando, junto con cada uno de los componentes involucrados y ciertos actores que están presentes en cada uno de ellos, un análisis mas detallado de cada uno de los componentes de la arquitectura se presenta en las siguientes subsecciones.

Asimismo, en dicha figura se presentan cada una de las tecnologías utilizadas por cada componente para poder cumplir con su labor principal. En el caso del primer componente, se asocian a los datos recibidos a partir de la red de sensores inalámbricos, las tecnologías de Zookeeper y kafka para el manejo de los flujos de información y todo esto se ejecuta bajo una infraestructura de contenedores Docker. Por otra parte el componente de Procesamiento tiene como actor principal el uso de Apache spark, especificamente el componente de MLIB que es el que nos provee el API para el manejo de los algoritmos de aprendizaje automático para poder realizar las predicciones basándonos el los datos que se le proveen al modelo, este servicio se ejecuta bajo contenedores Docker. Para finalizar, el último componente hace uso de MongoDb como mo-

tor de base de datos para el almacenamiento de los registros, alertas y rutas generadas y por medio de esa información Apache zepelin presenta al usuario de forma gráfica los eventos generados, nuevamente, todo este despliegue se realiza haciendo uso de contenedores para cada uno de los servicios. Como base de ejecución de los contenedores y servicios por los que esta compuesto el sistema, se usa un servidor linux dentro de una nube de Microsoft Azure.

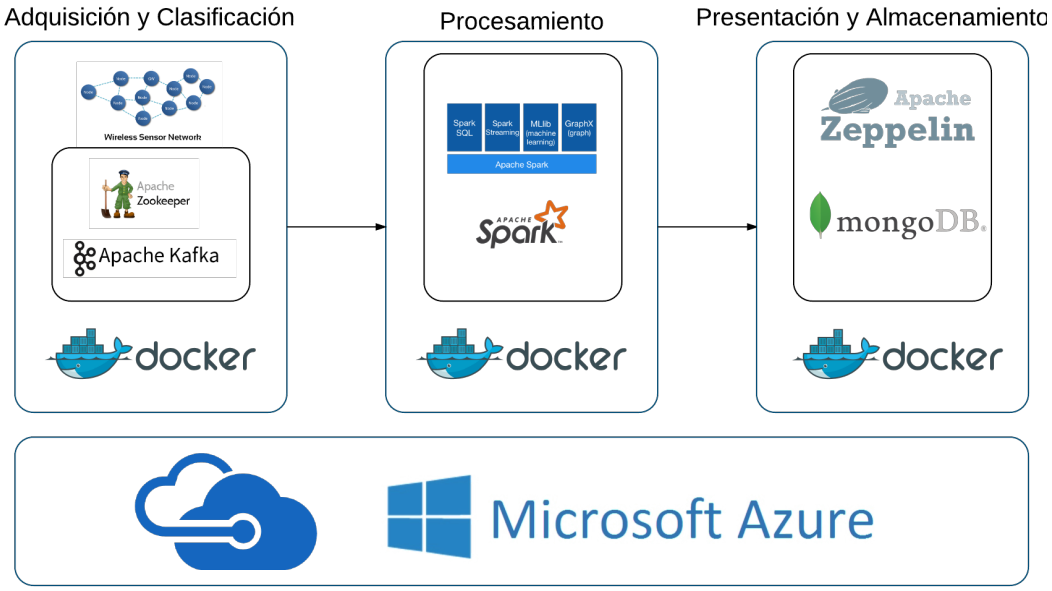

Figure 10: Arquitectura General del Sistema

3.2 Adquisición y Clasificación

Como se menciono anteriormente en este componente se reciben y clasifican los datos provenientes de las redes de sensores inlámbricos, debido a que la información se se genera a partir de los sensores es asíncrona esto hace necesario el uso de tecnologías que nos permitan poder manejar este tipo de eventos. Para ello se usa exactamente el mismo esquema presentado en A distributed system model for managing data ingestion in a wireless sensor network [20], en donde se plante justamente un sistema que permite la adquisición de datos haciendo uso de tópicos definidos en zookeeper, y kafka para el manejo de los eventos. Además se definen los productores y consumidores necesarios para manejar los datos provenientes de los sensores.

Partiendo del estudio realizado, en la Fig 11 se puede apreciar como está compuesto el subsistema de adquisición y clasificación, en nuestro caso es-

pecifico nos centraremos en las secciones a,b y c descritas en dicha figura, la seccion a muestra los nodes de la red de sensores y como estos envián dicha información a la sección b que es donde se encuentran los nodos recolectores y en donde se definen los productores que luego van a interactuar con la sección c que es donde se encuentra definidos los brokers de kafka.

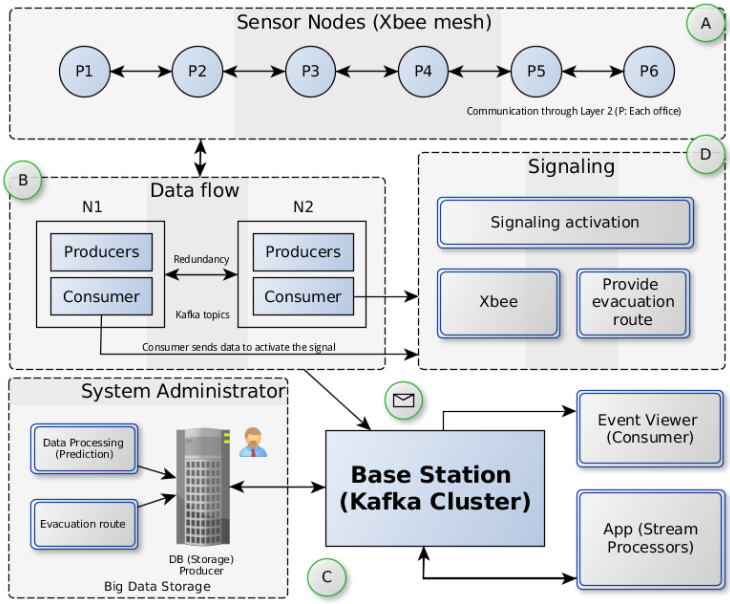

Figure 11: Esquema de componente de Adquisición [20]

En la Fig 12, se muestran cada uno de los productores y consumidores definidos en esta etapa, quedando definidos de la siguiente manera:

- Data Producers: Responsable de la información de datos de los nodos WSN y la transmisión a los productores del cluster kafka.
- Data Base Producer: Este productor es el encargado de transmitir la información de los productores de datos a la base de datos, porque las aplicaciones (procesadores de flujo) utilizarán información de los productores de datos y queremos evitar la sobrecarga.
- Alarm Producer:Cuando se obtienen datos de los sensores, el nodo los procesa para determinar si hay algún inconveniente con los umbrales que se han configurado para cada métrica. Si hubo alguna anomalía, se envía un paquete de "Alarma"
- Activating Signaling Consumer:Este consumidor se encargará de procesar la información enviada por el servidor para la activación de la señalética

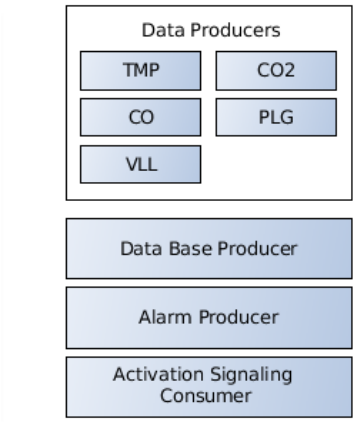

Figure 12: Productores y Consumidores definidos

Con todo lo anteriormente mencionado el subsistema visto de forma detallada se expande a lo presentado en la Fig 13, en donde se muestra como la información obtenida viaja por medio de un proceso de forwarding hacia los broker de kafka y este se encarga de manejar y balancear la carga generada por la ingesta de estos datos para finalmente proveerle al sistema de procesamiento el acceso a los consumidores necesarios para la creación de las rutas de evacuación.

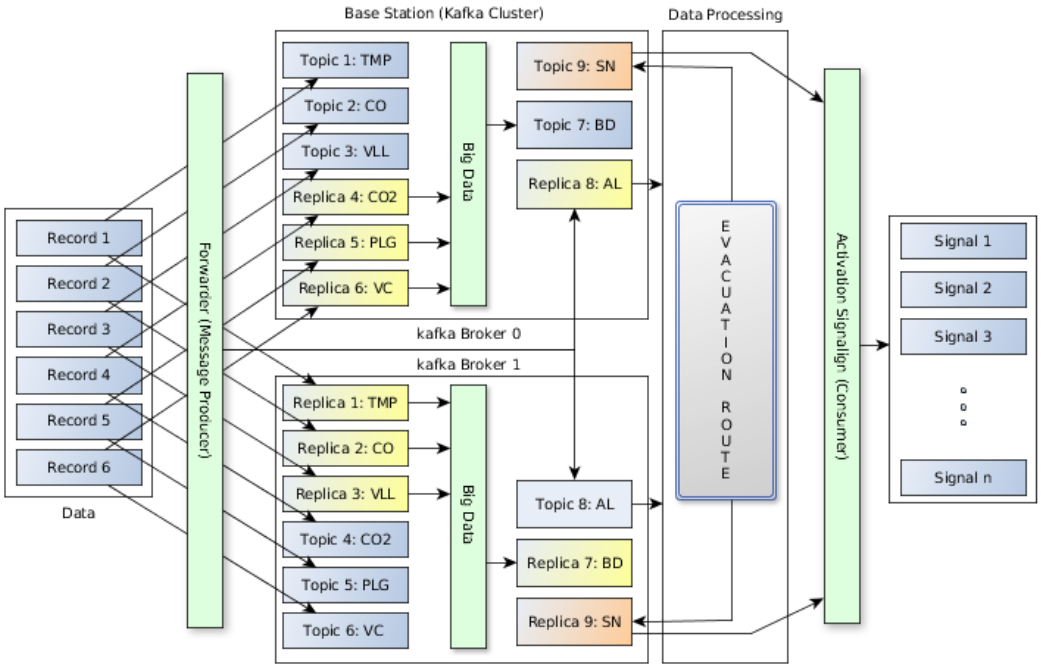

Figure 13: Esquema de distribución de Kakfa
[20]

3.3 Procesamiento

El componente de procesamiento tiene dos funciones, la primera es la de usar el API que provee el componente de MLIB de Spark para poder entrenar el modelo de aprendizaje automático haciendo uso del dataset obtenido en la fase de Adquisición y con varios algoritmos de clsificación como KNN, Desicion Tree, SVM , entre otros, lograr determinar cual es optimo para poder realizar las predicciones necesarias. La segunda función que realiza este componente es la de realizar las predicciones haciendo uso de los datos que proveen los nodos de la red, siempre y cuando ya se haya establecido el modelo de aprendizaje automático en la fase anterior.

