UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Diseño y desarrollo de un entorno basado en microservicios securizados Docker para la migración de una plataforma de conciencia cibersituacional.

ADRIÁN CALLEJAS ZURITA

2021

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Título: Diseño y desarrollo de un entorno basado en microservicios securizados Docker para la migración de una plataforma de conciencia cibersituacional.

Autor: D. Adrián Callejas Zurita.

Tutor: D. Mario Sanz Rodrigo.

Ponente: D. Víctor Abraham Villagrá González.

Departamento: Departamento de Ingeniería de Sistemas Telemáticos.

MIEMBROS DEL TRIBUNAL

Presidente: D. ……

Vocal: D. ……..

Secretario: D. ……..

Suplente: D. ……..

Los miembros del tribunal arriba nombrados acuerdan otorgar la calificación de: ……

Madrid, a de de 20…

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

Diseño y desarrollo de un entorno basado en microservicios securizados Docker para la migración de una plataforma de conciencia cibersituacional.

ADRIÁN CALLEJAS ZURITA

2021

Agradecimientos

En primer lugar, me gustaría dar las gracias a Mario, que desde el primer momento me ha apoyado en la realización del TFG, guiándome y aconsejándome cuando lo necesitaba.

También dar las gracias a todos los profesores que han contribuido a mi aprendizaje en la escuela y que me han ayudado a ser parte de como soy hoy en día, en especial a Víctor por contar conmigo para este proyecto.

Mención especial a mis padres y mi hermana, que siempre están dando todo por mí y escuchándome cuando lo necesito y a toda mi familia, que me ha apoyado desde el primer momento en todo y siempre confía en mí, especialmente a mi abuela que fue quien me enseñó que era eso de los números.

Gracias a todos mis amigos, que me han cuidado en todos los momentos difíciles que he tenido a lo largo de estos cuatro años y que me han dado todos los buenos momentos que se pueden pedir, además de aguantarme explicándoles este trabajo reiteradas veces sin entender nada. Y como no, en especial a Carlos y Javier, que han conseguido sobrevivir conmigo y a mí en todos los trabajos y laboratorios que hemos hecho en estos cuatro años.

Con este trabajo se cierra una etapa de mi vida de la que todos habéis formado parte y que no habría sido igual sin vosotros. De corazón, gracias.

Abstract

Nowadays, cybersecurity is one of the requirements on which any environment exposed to the Internet is based and one of the main concerns of companies, which have started to become aware of its importance due to the increase in attacks in recent years.

One of the most important parts of the securitization process in this context is monitoring to detect and identify patterns of cyber-attacks. This is what is being pursued in the PLICA (Plataforma Integrada de Conciencia Cibersituacional) project at RSTI (Redes y Servicios de Telecomunicación e Internet) in collaboration with the COINCIDENTE (Cooperación en Investigación Científica y Desarrollo en Tecnologías Estratégicas) government program, whose objective is focused on the use of multiple sources and sensors and the processing of the information obtained from these, through anomaly detection and correlation techniques and predictive modelling with machine learning techniques such as Bayesian or neural networks. Starting from this, cybersituational awareness will be provided through a visualization console, risk indicators and command and control tools.

The overall objective of this Bachelor Thesis will be to design and develop a system in a securitized Docker environment for the migration of the cybersituational awareness platform of the PLICA project. With this, it is intended to facilitate its scalability, increase its deployment speed, reduce as much as possible the necessary storage capacity and allow the elements that compose it to work individually in different containers. Thus, in the event of having to restart a specific application, it is not necessary to do so with the entire environment, with the loss of time and productivity that this entails. At the same time, all containers, technologies and communications will be secured by bastioning them with users and privileges and using security protocols to encrypt communications and, thanks to that, avoid the loss of confidentiality and integrity of the information being transmitted.

To achieve this, a methodology will be followed in which the Docker and Docker Compose tools will be used for the development of the applications and the results achieved will be presented, with tests of these deployed, individually and globally, both at a functional and security level.

Keywords: Cybersecurity, Docker, Docker Compose, PLICA, Container, Virtualization, Kafka, Spark, Elasticsearch.

Resumen

Hoy en día, la ciberseguridad es uno de los requerimientos y pilares fundamentales sobre los que se sustenta cualquier entorno expuesto a internet y una de las principales preocupaciones de las compañías, que se han empezado a concienciar de la importancia de esta debido al incremento de los ataques realizados en los últimos años.

Una parte fundamental del proceso de securización en este contexto es la monitorización con el fin de detectar e identificar patrones de ciberataques. Esto es lo que se está tratando de conseguir en el proyecto PLICA (Plataforma Integrada de Conciencia Cibersituacional) en el RSTI (Redes y Servicios de Telecomunicación e Internet) en colaboración con el programa de gobierno COINCIDENTE (Cooperación en Investigación Científica y Desarrollo en Tecnologías Estratégicas), cuyo objetivo se centra en el uso de múltiples fuentes y sensores y el tratamiento de la información obtenida de estos, mediante técnicas de detección y correlación de anomalías y modelado predictivo con técnicas de aprendizaje automático como redes bayesianas o neuronales. Con todo ello, se proporcionará conciencia cibersituacional a través de una consola de visualización, indicadores de riesgo y herramientas de comando y control.

El objetivo general de este trabajo de fin de grado será diseñar y desarrollar un sistema en un entorno Docker securizado para la migración de la plataforma de conciencia cibersituacional del proyecto PLICA. Con esto, se pretende facilitar su escalabilidad, incrementar su velocidad de despliegue, reducir al máximo la capacidad de almacenamiento necesaria y permitir que los elementos que lo componen funcionen individualmente en diferentes contenedores. Así, en caso de tener que reiniciar una aplicación específica, no es necesario hacerlo con todo el entorno, con la pérdida de tiempo y productividad que esto conlleva. Al mismo tiempo, se realizará una securización de todos los contenedores, de las tecnologías y de las comunicaciones mediante el bastionado de estos con usuarios y privilegios y la utilización de protocolos de seguridad para encriptar las comunicaciones y evitar así la pérdida de la confidencialidad y la integridad de la información que se está transmitiendo.

Con el fin lograr esto, se seguirá una metodología en la que se emplearán las herramientas Docker y Docker Compose para el desarrollo de las aplicaciones y se presentarán los resultados alcanzados con pruebas de estas desplegadas, individualmente y en global, tanto a nivel funcional como a nivel de seguridad.

Palabras clave: Ciberseguridad, Docker, Docker Compose, PLICA, Contenedor, Virtualización, Kafka, Spark, Elasticsearch.