Para poder realizar las dos funciones anteriormente descritas, se utiliza el esquema presentado en la Fig 14 , esta figura muestra un cluster de spark con tres instancias, cada una desplegada en distintos contenedores, la primera instancia, es la que actúa de master y las otras dos de workers, esto permite entre otras cosas poder distribuir la carga de trabajo cuando se tiene una flujo grande de datos, es decir si en un momento determinado muchos de los nodos envían información al modelo, el procesamiento se distribuirá a cada uno de los workers, para poder de esta forma realizar las predicciones de la manera más rápida.

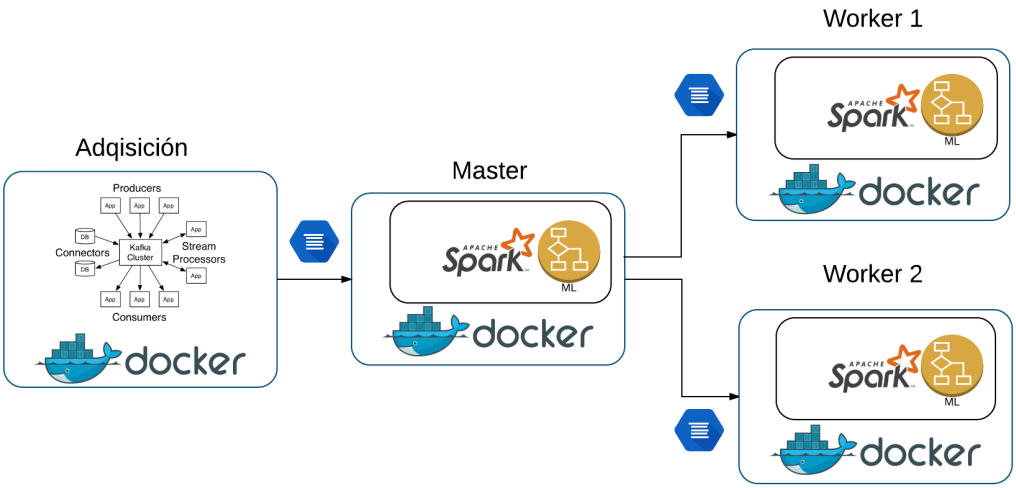

Figure 14: Esquema de Procesamiento

3.4 Almacenamiento y Presentación

Este componnte es el encargado de almacenar todos los eventos generados por los nodos sensores de la red, además de guardas las alertas generadas por el modelo entrenado en Spark. A esta información también se le adiciona, las rutas generadas por las alertas recibidas. Para ello se hace uso de MongoDB como motor de base de datos para el almacenamiento de la información antes mencionada.

Por otra parte como componente final provee una Interfaz de usuario, la misma que es definida y gestionada utilizando Apache Zepeling, que lo que hace es acceder directamente a la base de datos creada en MongoDB y extraer los datos de interés para presentarlos finalmente al usuario por medio de una interfaz web. En la Fig 15, se puede apreciar que al igual que los anteriores componentes, tanto mongoDB como Zepeling, se encuentran desplegados en contenedores, estos a sus vez se comunican entre sí para poder mostrar la información necesaria al usuario.

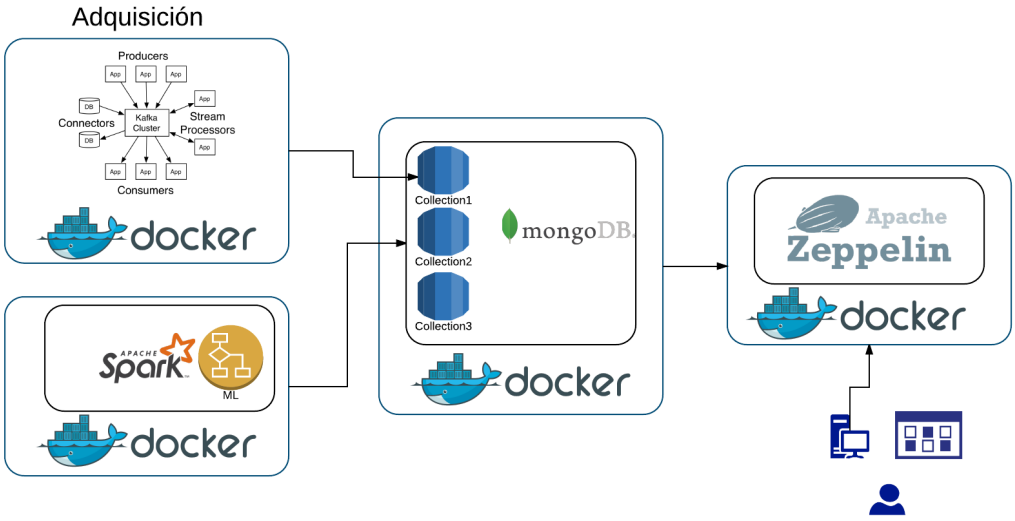

Figure 15: Esquema de Almacenamiento y Presentación

4 Implementación y Pruebas

En esta sección se describe cada uno de los componntes necesarios para poder realizar la simulación y puebas del sistema descrito en las secciones anteriores. Se toma como punto de partida la descripción del enterno de simulación describiendo cada uno de los parametros que forman parte de la simulación, y tomando como punto de partida la arquitectura planteada. Posteriormente se describe el proceso de entrenamiento del modelo y las metricas asociadas al mismo, para finalmente, realizar la prueba y ejecución del sistema completo.

4.1 Entorno de Simulación

El entorno de simulación que se utiliza en este desarrollo tiene como base una el modelamiento de una infraestructura física como la mostrada en la Fig 16, en donde se modela un área 1692 m^2 de un edificio administrativo ubicado en la Escuela Superior Politécnica del Litoral, en donde se presenta en detalle la estructura de cada área que conforma la estructura, para que con base en ello se pueda determinar la ubicación de los nodos sensores que van a enviar la información al centro de procesamiento (siendo este el modelo entrenado con los datos provenientes de los sensores), y este es el que se encargará de detectar si es necesario generar una alerta para encender la señalética y empezar con el proceso de evacuación.

En la misma figura, se puede apreciar el número y la ubicación de cada uno de los nodos sensores. Se ha elegido la utilización de siete nodos para cubrir toda el área presentada, la ubicación de cada uno de los nodos esta determinada por la cantidad de áreas con salidas directas dentro del edificio, además se cuentan con tres salidas de evacuación representadas por escaleras.

Cada nodo esta conformado por una serie de sensores, en nuestro caso específico se utilizan cinco sensores por cada nodo, la descripción de cada uno de estos sensores se detalla en la subsección 4.1.1. Con todo lo anteriormente descrito el escenario de simulación quedaría conformado por siete nodos, cada uno con cinco sensores, distribuidos estratégicamente dentro del edificio, por lo que tendríamos un total de 35 dispositivos que estarían enviando datos para ser procesado en un instante dado.

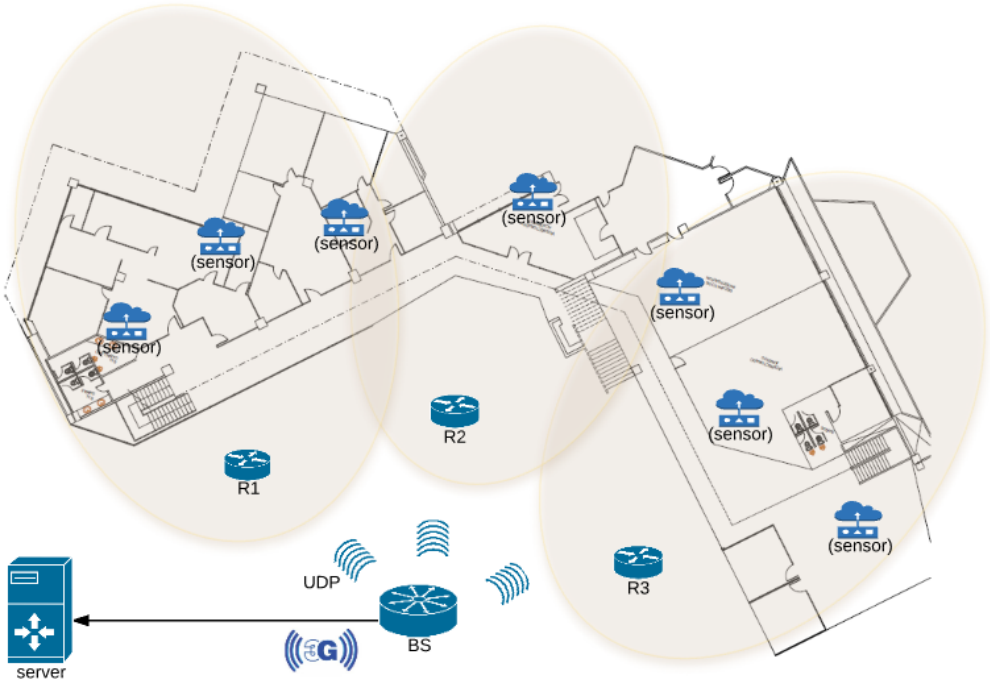

Figure 16: Diagrama de Infraestructura de Simulación

4.1.1 Descripción de variables

Como se mencionó anteriormente cada nodo está compuesto por sensores, y en el entorno definido en este trabajo dichos sensores se clasifican tal y como se muestra en la Tabla 1. Cada sensor tiene un valor de activación que es el que indica que pasado ese umbral se debería generar una alarma. A partir de dichos parámetros se realiza una simulación para poder lograr obtener un data set que sirva para entrenar de forma eficaz al modelo.