Índice general

Índice de figuras

Índice de tablas

Glosario

API Interfaz de programación de aplicaciones (Application Programming Interface)

COIT Colegio Oficial de Ingenieros de Telecomunicación

ECDSA Algoritmo de Firma Digital de Curva Elíptica (Elliptic Curve Digital Signature Algorithm)

ETSIT Escuela técnica superior de ingenieros de telecomunicación

ED Etapas de diseño

GB Gigabyte

IP Protocolo de internet (Internet Protocol)

JSON Notación de objeto de JavaScript (JavaScript Object Notation)

LXC Contenedor Linux (Linux container)

ML Aprendizaje automático (Machine Learning)

MiTM Ataque de intermediario (Man in The Middle)

MB Megabyte

O Objetivo

OS Sistema operativo (Operating System)

PLICA Plataforma Integrada de Conciencia Cibersituacional

PID Identificador de proceso (Process ID)

RSTI Redes y Servicios de Telecomunicación e Internet

REST Transferencia de Estado Representacional (Representational State Transfer)

RD Requisitos de diseño

SSL Capa de puertos seguros (Secure Sockets Layer)

SSH Terminal Seguro (Secure Shell)

SIEM Gestión de información y eventos de seguridad (Security Information and Event Management)

SASL Capa de seguridad y autenticación simple (Simple Authentication and Security Layer)

TFG Trabajo de Fin de Grado

VM Máquina virtual (Virtual machine)

Capítulo 1

Introducción

Este capítulo es un preámbulo al proyecto en cuestión, en el que se va a poner de manifiesto la motivación que ha dado lugar a la realización de este, así como los objetivos que son planteados para su desarrollo. Adicionalmente, se incluye un apartado donde se muestra la estructura global del documento, con un breve resumen de cada capítulo.

1.1 MOTIVACIÓN

Desde principios de siglo la tecnología está evolucionando continuamente, apareciendo nuevos lenguajes de programación, estructuras de red y entornos, hasta el punto de que todos los procesos están siendo automatizados y llevados a servidores en la nube. Cada vez es más común la desaparición de estructuras monolíticas que no han avanzado y no se han adecuado a estos nuevos cambios que se han producido, debido a que estas dificultan su inclusión en nuevos sistemas y el crecimiento de sí mismas. Además, este tipo de entornos son más fácilmente vulnerados debido a la aparición de nuevos tipos de ataques y la sofisticación en los procesos para realizarlos.

Esta es la situación raíz de la que surge este trabajo de fin de grado, en el contexto del proyecto PLICA en el RSTI de la ETSIT, en colaboración con el programa de gobierno COINCIDENTE. En este se emplea un entorno cuyas tecnologías de virtualización y orquestación están desactualizadas, lo que complica su despliegue, su portabilidad a otros sistemas, su escalabilidad y su securización. Además, se trata de un entorno de baja resiliencia, el cual es necesario reiniciar por completo cada vez que se produce algún error o se necesita modificar algún archivo.

Dicha estructura monolítica cuenta con una serie de sensores que captan información de diversa índole, como puede ser el caso de datos de WiFi, redes móviles, etc. Estos datos son monitorizados y procesados en tiempo real por un servidor Kafka que tiene instancias individualizadas para cada uno de ellos y cuyo orquestador es ZooKeeper. Tras esto, se procesan los datos en tiempo real mediante Spark, utilizando algoritmos de aprendizaje automático, uno por cada uno de los sensores, dando como resultado la propia trama de datos con un parámetro adicional que indica la presencia o no de una posible anomalía en los mismos. Este resultado se almacena en una base de datos, concretamente en Elasticsearch, donde además mediante ElastAlert se correlan los datos, de modo que se puedan buscar anomalías relacionadas con las detectadas en otros sensores.

Finalmente, se aplican unas reglas ontológicas y el resultado se envía a un servidor Fuseki, donde se observan los sensores afectados y cuáles son los posibles ataques que está recibiendo. Adicionalmente, cuenta con un sistema de monitorización basado en Pandora FMS, con el que se gestiona todo el entorno.

Tras este análisis de su distribución, sus tecnologías y sus deficiencias, se plantea la necesidad de llevar dicha estructura monolítica a un nuevo entorno basado en nuevas técnicas de virtualización y orquestación. Con esto debe permitirse, manteniendo la misma funcionalidad y respetando las tecnologías empleadas, mejorar el rendimiento del sistema y contar con las características anteriormente mencionadas de escalabilidad, portabilidad, seguridad y resiliencia, entre otras.

1.2 OBJETIVOS

El objetivo de este TFG es el diseño y desarrollo de un entorno basado en microservicios Docker securizados para la migración de la actual estructura de virtualización monolítica que está siendo utilizada en el proyecto PLICA. Con esta idea, se crearán las imágenes de cada uno de los subsistemas mencionados en la sección anterior, ejecutándolos en contenedores individuales, con la configuración y elementos necesarios en cada caso.

Para abordar este objetivo, se han propuesto una serie de propósitos secundarios enfocados en diferentes fases que en conjunto darán como resultado lo que se persigue en la realización de este proyecto. Estos han sido identificados desde el más básico y esencial para el comienzo del desarrollo hasta los últimos requisitos y características que se quieren conseguir en el entorno en cuestión. Podemos agruparlos como se muestra a continuación:

Id	**Objetivos del TFG**
O1	Determinar los diferentes servicios a "dockerizar", estableciendo las dependencias, puertos y configuración necesarias para cada uno de ellos.
O2	Habilitar el despliegue de todos los servicios conjuntamente y contar con un fichero de variables de entorno donde se almacenen las de todos los servicios, facilitando así su portabilidad, cambio de versiones, redes, credenciales y localización de ficheros.
O3	Securizar las comunicaciones entre contenedores, de modo que los mensajes que se envían se encuentren encriptados ante posibles ataques y conceder acceso SSH mediante claves públicas/privadas.
O4	Permitir la escalabilidad de los sistemas, dando la opción de que se puedan crear dinámicamente nuevos contenedores de los servicios en caso de ser necesario.
O5	Reducir el tiempo de arranque y el espacio de almacenamiento, haciendo que el entorno sea lo más ligero posible.
O6	Aumentar la resiliencia, por lo que si un proceso falla no sea necesario reiniciar de cero todo el entorno, con la pérdida de productividad que esto conlleva.

Tabla 1. Objetivos planteados

Con estos objetivos en mente se plantearán la fase de diseño del proyecto en primera instancia, para continuar con el desarrollo del entorno y finalizar con la validación y resultados de dichos objetivos a través de pruebas, mostrando su alcance.

1.3 ESTRUCTURA DEL DOCUMENTO

En esta sección se aporta una visión de la estructura basada en capítulos con la que se ha establecido este documento, así como una breve descripción de estos. Esta estructura se puede ver a continuación:

Capítulo 1. Presenta la introducción al TFG. En este se explica cuál es la motivación que ha llevado a la realización de proyecto y los objetivos que se han planteado.

Capítulo 2. Muestra el estado del arte de las tecnologías y herramientas que se han empleado en el proyecto para dar un acercamiento inicial a estas.

Capítulo 3. Enseña la parte de diseño del entorno en cuestión. Se pondrán en común los requisitos y decisiones de diseño, la solución propuesta y las etapas a seguir.

Capítulo 4. Ofrece una visión completa del desarrollo global del entorno y de cada uno de los servicios a desplegar de manera individual.

Capítulo 5. Manifiesta los resultados alcanzados tras la finalización del desarrollo y su validación con pruebas de diferente índole.

Capítulo 6. Expone las conclusiones que se extraen de la realización del proyecto y las líneas futuras sobre las que se trabajaría.

Capítulo 2

Estado del arte

En este capítulo se resumen las diferentes tecnologías que se han utilizado para desarrollar este proyecto. Principalmente, se mostrarán las tecnologías de virtualización y orquestación para construir y desplegar el entorno, pero también se dará una visión sobre las tecnologías virtualizadas y otras herramientas que serán utilizadas.

2.1 HERRAMIENTAS DE VIRTUALIZACIÓN

El término virtualización en computación hace referencia al proceso de ejecutar una instancia virtual en una capa abstraída del hardware real. El sistema donde se ejecuta la herramienta de virtualización (*hipervisor*) es el anfitrión o *host* y las máquinas instaladas sobre el *hipervisor* son los invitados o *guest*, que perciben el entorno como si fuera suyo, accediendo a todos los recursos del sistema. En esta sección se van a presentar tres tecnologías de virtualización que o bien son utilizadas en la actual estructura monolítica del proyecto o bien van a ser utilizadas durante el desarrollo de este.

Figura 1. Herramientas de virtualización

2.1.1 VIRTUALBOX

Oracle VM VirtualBox es un software de código abierto de virtualización x86, bajo los términos de la Licencia Pública General de GNU, creado por la empresa de software Innotek y desarrollado actualmente por Oracle. Esta tecnología se instala en un sistema operativo anfitrión como una aplicación, permitiendo la creación de sistemas operativos invitados adicionales para ser cargados y ejecutados cada uno con su entorno virtual [1].

Actualmente puede ser utilizada en anfitriones con sistemas operativos Linux, Mac OS X, Windows XP y posteriores, Solaris y OpenSolaris, siendo capaz de virtualizar múltiples OS invitados bajo el sistema operativo anfitrión y pudiendo iniciarse, pausarse y detenerse independientemente dentro de su propia máquina virtual.

VirtualBox utiliza su propio formato para los contenedores de almacenamiento y es conocido como Virtual Disk (VDI). Además, admite también otros formatos de almacenamiento conocidos como VMDK, VHD y HDD [2], haciendo que este software de virtualización ponga mayor énfasis en la portabilidad.

Por otra parte, cuenta con un kit de herramientas adicional, *Guest Additions*. que habilita otras opciones para interactuar con el sistema operativo anfitrión, como las carpetas compartidas. Otras características importantes que ofrece son:

- Compatibilidad con todo tipo de hardware y múltiples resoluciones de pantalla.
- Sistema de instantáneas, pantalla remota y soporte USB.
- Agrupación y clonado de máquinas virtuales, hasta 32 CPUs.

2.1.2 DOCKER

Docker es una tecnología de código abierto basada en contenedores, que utilizan la virtualización a nivel de sistema operativo y ofrecen software para el despliegue de aplicaciones. Docker surge como código abierto en marzo de 2013 y utiliza una arquitectura cliente-servidor, donde el cliente habla con el demonio Docker, que construye, ejecuta y distribuye los contenedores. Ambos pueden ejecutarse en el mismo sistema, o en sistemas distintos y se comunican mediante API REST, a través de sockets UNIX o una interfaz [3].

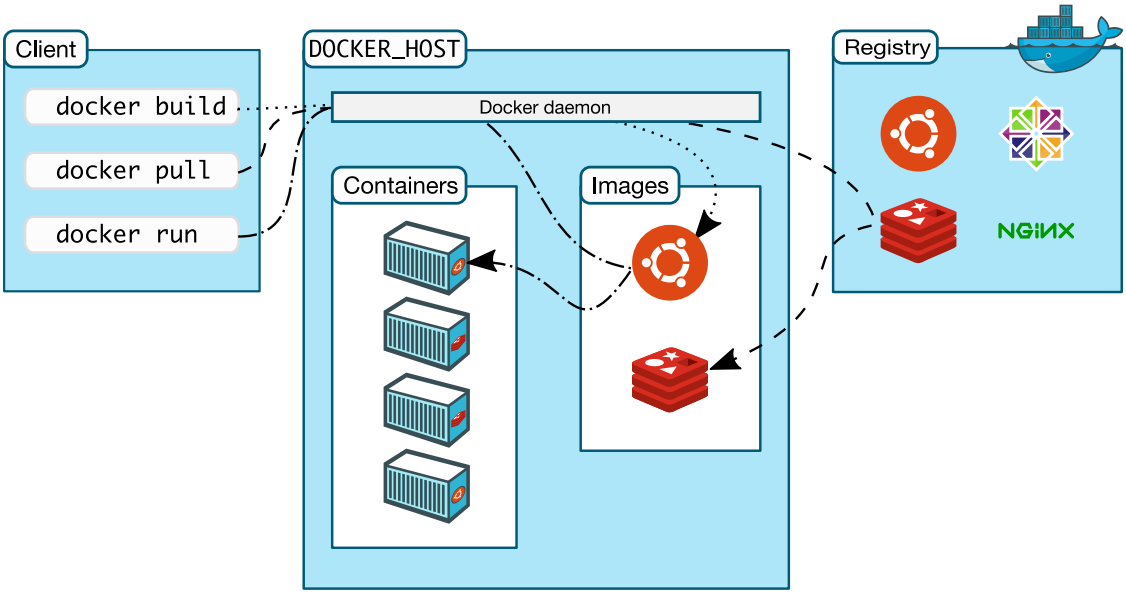

Figura 2. Arquitectura del ecosistema Docker [3]

Inicialmente, esta arquitectura estaba basada en LXC, formado por los *namespaces*, que envuelven un conjunto de recursos del sistema y los presentan a un proceso; y los *cgroups*, que gobiernan el aislamiento y el uso de los recursos del sistema, como la CPU y la memoria, para un grupo de procesos. Sin embargo, Docker terminó desarrollando su propia tecnología de contenedores, permitiendo que las aplicaciones se dividan en procesos individuales y facilitando el proceso de diseño y creación de estos.