Table 1: Descripción de sensores

Sensor	**Descripción**	**Valor de activación**
TMP	Sensor de Temperatura ambiental	>= 40
CO	Sensor de Monóxido de Carbono	>= 200
CO2	Sensor de Dióxido de Carbono	>= 400
PLG	Sensor de Gas licuado de pretróleo	>= 750
VLL	Sensor Medidor de liquido vertical	>= 40

4.2 Simulación

Esta simulación se realizó con el objetivo de poder generar un dataset que permitiera generar un modelo de aprendizaje automático a partir de los datos obtenidos por la misma. Inicialmente, para poder realizar esta simulación se realizó la implementación del entorno presentado en el apartado 3.2, para ello se implementó un programa en java que permite generar una cantidad variable de nodos con cada uno de los sensores definidos en la sección 4.1.1 ver Fig 17. Este programa además permite la inserción directa de umbrales en los cuales

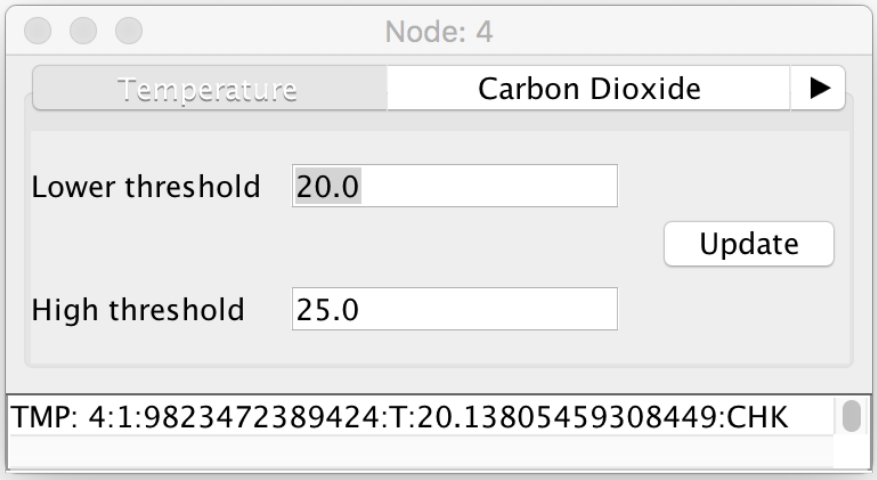

Figure 17: GUI de simulador

se generan aleatoriamente valores por cada sensor. Además a cada nodos se le puede definir una frecuencia de generación de datos. En la Fig 18 se muestra la ejecución del programa de generación de valores de los sensores con siete nodos, en ella se puede apreciar, que se presenta una ventana por cada nodo, en la que se puede definir los umbrales para cada uno de los cinco sensores asociados a él.

Para la generación y obtención del dataset se tuvieron en cuenta los siguientes parámetros:

- Número de nodos: 7
- Número de sensores por nodo: 5

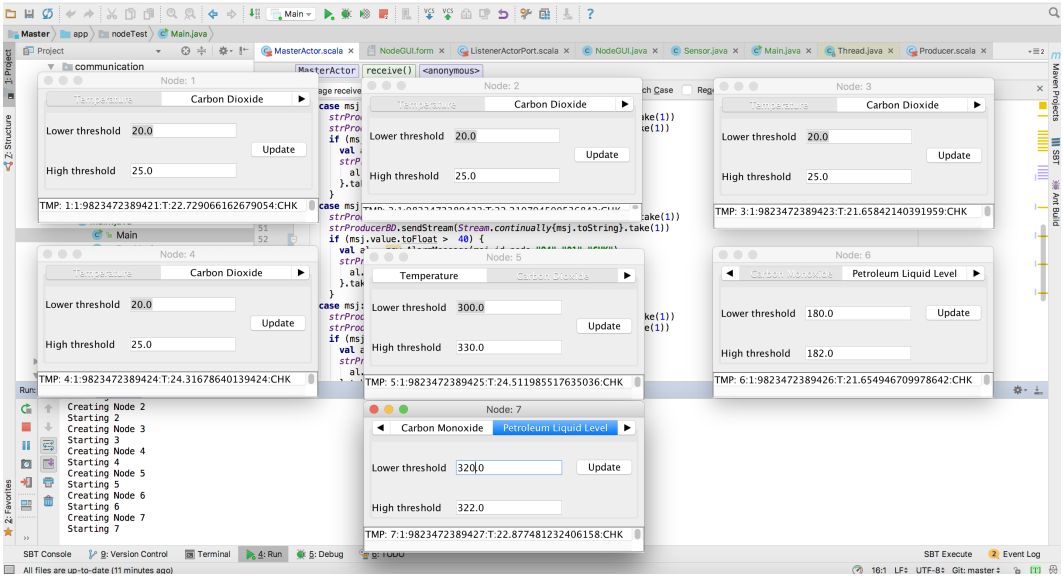

Figure 18: Interfaz de Software simulador de valores de sensores

- Frecuencia de generación de datos : 1 datos por cada 60 segundos
- Duración aproximada de la simulación: 120 minutos

Partiendo de los parámetros anteriormente definidos, los datos generados, fueron almacenados directamente en una base de datos en MongoDb con la estructura presentada en la Tabla 2

Table 2: Modelo de Base de Datos MongoDB

Parámetro	**Tipo**	**Descripción**
id	Object	ObjectID("")
id_node	Integer	Número del nodo
status	Integer	Estatus del Nodo
value	Integer	Valor del sensor
type	Sring	{T,X,C,P,V} - tipo de Sensor
date	Date	Fecha

En esta simulación se trato de modelar dos tipos de incidencias, la primera que se dio lugar en la primera hora de simulación se representa como la iniciación de un incendió originado en el nodo 1, por lo cual sus sensores de Temperatura, CO y CO2 empezaron a sufrir incrementos, además los nodos 2 y 3 también se vieron afectados al cabo de 5 minutos de iniciado el fuego generándose en ellos también variaciones y activaciones de las alertas, después de 15 minutos la afectación llego hasta el sensor 4 que también empezó a mostrar

variación en las mediciones de los sensores, finalmente al cabo de 30 minutos los nodos mencionados anteriormente dejaron de generar variaciones.

El segundo escenario planteado fue la rotura de una tubería de GLP dejando escapar este gas y afectando inicialmente a al nodo 6, en este nodo se empieza a notar la variación en la medida del sensor de GLP, hasta que se activa la alarma, el gas continua su expansión llegando hasta los nodos 4 y 5 quienes también detectan generan un cambio en los valores del sensor de GLP, finalmente al cabo de 10 minutos se genera una explosión que hace que todos los nodos 4 5 y 6 generen alertas en todos sus sensores a excepción del sensor de VLL, el fuego rápidamente se expande hacia los otros sensores del edificio y se capturan dichos valores hasta que finalmente la simulación finaliza con la afectación de todos los sensores del edificio.

Como consecuencia de las simulaciones realizadas, se obtuvieron 448 registros por cada sensor tal y como se puede apreciar en la Fig 19, por lo que al tener 7 nodos, habríamos obtenido un dataset de 3136 datos, a esto además, le sumamos los datos obtenidos en simulaciones en las cuales no se hicieron variar los valores de los sensores obteniendo un total de 6685 registros.

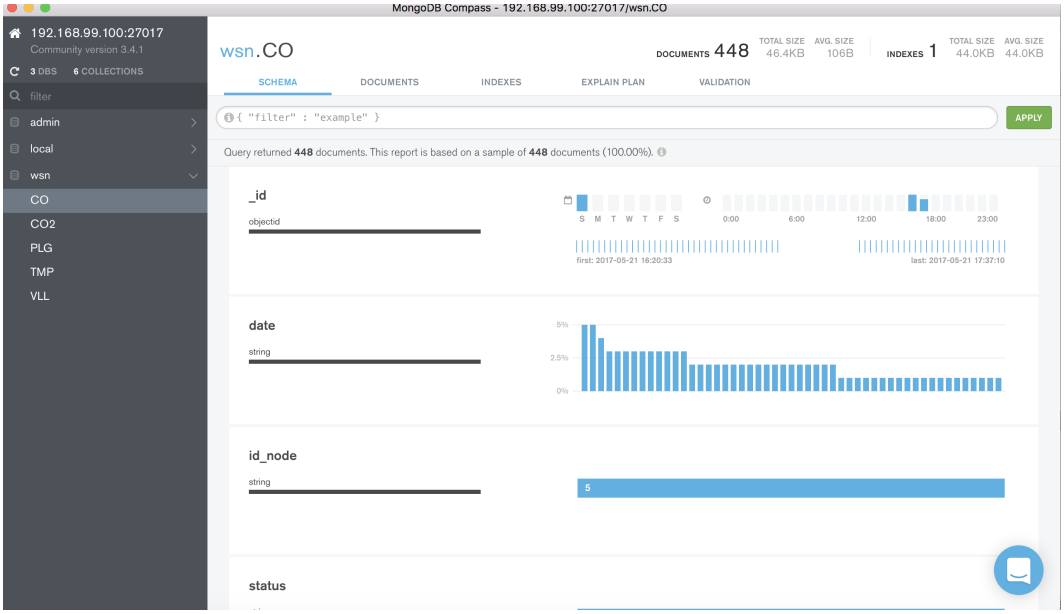

Figure 19: Valores almacenados en base de datos

Finalmente por medio de la simulación descrita en los párrafos anteriores se logró obtener el dataset que se muestra en la Fig 20, este data set esta compuesto por 6685 registros y 8 columnas, donde las columnas indican correspondientemente, el id del nodo que envió el dato, el identificados del sensor

asociado al nodo, la dirección MAC del dispositivo representada de forma numérica la descripción del tipo de sensor utilizado, el valor medido de dicho sensor, un campo de comprobación y finalmente el campo alarma representado por 1 si se ha activado o 0 sino lo ha hecho. Con los datos recopilados en este dataset se puede realizar un análisis de los datos y correspondientemente entrenar el modelo, todo este desarrollo se presenta en la subsección 4.3.

	nodo	tipo	mac	tipoS	valor	signal	alarma
0	5	1	9823472389425	temperatura	39.291137	coHK	0
1	1	1	9823472389421	temperatura	43.447629	coHK	1
2	4	1	9823472389424	temperatura	36.479601	coHK	0
3	3	1	9823472389423	temperatura	23.265346	coHK	0
4	2	1	9823472389422	temperatura	31.741695	coHK	0
5	6	1	9823472389426	temperatura	24.364914	coHK	0
6	5	1	9823472389425	temperatura	33.661448	coHK	0
7	4	1	9823472389424	temperatura	23.222862	coHK	0
8	1	1	9823472389421	temperatura	29.886574	coHK	0
9	3	1	9823472389423	temperatura	32.881698	coHK	0

Figure 20: Dataset obtenido de la simulación

4.3 Entrenamiento del Modelo

Con el dataset obtenido en la sección anterior, se puede entrenar el modelo haciendo uso de distintos algoritmos hasta poder encontrar el que represente de manera mas precisa y evitando el sobreajuste a los datos obtenidos, además de proveer una alta confiabilidad en la generación de alertas.

Para realizar el entrenamiento del modelo, previamente se preprocesaron los datos, eliminando y corrigiendo los errores que se presentaran, y además se realizo un análisis de los mismos para poder determinar cuales de todos los parámetros son lo que aportan información relevante al modelo y cuales no, en la Fig 21 se tiene una representación gráfica de cada uno de los campos

del dataset, en ellas se pueden observar el comportamiento de los datos, por ejemplo en el gráfico del campo alarma se ve como están distribuidos los datos en donde se muestra que alrededor de 600 de los registros muestran que una alarma ha sido activada, mientras que los restantes representa que no se han registrado incidentes dentro del edificio. Adicionalmente a lo mostrado en las

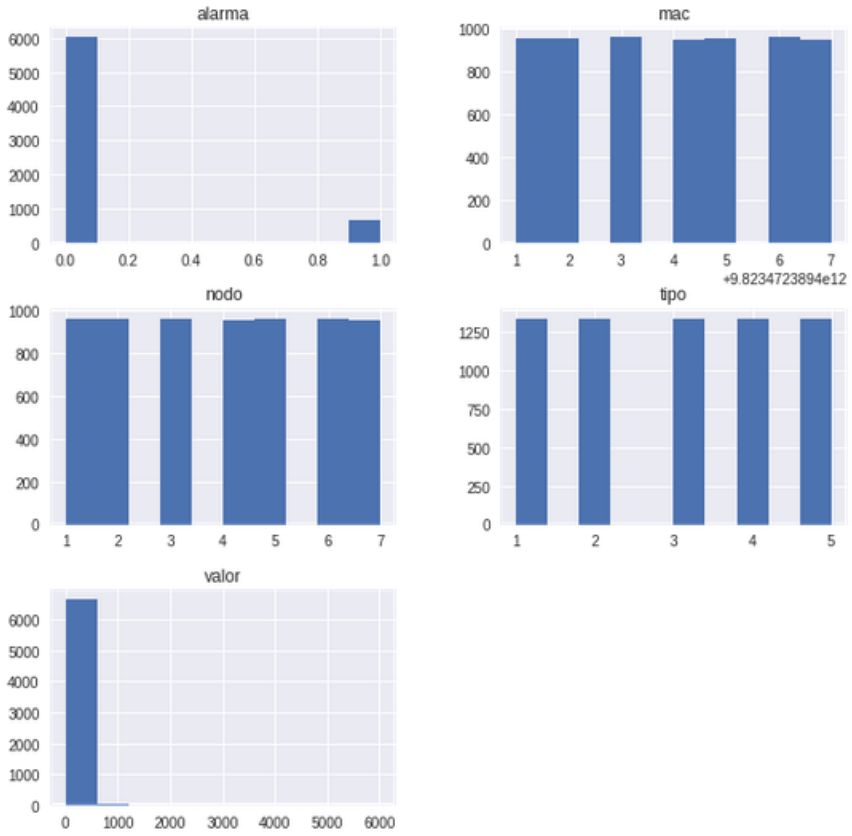

Figure 21: Distribución de los datos por campo

gráficas anteriores, en la Fig 22 se muestra la relación que existe entre el campo alarma y el campo valor, mostrando la distribución de como los datos de estos dos campos están correlacionados en algo que es importante tener el cuenta para saber como se debe comportar el modelo.

Con todo lo anteriormente analizado, se puede determinar que los parámetros que son relevantes para el entrenamiento del modelo son: nodo, tipo, valor y alarma, por lo cual todos los valores asociados a estos campos vana a ser enviados como parámetros para el entrenamiento del modelo, usando a el campo alarma como campo de comprobación una vez que el modelo haya sido entrenado.

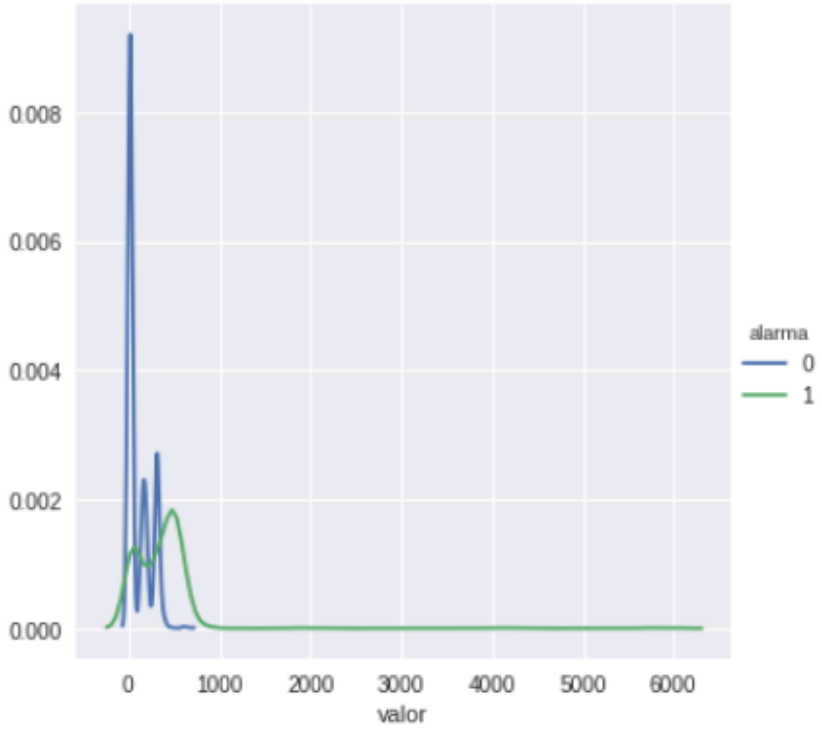

Figure 22: Gráfica de correlación Alarma-Valor

4.3.1 Elección de algoritmo

Para poder realizar la elección del algoritmo adecuado para este sistema es necesario realizar pruebas con varios algoritmos y dependiendo de los resultados obtenidos en cada uno de ellos se determinará cual es el mejor, teniendo como indicadores la precisión, el sobreajuste y los valores de falsos positivos y falsos negativos que nos arroje la matriz de confusión.

En este desarrollo se planteó el uso de por lo menos 3 algoritmos para entrenar el modelo siendo estos: KNN, Decision Tree y SVM, por lo que el dataset obtenido será usado con cada uno de estos algoritmos y a continuación se muestran cada uno de los resultados obtenidos. Cada uno de los algoritmos fue implementado haciendo uso de las librerías de sckitlearn bajo código desarrollado en phyton3, como infraestructura inicial de prueba de entrenamiento se utilizó Jupyter, pero al Spark poseer un mayor despliegue de servicios de streaming desarrollados en scala, este mismo código será adaptado con las librerías específicas de scala pero siguiendo la misma estructura, en la sección 4.4 donde se hace la prueba y ejecución de toda la arquitectura presentada.

KNN

El codigo para realizar el entrenamiento del modelo utilizando este algoritmo y el dataset proporcionado se muestra en la Fig 23 alli se describen cada uno des los pasos que se siguirón, desde la disvision del conjuno de entrenamiento y prueba hasta las métricas obtenidas En este caso al entrenar el modelo, se obtuvo como resultado una precisión del 0.99 y la matriz de confusión nos aporta con información adicional, mostrándonos que como resultado del entrenamiento se tiene que se generaron 19 falsos positivos y 6 falsos negativos. Para tratar de mejorar estos valores se hizo uso de la técnica de validación

```
features = ["nodo","tipo","valor"]
x = df[features].values
y = df['alarma'].values
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=33)
```

```
#Se entrena el modelo haciendo uso del algoritmo Knn
model = KNeighborsClassifier(n_neighbors=15)
model.fit(x_train, y_train)
predicted = model.predict(x_test)
expected = y_test
```

```
# Se mide la precisión del algoritmo
metrics.accuracy_score(expected, predicted)
```

```
0.98504784688995217
```

```
#  Matiz de Confusión
print(metrics.confusion_matrix(expected, predicted))
```

```
[[1489   19]
 [   6  158]]
```

```
# Reporte de clasificación
print(classification_report(expected, predicted))
```

```
             precision    recall  f1-score   support

          0       1.00      0.99      0.99      1508
          1       0.89      0.96      0.93       164

avg / total       0.99      0.99      0.99      1672
```

Figure 23: Código Knn

cruzada con kfold, después de realizar esta etapa se obtuvo una precisión menor tal y como se muestra en la Fig 24, pero allí se muestra que a pesar de tener una buena precisión, el modelo con este algoritmo recién después de los 3000 datos de entrenamiento se acopla de forma correcta a los datos con una precisión máxima de 0.84.