Docker utiliza una serie de objetos como son imágenes, contenedores, redes y volúmenes, entre otros, que son la base de su funcionamiento. Los contenedores son instancias en ejecución de una imagen que contienen el software específico que está incluido dentro la imagen que este ejecuta. Estas imágenes son creadas a partir de unos archivos denominados *Dockerfile*, los cuales son la raíz de su construcción, ya que a través de ellos se consigue crear las imágenes con las que despliegan todos los entornos basados en contenedores.

Los *Dockerfile* se caracterizan por ser un documento de texto que contiene una serie de instrucciones que van a ser procesadas para crear diversas imágenes. En ellos, está descrito todo el software que encapsularán los contenedores. Un *Dockerfile* puede heredar de otros, de modo que en el mismo se herede de otra imagen incluyendo otros módulos.

```
FROM ubuntu:18.6

ARG SPARK_VERSION
ARG HADOOP_VERSION

LABEL org.label-schema.name="spark_master" \
      org.label-schema.description="Apache spark" \
      org.label-schema.version="${SPARK_VERSION}_${HADOOP_VERSION}" \
      org.label-schema.schema-version="1.0" \
      maintainer="acallejasz"

COPY master.sh template.sh /
COPY log4j.properties /spark/conf/

EXPOSE 8080 7077 6066

CMD ["/bin/bash", "/master.sh"]
```

Figura 3. Ejemplo de fichero: *Dockerfile*

Por su parte, una imagen de Docker es un archivo que contiene el código fuente, bibliotecas, dependencias, herramientas y otros archivos necesarios para que se ejecute una aplicación, siendo los contenedores los que ejecuten una de estas imágenes previamente compiladas. Se caracterizan por ser inmutables, es decir, una imagen una vez creada no puede ser modificada, por lo que se debe crear una nueva a partir de un *Dockerfile* con las nuevas características a añadir [4]. En la siguiente figura podemos ver como los contenedores que se crean son variados, con más o menos aplicaciones y sobre un mismo kernel.

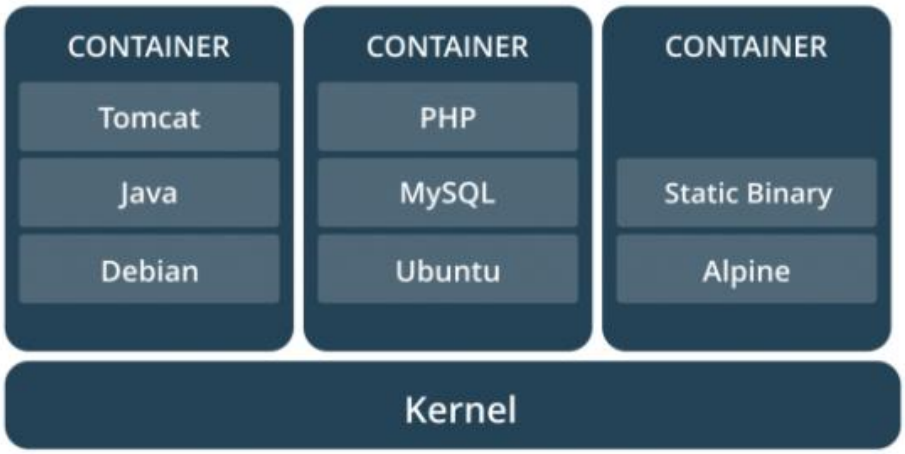

Figura 4. Contenedores desplegados sobre varias imágenes generadas con *Dockerfiles* [5]

Entre las principales características de esta tecnología se destaca que los recursos son aislados, los servicios restringidos y se otorga a los procesos la capacidad de tener una visión casi completamente privada del sistema operativo, con su propio identificador de espacio de proceso, estructura del sistema de archivos e interfaces de red. Además, también se permite:

- *Mejora de la portabilidad*. Los contenedores Docker se ejecutan sin modificaciones en cualquier escritorio, centro de datos y entorno en la nube.
- *Mayor ligereza*. Dado que sólo puede ejecutarse un proceso en cada contenedor, es posible construir una aplicación que pueda seguir funcionando mientras una de sus partes se retira para una actualización o reparación.
- *Creación automática de contenedores*. Docker puede construir automáticamente un contenedor basado en el código fuente de la aplicación.
- *Reutilización de contenedores*. Los existentes pueden utilizarse como imágenes base para construir otros nuevos.

2.1.3 VAGRANT

Vagrant es una herramienta de código abierto enfocada en proporcionar un flujo de trabajo consistente a través de múltiples sistemas operativos, manejando un conjunto heterogéneo de máquinas virtuales [6]. Desarrollada por HashiCop, esta permite la posibilidad de modificar las propiedades de la máquina virtual tales como la RAM, el número de CPUs, etc. También dispone de un sistema de *mirroring* con la maquina anfitriona, lo que permite ver directorios y modificar ficheros, siendo visible para ambas partes.

Esta tecnología también permite reducir el tiempo de configuración del entorno de desarrollo, aumentar la paridad de producción y pausar, suspender o borrar el estado de la máquina. Las máquinas virtuales que utiliza Vagrant se aprovisionan sobre VirtualBox, VMware, AWS o cualquier otro proveedor y pueden instalar y configurar automáticamente el software en la máquina virtual mediante archivos de terminal, Chef o Puppet, [6].

Los requisitos de la máquina y el software se escriben en un archivo de configuración llamado *Vagrantfile*, que indica los pasos necesario a ejecutar con el fin de crear una *box* para el desarrollo. Esto último es un formato y una extensión (.box) para los entornos y paquetes Vagrant que se copia en otra máquina para replicar el mismo entorno. A grandes rasgos, se trata de las imágenes de las máquinas virtuales, las cuales se pueden encontrar en la página *Bento* de Vagrant [7].

Vagrant también soporta la inclusión de *plugins* para conseguir funcionalidades extra, la configuración de la interfaz red, la penetración de sus sistemas en la nube y siempre asegura que el entorno tenga la misma configuración.

2.2 HERRAMIENTAS DE ORQUESTACIÓN

El término orquestación en computación hace referencia a la automatización de muchas tareas individuales para que funcionen juntas, por lo que se requiere que se conozcan todos los pasos que se llevan a cabo en cada proceso. El objetivo de la orquestación es agilizar y optimizar procesos frecuentes y repetitivos y eliminar así redundancias. En esta sección se van a presentar las dos tecnologías de orquestación que se van a proponer en la fase de diseño del entorno para su posterior comparativa y elección.

Figura 5. Herramientas de orquestación

2.2.1 DOCKER COMPOSE

Docker Compose se define como una herramienta de orquestación para resolver y ejecutar aplicaciones Docker multi-contenedor. Utiliza un fichero tipo *YAML* donde se configuran los servicios y puede desplegar y ejecutar contenedores con diferentes servicios a través de un solo comando [8]. Además, permite visualizar el estado de los servicios activos, así como reconstruirlos u obtener salidas con información de estos. También permite utilizar volúmenes, interfaces de red, variables de entorno, documentar y configurar las dependencias de los servicios y aislar los entornos unos de otros, siempre todo ello dentro del mismo *host* y por tanto, corriendo localmente.

Los principales casos de uso de esta tecnología, de acuerdo con su especificación, se pueden agrupar como se muestra a continuación [8]:

- *Entornos de desarrollo*

Durante el desarrollo de software, la herramienta de línea de comandos que ofrece Compose permite ser utilizada para crear un entorno aislado e interactuar con él. Admite una forma de documentar y configurar todas las dependencias de servicios de la aplicación como pueden ser datos, cachés o colas. Además, permite crear e iniciar uno o más contenedores para cada dependencia con un solo comando: *docker-compose up*. De este modo, Compose permite reducir toda la orquestación a un único archivo simple y unos pocos comandos, haciéndolo ideal para esta tarea.

- *Entornos de prueba automatizados*

Una parte importante de cualquier proceso de despliegue o integración continua es el conjunto de pruebas automatizadas. Las pruebas automatizadas requieren de un entorno en el que ejecutarlas y esta herramienta ofrece una forma sencilla de crearlos y destruirlos, con pocos comandos para el conjunto de pruebas.

- *Despliegues de un solo host*

Compose se centra tradicionalmente en flujos de trabajo de desarrollo y pruebas, pero cada vez está más orientado a producción. Dado que este crea redes locales dentro de un mismo sistema está orientado a este tipo de despliegues.

Teniendo en cuenta que todo se basa en un único archivo tipo *.yml*, se muestra a continuación un ejemplo en la siguiente figura:

```yaml
version: '3.9'
services:
  worker:
    image: spark-worker:${SPARK_VERSION}-hadoop${HADOOP_VERSION}
    environment:
      SPARK_MASTER: ${SPARK_MASTER}
      SPARK_WORKER_WEBUI_PORT: ${SPARK_WORKER_WEBUI_PORT}
      SPARK_WORKER_LOG: ${SPARK_WORKER_LOG}
      SPARK_WORKER_CORES: ${SPARK_WORKER_CORES}
      SPARK_WORKER_MEMORY: ${SPARK_WORKER_MEMORY}
    ports:
      - "22"
    networks:
      plica_SPARK:
    volumes:
      - ../base/PLICA_V6/:/app
networks:
  plica_SPARK:
    external: true
```

Figura 6. Ejemplo de fichero: *docker-compose.yml*

Un resumen de las principales características de esta tecnología se puede ver a continuación:

- *Configuración rápida y sencilla*. Con los scripts *YAML*.
- *Alta productividad*. Reduce el tiempo que se tarda en realizar las tareas y solo recrea aquellos contenedores en los que se ha producido algún cambio.
- *Preservación de los datos de los volúmenes*. Accediendo a ellos cuando se arranca el contenedor que los incluye y cuando este es apagado.
- *Despliegue en un solo host*. Puede ejecutar todo en una sola pieza de hardware.
- *Intercambio de variables de entorno*. A través de un único fichero *.env* se pueden asignar valores a todos los servicios a desplegar y utilizarlas para personalizar los entornos y usuarios.
- *Seguridad*. Al estar los contenedores aislados unos de otros se reducen amenazas.

2.2.2 KUBERNETES

Kubernetes, también conocido como "*k8s*" o "*kube*", se define como una plataforma portátil, extensible y de código abierto para administrar cargas de trabajo y servicios en contenedores, que facilita tanto la configuración declarativa como la automatización [9]. Fue desarrollado por ingenieros de Google antes de ser de código abierto, ya que en sus inicios fue una plataforma de orquestación de contenedores utilizada internamente en Google. Esta plataforma es compatible con Docker y, aunque su uso es más complejo, aporta mayores funcionalidades y está más extendido en la comunidad, ya que es utilizado por la mayor parte de los proveedores *cloud* en la actualidad.

Entrando en detalle, se trata de un orquestador de contenedores que agrupa múltiples *host* juntos en un clúster, haciendo que sean tolerantes a fallos, escalables, optimizables y permitiendo el acceso desde la red. Adicionalmente, incluye el despliegue de contenedores, las actualizaciones, el descubrimiento de servicios, el aprovisionamiento de almacenamiento, el equilibrio de carga y la supervisión de la 'salud' del contenedor, entre otros.

Este ecosistema permite a las organizaciones ofrecer una plataforma como servicio (PaaS) de alta productividad que aborda múltiples tareas y entre sus principales componentes se pueden destacar los siguientes [10]:

- *Clústeres y nodos*

Los clústeres son los bloques de construcción de la arquitectura de Kubernetes, que están formados por nodos o conjunto de máquinas de trabajo, donde cada uno representa un único *host* de computación virtual o físico. Todo clúster tiene al menos un nodo trabajador que despliega, ejecuta y gestiona aplicaciones en contenedores y un nodo maestro que controla y supervisa los nodos trabajadores.

El nodo maestro tiene un servicio de programación que automatiza cuándo y dónde se despliegan los contenedores en función de los requisitos establecidos por el desarrollador y la capacidad informática disponible. Cada nodo trabajador incluye la herramienta de línea de comandos que se utiliza para gestionar los contenedores, conocida como *kubectl*, la cual utiliza un fichero *.yml* a través del cual se hace una llamada a la API de Kubernetes [11] y un agente de software llamado *kubelet* que recibe y ejecuta las órdenes del nodo maestro y se asegura que los contenedores están corriendo en un *Pod*, aunque solo aquellos que han sido creados con Kubernetes.

- *Pod y despliegues*

Los *pods* son grupos de contenedores que comparten los mismos recursos y red. También son la unidad de escalabilidad en Kubernetes, de modo que si un contenedor que está en un *pod* recibe más tráfico del que puede manejar, se replicará el *pod* a otros nodos en el clúster.

El despliegue se encarga de controlar la creación y estado de la aplicación y la mantiene en funcionamiento. Además, especifica el número de réplicas de un *pod* que deben ejecutarse en el clúster, dando el caso de que si ocurre un fallo en uno de los *pods*, el despliegue se hace cargo de crear uno nuevo.