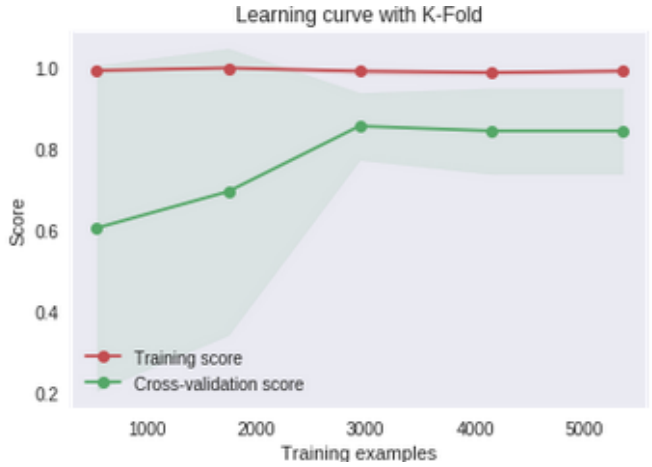

Figure 24: Gráfica de evaluación de entrenamiento KNN

Decision Tree

Por otra parte en la Fig 25 se muestra la implementación bajo el algoritmo Desición tree y se obtiene una precisión de 1.0 y la matriz de confusión muestra que todos los datos han sido clasificados de manera correcta, es decir no hay presentes valores clasificados como falsos positivos o falsos negativos, sin embargo para conocer si hay sobre ajuste con respecto a los datos de entrenamiento, se realizó también la validación cruzada, obteniendo como resulta la gráfica mostrada en la Fig 26, en donde se muestra que a partir de las 4500 muestras el modelo se generaliza y se evita completamente el sobreajuste

```
max_depth=3
random_state=1

# Se crea el modelo usando decision tree  con max_depth=3 y random_state=1
model = tree.DecisionTreeClassifier(max_depth=max_depth, random_state=random_state)

# Se estandarizan los valores de entrenamiento y prueba y se entrena el modelo
scaler = preprocessing.StandardScaler().fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
model.fit(x_train, y_train)

predicted = model.predict(x_test)
expected = y_test
metrics.accuracy_score(expected, predicted)
```

```
1.0
```

```
print(metrics.confusion_matrix(expected, predicted))
```

```
[[1337]]
```

```
print(classification_report(expected, predicted))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00      1337

avg / total       1.00      1.00      1.00      1337
```

Figure 25: Implementación Decision Tree

```
Scores in every iteration [ 0.99850523  1.          1.          1.
  1.          1.          1.        ]
Accuracy: 1.00 (+/- 0.00)
```

```
plot_learning_curve(model, "Learning curve with K-Fold", x, y, cv=cv)
```

```
<module 'matplotlib.pyplot' from '/home/andres/anaconda3/lib/python3.6/
```

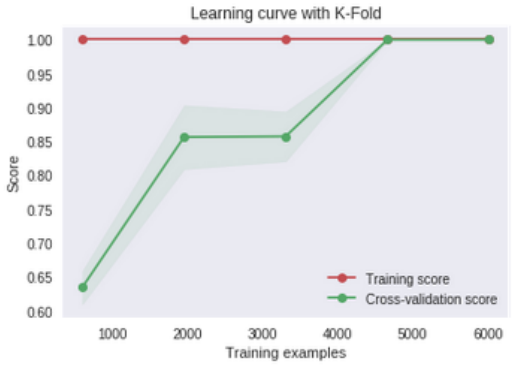

Figure 26: Gráfica de evaluación de entrenamiento Decision tree

SVM

Adicionalmente a los algoritmos anteriormente presentados, se probó con el algoritmo SVM para tratar de determinar si este mostraba alguna ventaja frente a los anteriores, su implementación se muestra en la Fig 27, siguiendo la misma metodología planteada en los algoritmos Knn y Decision Tree. Sin embargo este algoritmo mostró una precisión menor que los algoritmos antes analizados legando esta a 0.92, además la matriz de confusión muestra que se han generado una cantidad de 123 falsos negativos, haciendo de este el modelo con las características menos favorables de los analizados en este desarrollo.

Como resultado del análisis realizado por medio de la implementación de los algoritmos KNN, Decision Tree y SVM, se puede concluir que el algoritmo óptimo para crear el modelo para este sistema es el obtenido por el uso de Decision Tree, debido a que lo que se busca en este tipo de sistemas es obtener la mejor precisión, el menor sobre ajustes y la menos cantidad de falsos negativos, y el algoritmo mencionados anteriormente cumple con todas y cada una de estas métricas. Por lo tanto el algoritmo que se implementa el el escenario totalmente integrado es el de Decision Tree.

```
types_of_kernels = ['linear', 'rbf', 'poly']

kernel = types_of_kernels[0]
gamma = 3.0

# Create kNN model
model = SVC(kernel=kernel, probability=True, gamma=gamma)
```

```
#This step will take some time
# Train - This is not needed if you use K-Fold

model.fit(x_train, y_train)

predicted = model.predict(x_test)
expected = y_test
```

```
# Accuracy
metrics.accuracy_score(expected, predicted)
```

```
0.92404306220095689
```

```
# Confusion matrix
print(metrics.confusion_matrix(expected, predicted))
```

```
[[1504    4]
 [ 123   41]]
```

```
# Report
print(classification_report(expected, predicted))
```

```
             precision    recall  f1-score   support

          0       0.92      1.00      0.96      1508
          1       0.91      0.25      0.39       164

avg / total       0.92      0.92      0.90      1672
```

Figure 27: Implementación SVM

4.4 Prueba y Ejecución de Modelo predictivo

En esta sección se describe el procedimiento para probar cada una de las etapas presentadas en la sección de arquitectura. Además se muestra la ejecución y funcionamiento del modelo de aprendizaje automático y sus resultados en un ambiente de pruebas.

Inicialmente, se realizó el despliegue de cada uno de los componentes del sistema en diferentes contenedores asociados a una misma interfaz de comunicación para que los contenedores pudieran enviar y recibir los datos entre si, la Fig 28, muestra una captura de los contenedores desplegados en el servidor, en ella se puede apreciar que se tienen 8 contenedores la descripción de cada uno de los servicios que se ejecutan en los contenedores se muestra en la Tabla 3, definiendo así un contenedor para cada uno de los servicios especificado en la sección de arquitectura. En las siguientes subsecciones se muestra la ejecución de los servicios mencionados funcionando de manera conjunta mostrando así como se realizó la integración total del sistema

```
andres@anmunoz:/opt/spark-1.5.1-bin-hadoop2.6/class1$ sudo docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS
                    NAMES
0bb45a1f8609        mongo                                "docker-entrypoint..."   11 minutes ago      Up 11 minutes       27017/tcp
                    mongodb
ccdc17953898        ches/kafka                           "/start.sh"              2 days ago          Up 2 days           0.0.0.0:7203->7203/
tcp, 0.0.0.0:9092->9092/tcp   kafka
c4083500cf9d        zookeeper:latest                     "/docker-entrypoin..."   2 days ago          Up 2 days           2888/tcp, 0.0.0.0:2
181->2181/tcp, 3888/tcp       zookeeper
3f53a7aa531e        dylanmei/zeppelin                    "bin/zeppelin.sh"        2 days ago          Up 2 days           0.0.0.0:8090->8080/
tcp                 eloquent_dubinsky
448d8c1cea09        bde2020/spark-worker:1.5.1-hadoop2.6 "/bin/bash /worker.sh"   3 days ago          Up 3 days           8081/tcp
                    spark-worker-1
c77ac0773b97        bde2020/spark-worker:1.5.1-hadoop2.6 "/bin/bash /worker.sh"   3 days ago          Up 3 days           8081/tcp
                    spark-worker-2
7b9699572f8f        bde2020/spark-master:1.5.1-hadoop2.6 "/bin/bash /master.sh"   3 days ago          Up 3 days           6066/tcp, 7077/tcp,
 8080/tcp           spark-master
```

Figure 28: Contenedores Docker

Table 3: Detalle de Contenedores Desplegados

Nombre	**Servicio**	**Descripción**
kafka	Kafka Brokers	Servidor kafka para comunicación con zookeeper
zookeeper	Zookeeper	Servidor zookeeper para la administración de tópicos
eloquent-dubinsky	Zeppelin	Servicio de Apache Zeppelin para visualización
spark-worker-1	Spark	Spark master para ejecución del modelo
spark-worker-1	Spark	Spark worker 1 en modo cluster
spark-worker-2	Spark	Spark worker 2 en modo cluster
mongodb	Mongo DB	Base de datos para almacenar el data set de entrenamie
kafka-consumer	Consumidor Kafka	Consumidor de kafka escuchando en el tópico AL

4.5 Desarrollo del programa y Despliegue en Spark

Para poder definir el modelo predictivo que se implementó en este proyecto, fue necesario en uso de Spark Streaming para poder manejar los datos provenientes de los tópicos definidos en Kafka y de esta forma obtener los datos en tiempo real de los sensores. Además, se utilizó el componente de Spark MLib para por medio de el, definir el modelo de aprendizaje automático y que este procese los datos obtenidos por el contexto de Spark Streaming y los envíe al tópico correspondiente. En el capitulo 3 la figura 13, se mostró cada uno de los tópicos que se utilizaron para este desarrollo, entre se reservó el tópico AL para que en él se enviaran los valores de salida del modelo predictivo. En la Fig 29 se

```
def main(args: Array[String]) {
  if (args.length < 4) {
    System.err.println("Usage: KafkaWordCount <zkQuorum><group> <topics> <numThreads>")
    System.exit(1)
  }
  var j=0
  val topic="AL"
  val Array(zkQuorum, group, topics, numThreads) = args
  val sparkConf = new SparkConf().setAppName("KafkaWordCount").setMaster("local[4]")
  val ssc = new StreamingContext(sparkConf, Seconds(2))
  val rootLogger = Logger.getRootLogger()
  rootLogger.setLevel(Level.ERROR)
  //ssc.checkpoint("checkpoint")
  val  props = new Properties()
  props.put("bootstrap.servers", "192.168.1.111:9092")
  props.put("acks","1")
  props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
  props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")
  val producer = new KafkaProducer[String, String](props)
  val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
  val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)

  val model= default.modelo.inicializarModelo(ssc)
  lines.foreachRDD { (rdd: RDD[String], time: Time) =>
    val sqlContext = new org.apache.spark.sql.SQLContext(ssc.sparkContext)
    import sqlContext.implicits._

    val wordsDataFrame = rdd.map(w => Record(w.split(":")(0).toInt,w.split(":")(1).toInt,w.split(":")(4).to
    val assembler = new VectorAssembler().setInputCols(Array("nodo", "tipo", "valor")).setOutputCol("featur
    val output = assembler.transform(wordsDataFrame)
    val data = output.select($"alarma",$"features")
    data.registerTempTable("words")
```

Figure 29: Fragmento de Código del Programa

muestra un fragmento del programa que ejecuta cada una de las funciones de manejo de streaming, definiciones del modelo y despliegue del productor kafka

fue desarrollado en Scala, quedando definidas dos clases, la clase modelo, en donde se manejan el procesamiento del data set y la definición del modelo de aprendizaje automático basado en Árbol de decisión y la clase Clasificador, en donde se crea un consumidor de Kafka para obtener los datos de los sensores, se hace el llamado a las funciones definidas en la clase modelo para crear y ejecutar el modelo y finalmente se crea un productor de kafka que registre la salida del modelo en el típico AL para que el servidor de gestión de rutas pueda consumir dicha información y definir las rutas de evacuación.