A continuación, se muestra una imagen que representa un clúster de Kubernetes con todos sus componentes interconectados:

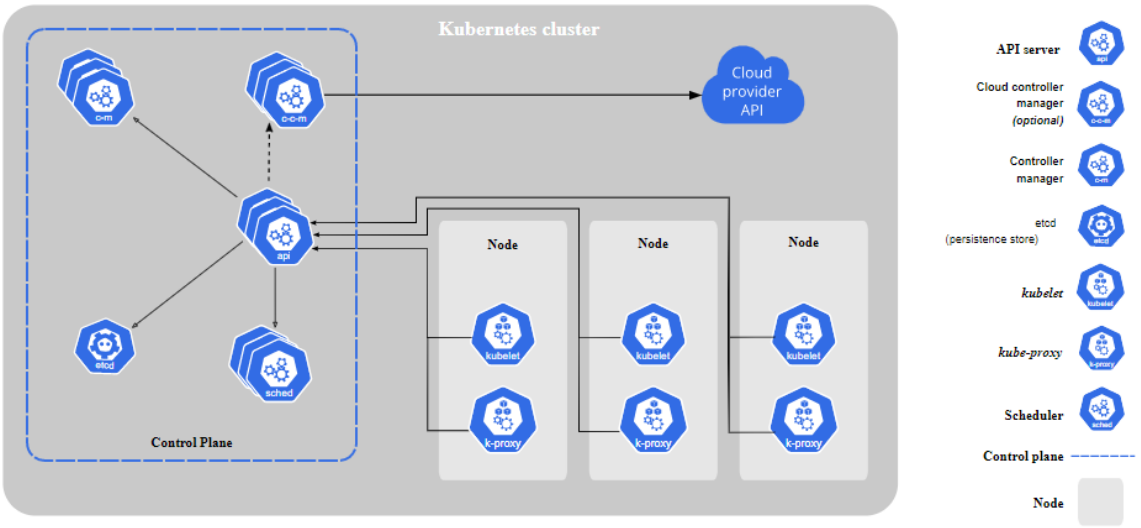

Figura 7. Clúster de Kubernetes y sus componentes [10]

Se detalla ahora un resumen de las principales características de esta tecnología:

- *Despliegues y escalabilidad automatizados*. Despliega progresivamente los cambios en su aplicación o su configuración, al tiempo que supervisa el estado de la aplicación.
- *Descubrimiento y balanceo de carga*. Mediante los *pods*, con sus propias direcciones IP y un único DNS, consigue equilibrar la carga entre ellos.
- *Resiliencia*. Reinicia los contenedores que fallan y reemplaza y reprograma los contenedores cuando los nodos mueren.
- *Seguridad*. Actualiza la configuración y credenciales sin reconstruir la imagen y sin exponerlos en la pila.

2.3 TECNOLOGÍAS

En esta sección se van a presentar las siete tecnologías de las que se hace uso en el proyecto PLICA y que van a ser virtualizadas durante el desarrollo.

Figura 8. Tecnologías PLICA

2.3.1 APACHE ZOOKEEPER

ZooKeeper es un servicio de código abierto para aplicaciones distribuidas y centralizado para mantener la información de configuración, nombrar, proporcionar sincronización distribuida y servicios grupales [12]. Este sirve y es utilizado como orquestador para múltiples plataformas, como puede ser Apache Kafka, utiliza un modelo de datos basado en la estructura de árbol de directorios de los sistemas de archivos y se ejecuta en Java.

En cuanto a los objetivos de diseño que se tratan de conseguir a través de esta tecnología y sus principales características se destacan [13]:

- *Simplicidad*

ZooKeeper es sencillo ya que permite coordinación a través de un espacio de nombres jerárquico compartido con un registro de datos en *znodes*. Además, los datos de ZooKeeper se guardan en la memoria, lo que significa que puede alcanzar cifras de alto rendimiento y baja latencia.

- *Replicación*

Este servicio está pensado para ser replicado en un conjunto de *hosts* llamados *ensemble*. Los servidores que lo componen deben conocerse entre sí y mantener una imagen de estado en memoria, junto con un registro de transacciones e instantáneas en un almacén persistente. Zookeeper estará disponible mientras que la mayoría de los servidores lo estén.

- *Ordenación y rapidez*

En cada actualización se utiliza un número que refleja el orden de todas las transacciones. Las operaciones posteriores pueden utilizar dicho orden para implementar abstracciones de nivel superior, como primitivas de sincronización. Adicionalmente, es especialmente rápido en cargas de trabajo de lectura, ya que se ejecuta en muchas máquinas, siendo su rendimiento mejor cuando las lecturas son más comunes que las escrituras.

Haciendo hincapié en su estructura se puede ver como los clientes se conectan a un único servidor. El cliente mantiene una conexión TCP a través de la cual envía solicitudes, obtiene respuestas y recibe eventos de vigilancia. Si la conexión TCP con el servidor se rompe, el cliente se conectará a otro servidor.

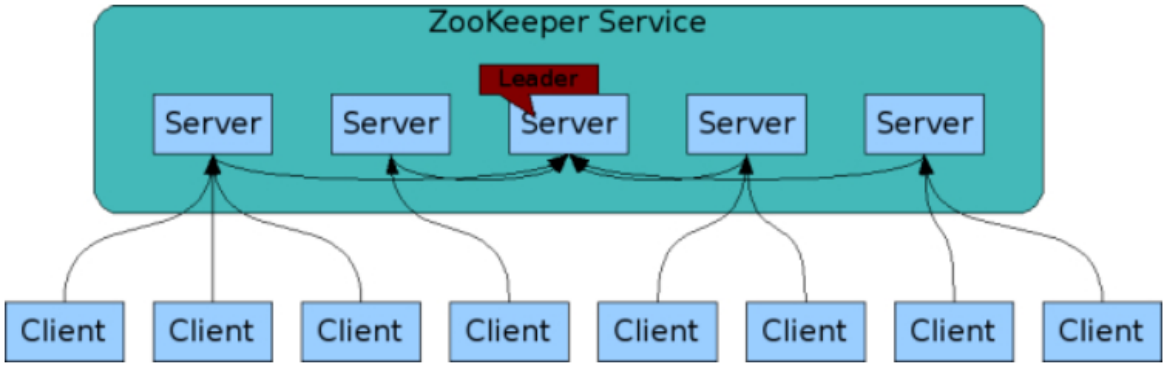

Figura 9. Estructura de Apache ZooKeeper [13]

2.3.2 APACHE KAFKA

Apache Kafka es una plataforma de transmisión de eventos distribuida, de código abierto y utilizada para canalizaciones de datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica [14]. Básicamente, la trasmisión de eventos es capturar datos en tiempo real desde diferentes fuentes de eventos en forma de flujos, almacenarlos para su posterior recuperación y manipularlos y procesarlos.