La ejecución del código descrito anteriormente, se desplegó directamente en el contenedor de spark-master, este a su vez al estar en modo cluster distribuyó la tarea de procesamiento entre sus workers tal y como se muestra en la Fig 30, en ella se puede apreciar que el programa se ha ejecutado exitosamente. Además en las Figuras 31 y 32 se muestra la ejecución de cada uno de los jobs generados por el programa, tanto gráficamente como descriptivamente.

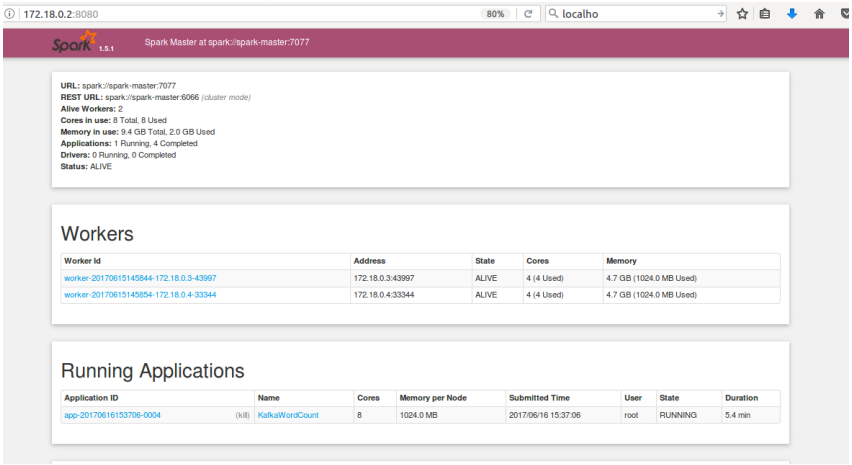

Figure 30: Web UI Cluster Spark

Por otra parte si observamos la información que muestra el programa por medio de la consola de Spark podemos aportar con la siguiente información. En la Fig 33 se muestra que en la primera ejecución del programa, se muestra la estructura que maneja el dataset que se usa para entrenamiento y prueba del modelo, luego presenta el top 20 de los registros del mismo dataset, posteriormente el modelo es entrenado con todos los datos y se obtiene una salida con un nuevo dataset, en donde se incluye el valor que predice el modelo, el valor actual de la alarma y los registros correspondiente al nodo, tipo y valor obtenido en esa muestra de prueba. Posteriormente se muestra la información referente al proceso de entrenamiento seguido y finalmente los datos de salida de el campo de predicción, el campo alarma y el registro de entrada, siendo

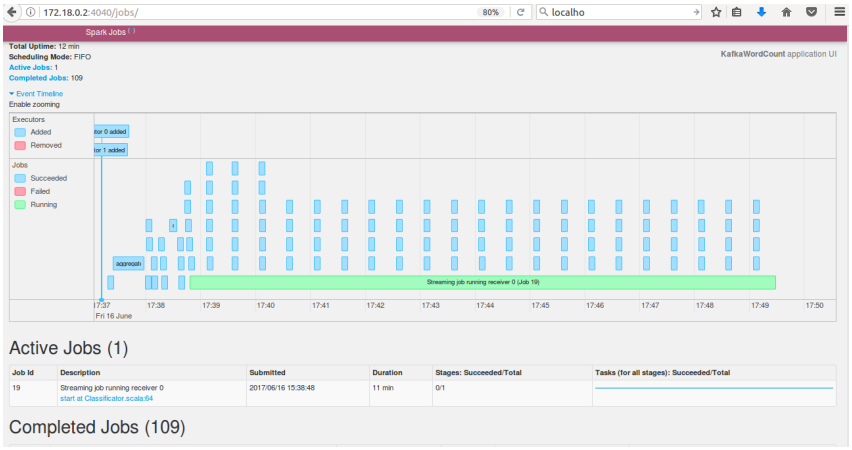

Figure 31: Descripción Gráfica de Jobs

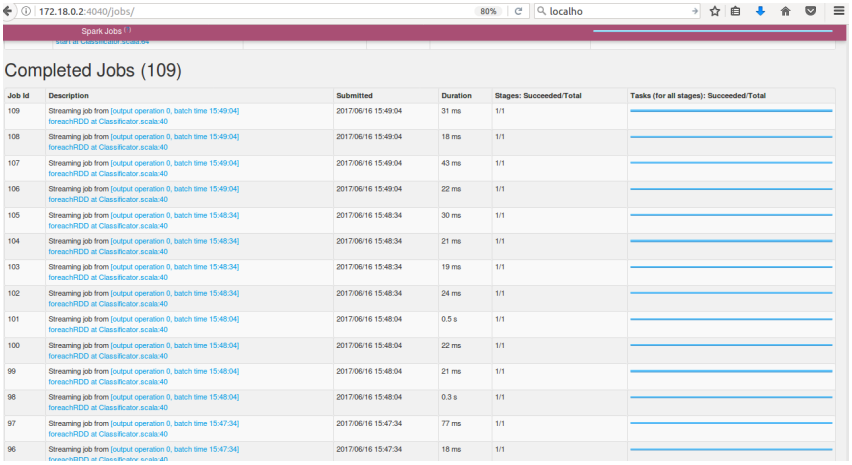

Figure 32: Descripción de ejecución de Jobs

esta misma información la que se registra directamente en el tópico AL

4.6 Prueba de Integración Completa

Para realizar la prueba de ejecución completa de el escanrio propuesto fue necesario, hacer uso de varios componentes, en primer lugar se iniciaron los servicios de kafaka y zookeeper con cada uno de los tóicos definidos en la sección 3, luego se inicio una simulación de los sensores haciendo uso del mismo programa descrito en la subsección 4.2, una vez que ya se estaban ejecutando estos servicios se realizo el despliegue de la aplicación en el cluster de spark. Como la información que se obtiene del modelo se registra directamente en el tópico LA, fue necesario usar un consumidor de kafka externo a la aplicación

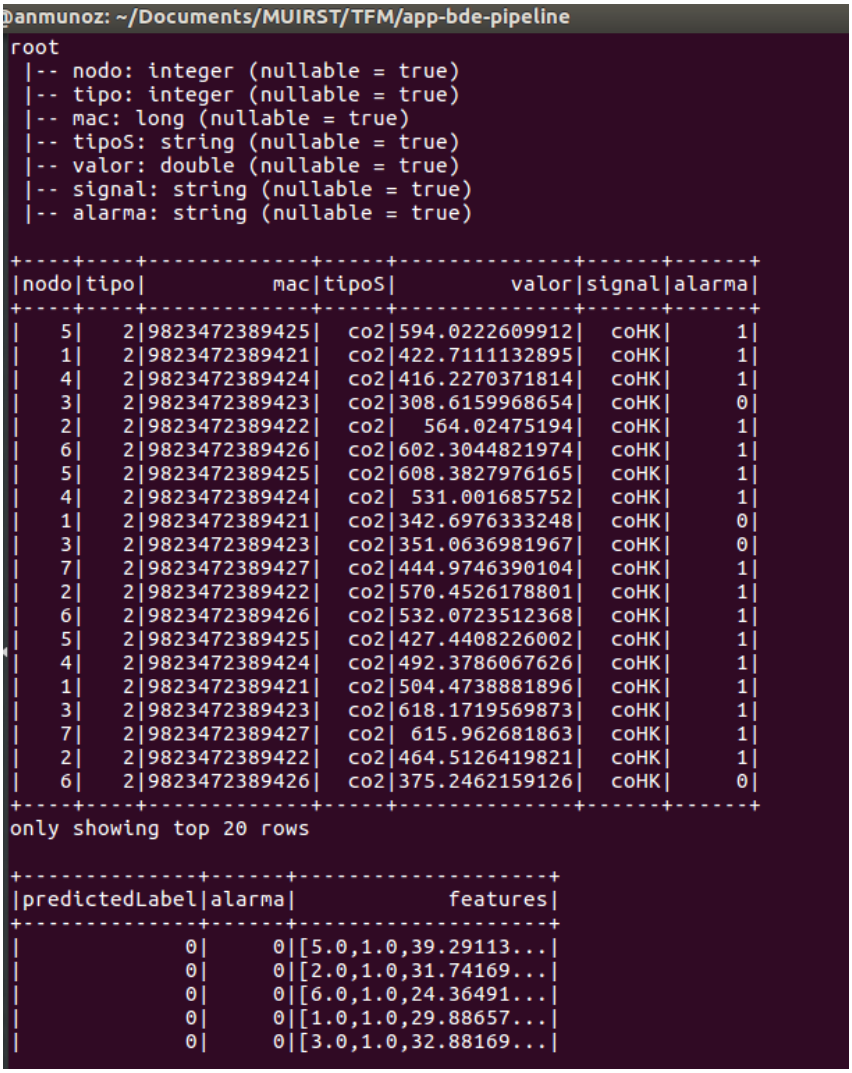

Figure 33: Salida de consola de Spark

para que este obtenga la información registrada en dicho tópico y nos permita de esta forma verificar por medio de los datos de entrada y de salida que todo el sistema estaba funcionando correctamente y que el modelo realizaba las predicciones de manera correcta. Una ves que ya todos los conpoentes se estaban ejecutando se empezó con la simulación, esta estaba compuesta por 7 nodos sensores, los cuales arrojaban valores dentro de su umbral de ejecución normal con una frecuencia de adquisición de datos de 1 valor cada 30 segundos, después de 10 minutos de ejecución normal se eligió un nodo al zar para que este empezara a subir sus niveles de los sensores por lo que al llegar al valor indicado el sistema debería detectar que se ha sobrepasado del nivel de ejecución normal y de esta forma debería generar una alarma y enviarla al tópico correspondiente. En las figuras 34 y 35 se muestra la ejecución de una des la pruebas realizadas, mostrando como el modelo realiza la predicción de

forma correcta y como esta alerta generada es correctamente registrada en el tópico AL y consumida por medio del consumidor kafka.

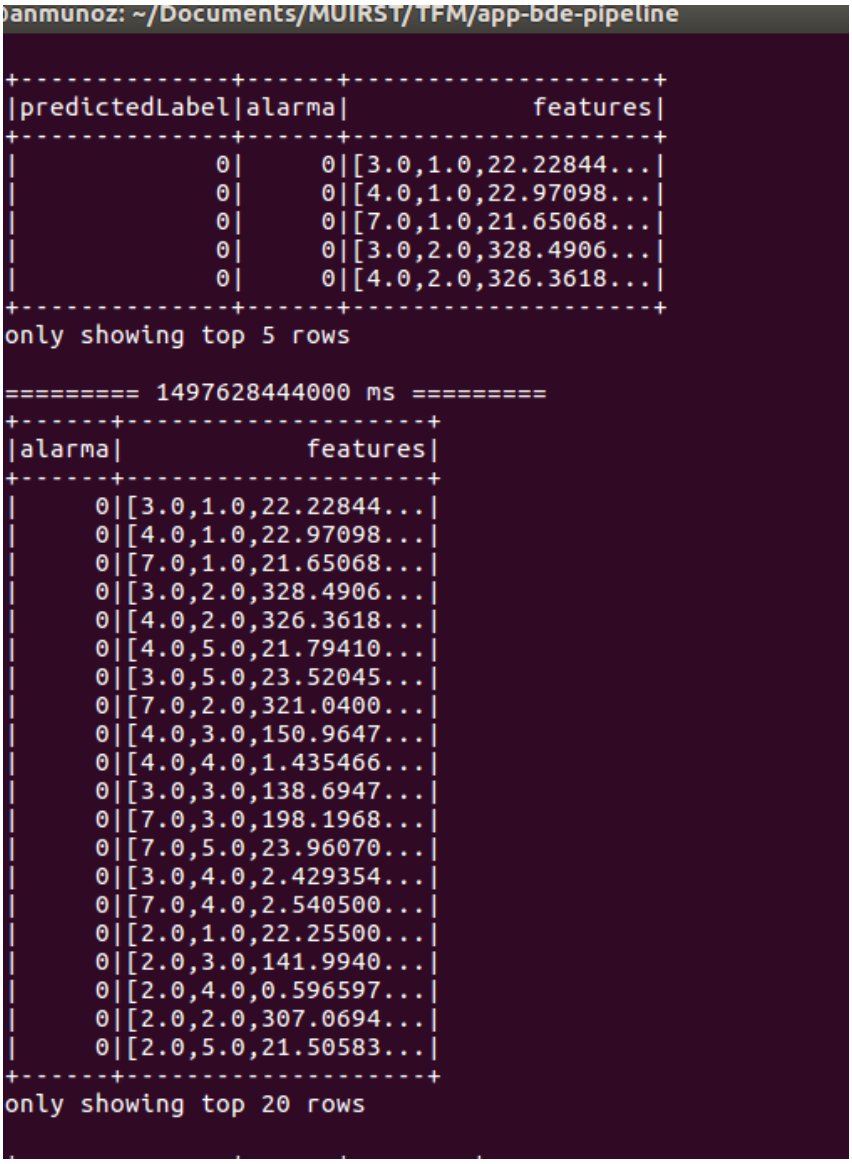

Figure 34: Ejecución del completa del Sistema

Finalmente se define como una herramienta de visualización del streaming de los valores que se están consumiendo por medio del tópico AL,a un pequeño programa ha sido desarrollado en en notebook Apache Zeppelin, el cual provee de una interfaz de visualización genérica para aplicaciones desarrolladas en scala. Este programa lo que hace es implementar un consumidor de Kafka en el tópico AL y procesar por valores allí obtenidos, esta información es guardada en una tabla temporal con la estructura de Nodo, tipo, valor y

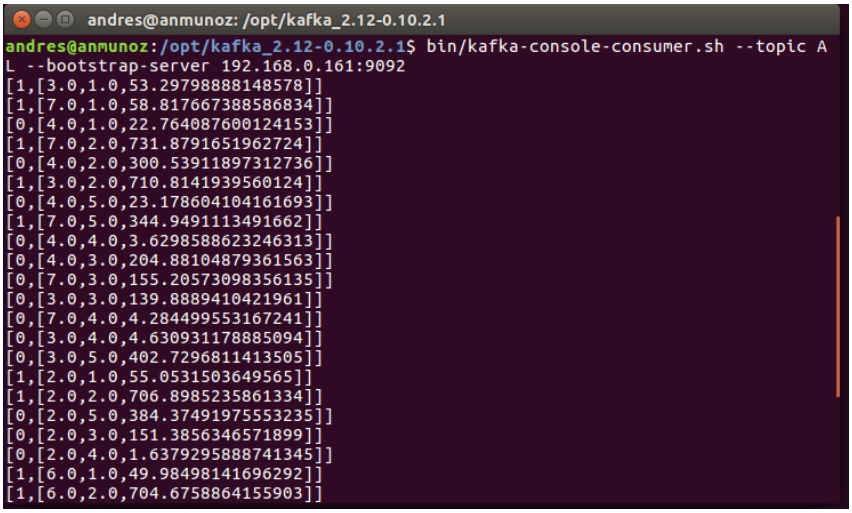

Figure 35: Salida Consumidor de AL

alarma, posteriormente como se puede apreciar el la Fig 36 en la celda inferior se despliega una gráfica con los valores consultados a la tabla temporal generada. Con dicha información se pueden generar múltiples consultas de tipo SQL y obtener y agrupar información para que esta sea presentada de forma gráfica, en este caso se realizó una consulta para obtener de la tabla los valores obtenidos correspondientes al nodo 1. Cabe recalcar que en esta herramienta se pueden añadir múltiples celdas para visualizar otros tipos de datos como , las alarmas generadas, etc. Sin embargo para la visualización y manejo de opciones mas complejas sería necesario desarrollar una aplicación que permita al usuario visualizar y administrar las diferentes funcionalidades que se requieran.

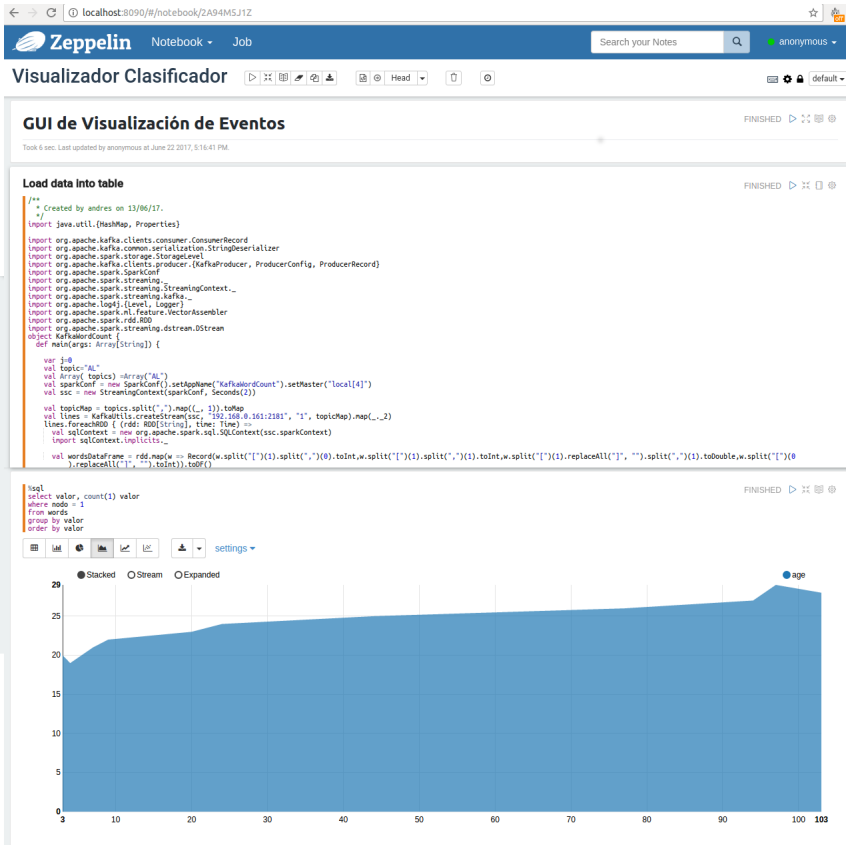

Figure 36: Visualizador en Zeppelin

5 Conclusiones

Por medio de realización del trabajo presentado en este documento, se ha permitido realizar un programa que permite clasificar con una precisión de un 0.99 si los datos generados por un sensor requieren que se genere una alerta para que se realice una evacuación de emergencia. Además la arquitectura y tecnologías utilizadas, permiten que esta implementación se utilice en entornos donde se manejan grandes volúmenes de datos, dando paso a que se adopte su aplicación tanto en entornos pequeños, medianos y grandes.

La arquitectura presentada en este estudio en combinación con las tecnologías utilizadas, permiten que este sistema pueda ser desplegado de forma fácil, rápida y confiable gracias a la utilización de contenedores para la implementación de cada uno de los servicios desplegados, facilitando así la integración, despliegue y comunicación de cada uno de dichos servicios.

Las pruebas realizadas en este desarrollo permitieron comprobar que el sistema de procesamiento y análisis predictivo de eventos es capaz de realizar un procesamiento en tiempo real de los eventos generas dentro de un entorno de simulación adaptado a interiores y que por medio de este se permite la generación dinámica de rutas de evacuación dentro de dichas inmediaciones.

Finalmente, este sistema permite que se pueda realizar una adaptación del mismo a otros entornos, debido a la flexibilidad del software desarrollado y la capacidad de entrenar el modelo con distintos datasets para otro tipo de aplicaciones, en donde, la precisión, rapidez y el manejo de grandes volumenes de datos es un requisito indispensable.

6 Bibliografía

References

[1] H. Mendonça and S. Martinez, "Modeling of a wave energy converter connected to a resistive load," in *4th International Youth Conference on Energy (IYCE)*, pp. 1–6, IEEE, 2013.

[2] M. O'Gara, "Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud," 2013.

[3] pico.dev, "Introducción y características de Dockerblog bitix," 2017.

[4] Docker.com, "What is Docker," 2017.

[5] E. A, "Docker, Qué es y sus principales características, open webinars," 2014.

[6] DockerDocs, "Docker Overview," 2017.

[7] DockerDocs, "Docker Documentation," 2015.

[8] A. Spark, "Spark Overview," 2015.

[9] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets.," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[10] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pp. 2–2, USENIX Association, 2012.

[11] T. Point, "Apache Spark- Intoduction," 2015.

[12] A. Spark, "Cluster Mode Overview," 2015.

[13] R. Ostrowski, "Introduction to Apache Spark with Examples and Use Cases," 2016.

[14] A. Kafka, "ntroduction to Apache Kafka," 2016.