Las principales capacidades que aporta esta tecnología se pueden resumir en [15]:

- *Publicar y suscribir flujos de eventos*. Incluyendo la importación/exportación continua de sus datos desde otros sistemas.
- *Almacenar flujos de eventos de forma duradera y fiable*. Permitiendo de este modo que puedan ser utilizados en cualquier momento.
- *Procesar flujos de eventos*. Ya sea a medida que se producen o de forma retrospectiva.

Todas estas funcionalidades se proporcionan de forma distribuida, escalable, tolerante a fallos y segura. Este sistema puede desplegarse en hardware, máquinas virtuales, contenedores y en la nube. Para conseguir todas estas características, Kafka utiliza una estructura especial y numerosos componentes, los cuales podemos ver a continuación:

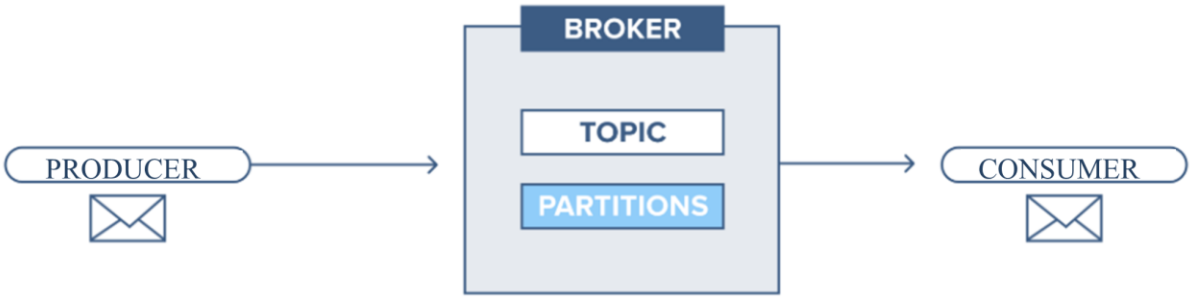

Figura 10. Estructura y componentes de Apache Kafka

En primer lugar, Kafka cuenta con los llamados *broker*, que son cada uno de los servidores de los que dispone la aplicación como nodos de un clúster, donde cada uno mantiene un conjunto de particiones (primaria y secundaria) y réplicas de cada uno de los *topic*, siendo todo gestionado mediante ZooKeeper.

Los *topics* son categorías sobre las que se agrupan un conjunto concreto de mensajes, los cuales son publicados en el clúster y permanecen en este hasta que haya pasado un periodo de retención configurable. Además, se dividen en un número de particiones, que contienen registros en una secuencia inalterable, donde a cada uno se le asigna e identifica por su única secuencia.

Para el envío y recepción de mensajes se utilizan los conocidos como *productores* y *consumidores*, respectivamente, los cuales pueden ser cualquier tipo de sistema y que se conectan a uno o más *brokers* para que estos, en conjunción con Zookeeper, se encarguen de orquestar, gestionar y traspasar todos los mensajes de un *topic*. De este modo, Kafka consigue toda su funcionalidad.

2.3.3 APACHE SPARK

Apache Spark es un sistema de código abierto que utiliza el análisis unificado para el procesamiento de datos a gran escala y el aprendizaje automático. Proporciona APIs de alto nivel en Java, Scala, Python y un motor optimizado que soporta gráficos. También es compatible con un amplio conjunto de herramientas, como *Spark SQL* para el procesamiento de datos estructurados, *SQL* para el aprendizaje automático, *GraphX* para el procesamiento de gráficos y *Structured Streaming* para el cálculo incremental y el procesamiento de flujos [16].

Las aplicaciones que se ejecutan mediante esta tecnología lo hacen como un conjunto independiente de elementos en un clúster, realizándolo de manera coordinada mediante lo que se conoce como el *SparkContext* o programa controlador. De este modo, este se conecta a un nodo gestor del clúster para asignar los recursos a las aplicaciones y enviar tareas a los ejecutores, el cual debe estar preferiblemente en la misma red de área local que los trabajadores. Estos son una serie de nodos trabajadores que se crean, los *workers*, donde se localizan y ejecutan los diferentes procesos y tareas de las aplicaciones [17].

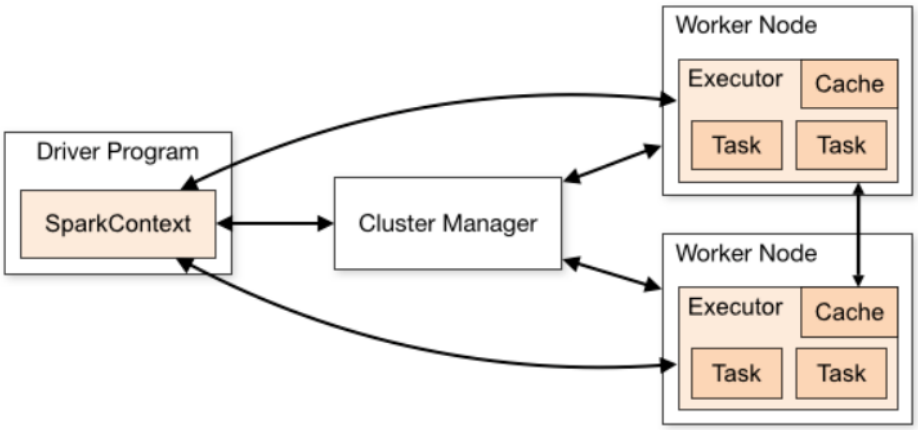

Figura 11. Estructura y componentes de Apache Spark [17]

Cada aplicación tiene sus propios procesos ejecutores, que permanecen activos durante toda la aplicación y ejecutan tareas en múltiples hilos. Esto tiene la ventaja de aislar las aplicaciones entre sí, pero también significa que los datos no pueden ser compartidos entre diferentes aplicaciones Spark sin escribirlos en un sistema de almacenamiento externo.

Los gestores de clústeres que están soportados son:

- *Standalone*. Gestor sencillo, incluido con Spark, que facilita la configuración de los clústeres.
- *Apache Mesos*. Gestor de clústeres que también puede ejecutar Hadoop y aplicaciones de servicio.
- *Hadoop Yarn*. Gestor de recursos de Hadoop 2.
- *Kubernetes*. Sistema de código abierto para automatizar el despliegue, el escalado y la gestión de aplicaciones en contenedores.

2.3.4 ELASTICSEARCH

Elasticsearch es un motor de analítica y análisis distribuido de código abierto basado en Java para todos los tipos de datos, incluidos textuales, numéricos, estructurados y desestructurados. A grandes rasgos, es una base de datos que permite almacenar, buscar y analizar enormes volúmenes de datos de forma rápida y casi en tiempo real y es comúnmente presentada bajo las siglas de ELK, de Elasticsearch, Logstash y Kibana, que se usan en conjunto para añadirle una interfaz gráfica y procesamiento de datos [18].