[15] Datadog, "Monitoring Kafka performance metrics," 2016.

[16] A. Munoz, "Machine learning and optimization," *URL: https://www. cims. nyu. edu/˜ munoz/files/ml_ optimization. pdf [accessed 2016-03-02][WebCite Cache ID 6fiLfZvnG]*, 2014.

[17] R. Kohavi and F. Provost, "Glossary of terms," *Machine Learning*, vol. 30, no. 2-3, pp. 271–274, 1998.

[18] J. L. Solé, "Book review: Pattern recognition and machine learning. cristopher m. bishop. information science and statistics. springer 2006, 738 pages.," *SORT-Statistics and Operations Research Transactions*, vol. 31, no. 2, 2007.

[19] SAS, "Machine Learning What it is and why it matters," 2014.

[20] W. V. Vargas, A. Munoz-Arcentales, and J. S. Rodríguez, "A distributed system model for managing data ingestion in a wireless sensor network," in *Computing and Communication Workshop and Conference (CCWC), 2017 IEEE 7th Annual*, pp. 1–5, IEEE, 2017.

7 Anexos

Anexo A

Código fuente de de la clase Clasificador.scala

```
/**
  * Created by andres
  */
import java.util.{HashMap, Properties}

import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import kafka.serializer.StringDecoder
import org.apache.spark.storage.StorageLevel
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig, ProducerRecord}
import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.kafka._
import org.apache.log4j.{Level, Logger}
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.dstream.DStream
object KafkaWordCount {
  def main(args: Array[String]) {
    if (args.length < 4) {
      System.err.println("Usage: KafkaWordCount <zkQuorum><group> <topics> <numThreads>")
      System.exit(1)
    }
    var j=0
    val topic="AL"
    val Array(zkQuorum, group, topics, numThreads) = args
    val sparkConf = new SparkConf().setAppName("KafkaWordCount").setMaster("local[4]")
    val ssc = new StreamingContext(sparkConf, Seconds(2))
    val rootLogger = Logger.getRootLogger()
    rootLogger.setLevel(Level.ERROR)
    val  props = new Properties()
    props.put("bootstrap.servers", "192.168.1.111:9092")
    props.put("acks","1")
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer")
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")
    val producer = new KafkaProducer[String, String](props)
    val topicMap = topics.split(",").map((_, numThreads.toInt)).toMap
    val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(_._2)

    val model= default.modelo.inicializarModelo(ssc)
    lines.foreachRDD { (rdd: RDD[String], time: Time) =>
      val sqlContext = new org.apache.spark.sql.SQLContext(ssc.sparkContext)
      import sqlContext.implicits._
```

```
      val wordsDataFrame = rdd.map(w => Record(w.split(":")(0).toInt,w.split(":")
(1).toInt,w.split(":")(4).toDouble,0)).toDF()
      val assembler = new VectorAssembler().setInputCols(Array("nodo", "tipo",
"valor")).setOutputCol("features")
      val output = assembler.transform(wordsDataFrame)
      val data = output.select($"alarma",$"features")
      data.registerTempTable("words")
      val predecir = default.modelo.imputModelo(data,model)
      predecir.select("predictedLabel", "alarma", "features").show(5)
      val wordCountsDataFrame =
        sqlContext.sql("select * from words ")
      println(s"========= $time =========")
      wordCountsDataFrame.show()
      val record = new ProducerRecord(topic, "key"+j, "1")
      if(predecir.select($"predictedLabel").head(1).nonEmpty){

        val al = predecir.select($"predictedLabel",$"features")
        j=j+1
        al.collect().foreach(x=> producer.send(new ProducerRecord(topic, x.toString())))

      }
    }
    ssc.start()
    ssc.awaitTermination()
  }
  case class Record(nodo: Int, tipo: Int, valor: Double , alarma: Int)
  case class Stream(s: String)
}
```

Anexo B

Código fuente de de la clase modelo.scala

```
/**
  * Created by andres
  */
package default
import org.apache.spark.sql.types.IntegerType
import org.apache.spark.sql.functions.{col, lit, when}
import org.apache.spark.sql.{DataFrame, SQLContext}
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.{IndexToString, StringIndexer, VectorIndexer}
import org.apache.spark.ml.{Pipeline, PipelineModel}
import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.classification.DecisionTreeClassificationModel
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.log4j.{Level, Logger}
import org.apache.spark.streaming._

//

object modelo{

  def inicializarModelo(ssc: StreamingContext ): PipelineModel = {


    val sc = ssc.sparkContext

    val sqlContext = new org.apache.spark.sql.SQLContext(sc)
    import sqlContext.implicits._
    val df = sqlContext.read.format("com.databricks.spark.csv").option("header",
"true").option("inferSchema", "true").option("parserLib", "univocity").load("./wsnnew.csv")
    df.printSchema()
    val df2 = df.withColumn("alarma", df("alarma") cast
IntegerType).na.fill(0,Seq("alarma")).withColumn("nodo", df("nodo") cast
IntegerType).withColumn("tipo", df("tipo") cast IntegerType).withColumn("valor",
df("valor").cast("double"))
    val df3 = df2.withColumn("alarma", when($"tipo"===1 and
$"valor">=40,1).otherwise($"alarma"))
      .withColumn("alarma", when($"tipo"===2 and $"valor">=400,1).otherwise($"alarma"))
      .withColumn("alarma", when($"tipo"===3 and $"valor">=700,1).otherwise($"alarma"))
      .withColumn("alarma", when($"tipo"===4 and $"valor">=10,1).otherwise($"alarma"))
      .withColumn("alarma", when($"tipo"===5 and $"valor">=200,1).otherwise($"alarma"))
    df3.filter($"tipo".equalTo(2)).show()
    val assembler = new VectorAssembler().setInputCols(Array("nodo", "tipo",
"valor")).setOutputCol("features")
    val output = assembler.transform(df3)
    val data = output.select($"alarma",$"features")
```

```
    val labelIndexer = new
StringIndexer().setInputCol("alarma").setOutputCol("indexedLabel").fit(data)
    val featureIndexer = new
VectorIndexer().setInputCol("features").setOutputCol("indexedFeatures").setMaxCategories(4).fit(d
ata)

    val Array(trainingData, testData) = data.randomSplit(Array(0.7, 0.3))


    val dt = new
DecisionTreeClassifier().setLabelCol("indexedLabel").setFeaturesCol("indexedFeatures")


    val labelConverter = new
IndexToString().setInputCol("prediction").setOutputCol("predictedLabel").setLabels(labelIndexer.l
abels)


    val pipeline = new Pipeline().setStages(Array(labelIndexer, featureIndexer, dt, labelConverter))


    val model = pipeline.fit(trainingData)


    val predictions = model.transform(testData)


    predictions.select("predictedLabel", "alarma", "features").show(5)


    val evaluator = new
MulticlassClassificationEvaluator().setLabelCol("indexedLabel").setPredictionCol("prediction").set
MetricName("precision")
    val accuracy = evaluator.evaluate(predictions)
    println("Test Error = " + (1.0 - accuracy))

    val treeModel = model.stages(2).asInstanceOf[DecisionTreeClassificationModel]
    println("Learned classification tree model:\n" + treeModel.toDebugString)
    return model

  }
  def imputModelo(data:DataFrame, model:PipelineModel): DataFrame ={
    val prediction = model.transform(data)
    return prediction
  }
}
```

Anexo C

Código fuente aplicación Apche zeppelin

```
/**
  * Created by andres
  */
import java.util.{HashMap, Properties}

import org.apache.kafka.clients.consumer.ConsumerRecord
import org.apache.kafka.common.serialization.StringDeserializer
import org.apache.spark.storage.StorageLevel
import org.apache.kafka.clients.producer.{KafkaProducer, ProducerConfig, ProducerRecord}
import org.apache.spark.SparkConf
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.streaming.kafka._
import org.apache.log4j.{Level, Logger}
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.dstream.DStream
object KafkaWordCount {
  def main(args: Array[String]) {

    val topic="AL"
    val Array( topics) =Array("AL")
    val sparkConf = new SparkConf().setAppName("KafkaWordCount").setMaster("local[4]")
    val ssc = new StreamingContext(sparkConf, Seconds(2))

    val topicMap = topics.split(",").map((_, 1)).toMap
    val lines = KafkaUtils.createStream(ssc, "192.168.0.161:2181", "1", topicMap).map(_._2)
    lines.foreachRDD { (rdd: RDD[String], time: Time) =>
      val sqlContext = new org.apache.spark.sql.SQLContext(ssc.sparkContext)
      import sqlContext.implicits._

      val wordsDataFrame = rdd.map(w => Record(w.split("[")(1).split(",")(0).toInt,w.split("[")
(1).split(",")(1).toInt,w.split("[")(1).replaceAll("]", "").split(",")(1).toDouble,w.split("[")
(0).replaceAll("]", "").toInt)).toDF()

      wordsDataFrame.registerTempTable("words")
      wordsDataFrame.show()
      val wordCountsDataFrame =
        sqlContext.sql("select * from words ")
      println(s"========= $time =========")
      wordCountsDataFrame.show()
    }
    ssc.start()
    ssc.awaitTermination()
  }
  case class Record(nodo: Int, tipo: Int, valor: Double , alarma: Int)
  case class Stream(s: String)
```

}

Anexo D

Archivo docker-compose.yml para despligue de contenedores Spark

```
version: '2'
services:
  ###
  # Add your pipeline here
  ###

  ###
  # mu.semte.ch stack
  ###
  spark-master:
    image: bde2020/spark-master:1.5.1-hadoop2.6
    container_name: spark-master
    ports:
     - "8080:8080"
     - "7077:7077"
    environment:
     - INIT_DAEMON_STEP=setup_spark
     - "constraint:node==<yourmasternode>"
  spark-worker-1:
    image: bde2020/spark-worker:1.5.1-hadoop2.6
    container_name: spark-worker-1
    depends_on:
     - spark-master
    ports:
     - "8081:8081"
    environment:
     - "SPARK_MASTER=spark://spark-master:7077"
     - "constraint:node==<yourworkernode1>"
  spark-worker-2:
    image: bde2020/spark-worker:1.5.1-hadoop2.6
    container_name: spark-worker-2
    depends_on:
     - spark-master
    ports:
     - "8082:8081"
    environment:
     - "SPARK_MASTER=spark://spark-master:7077"
     - "constraint:node==<yourworkernode2>"
```