Esta tecnología es capaz de lograr respuestas de búsqueda rápidas, gracias a su sistema de índices y consultas. Entrando en detalle, un índice de Elasticsearch es una colección de documentos con características similares, siendo su entidad de máximo nivel. Este se identifica con un nombre y sobre él se realizan las operaciones de indexación, búsqueda, actualización y eliminación de los documentos que contiene.

Adicionalmente, cuenta con amplias APIs REST para almacenar y buscar los datos. En esencia, se puede ver como un servidor que puede procesar solicitudes JSON y devolver datos JSON. Admite numerosos lenguajes de programación como Java, JavaScript, Python o Ruby, entre otros.

Entre los principales casos de uso de Elasticsearch cabe destacar la búsqueda en aplicaciones y sitios web, métricas de infraestructura, supervisión de contenedores y monitorización del rendimiento de las aplicaciones. En cuanto a sus principales características se destacan su rapidez en las consultas, su distribución en contenedores, su simplicidad a la hora en introducir, visualizar y reportar datos y su gran variedad de características.

2.3.5 ELASTALERT

ElastAlert es una tecnología para alertar a cerca de anomalías u otros patrones de interés, sobre los datos en Elasticsearch. Se creó como una herramienta complementaria para alertar sobre las incoherencias en los datos almacenados casi en tiempo real y de manera fiable, altamente modular y fácil de instalar y configurar [19].

Funciona en combinación con Elasticsearch mediante dos tipos de componentes conocidos como reglas y alertas, definidas por el usuario en un fichero de configuración en función del tipo de aplicación y de acuerdo con ciertos patrones. Así, se consulta periódicamente y los datos se pasan al tipo de regla, que determina cuándo se encuentra una coincidencia. Cuando se produce una coincidencia, se dan a una o más alertas, que toman medidas basadas en la coincidencia y almacenan su estado en Elasticsearch.

Existen diferentes tipos de reglas para alertas, como por ejemplo de tiempo y frecuencia, límites, cambios específicos o coincidencias con elementos de una lista negra. Asimismo, estas alertas tienen soporte con un grupo variado de aplicaciones como correo electrónico, Telegram y GoogleChat. Otras características que aporta este sistema son paneles de control con Kibana, informes periódicos de alertas o interceptar y mejorar los datos coincidentes con las reglas.

2.3.6 PANDORA FMS

Pandora FMS es un software de monitorización y código abierto que se encarga de recoger datos de cualquier tipo de sistema y generar alertas en base a estos, proporcionando informes, gráficas y mapas del entorno a monitorizar. La información que se monitoriza es recopilada y guardada para su representación visual y las posteriores acciones que se requieran. Además, está disponible en múltiples sistemas operativos, aunque se recomienda su uso en Linux [20].

Los principales componentes que forman esta tecnología son:

- *Servidores*. Se encargan de recoger y procesar los datos. Son conocidos como agentes.
- *Base de datos*. Donde se almacenan la información recogida y la configuración.
- *Consola*. Interfaz web que se encarga de mostrar los datos e interaccionar con el usuario.

Por otra parte, los elementos sobre los que se compone y algunas de sus funcionalidades se pueden resumir en: los agentes, que contienen los módulos para la monitorización, que son de donde se extrae la información del entorno; los eventos, que es todo lo que ocurre en el sistema; las alertas, como reacción a algún tipo de evento y la visualización de datos, que permite su observación con un amplio grado de posibilidades.

2.3.7 APACHE JENA FUSEKI

El servidor Fuseki es un servidor SPARQL que puede ejecutarse como un servicio del sistema operativo, como una aplicación web Java (archivo WAR) y como un servidor independiente [21]. Esta tecnología puede ser vista, bien como un sistema de aplicación web junto con una interfaz de usuario para la administración, o bien como un servidor para un gran despliegue, como podría ser con Docker.

Siempre utiliza el mismo motor de protocolo y el mismo formato de archivo de configuración, específico de este sistema. Fuseki utiliza los protocolos SPARQL 1.1 de consulta y actualización y el protocolo SPARQL Graph Store. Además, está integrado con TDB, un componente de Apache Jena para el almacenamiento y la gestión de peticiones de manera robusta.

2.4 OTRAS HERRAMIENTAS

En esta sección se van a presentar otras herramientas de menor relevancia que han sido utilizadas durante el desarrollo de este proyecto para tareas de diversa índole.

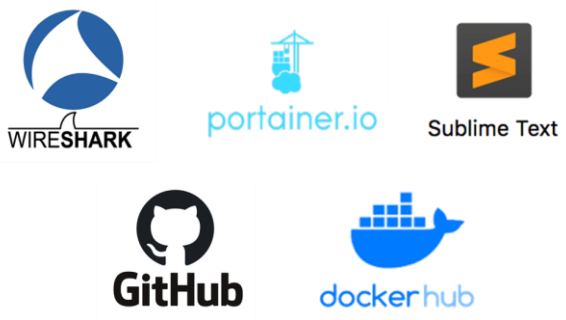

Figura 12. Otras herramientas

2.4.1 PORTAINER

Portainer es una herramienta de código abierto para gestionar aplicaciones en contenedores de Kubernetes, Docker, Docker Swarm y Azure ACI. Se encarga de eliminar la complejidad asociada a los orquestadores, pudiendo utilizarse para desplegar y gestionar aplicaciones, observar el comportamiento de los contenedores y proporcionar la seguridad necesaria para desplegar contenedores de forma generalizada, con una interfaz gráfica simple e intuitiva que permite realizar todas estas tareas [22].

2.4.2 WIRESHARK

Wireshark es una herramienta de código abierto para el análisis de protocolos y paquetes de red, disponible en UNIX y Windows, que permite una visualización altamente detallada de los datos de los paquetes capturados. De este modo, habilita la solución de problemas en la red, da la posibilidad de hacer exámenes de seguridad sobre los paquetes, evitando posibles ataques, verifica las aplicaciones en la red y facilita la depuración de protocolos [23].

Entre sus principales características está la captura datos en tiempo real, la importación de paquetes desde archivos de texto, la opción de guardar los datos de los paquetes capturados, el filtrado y búsqueda de paquetes según varios criterios y la creación de estadísticas a partir de los datos capturados.

2.4.3 SUBLIME TEXT

Sublime Text es un editor de código fuente multiplataforma, de código abierto y de uso compartido, con una interfaz de programación de aplicaciones de Python. Soporta muchos lenguajes de programación y permite a los usuarios añadir otras funciones que normalmente, son creadas por la comunidad y mantenidas bajo licencias de software libre [24].

Da la posibilidad de una navegación rápida a archivos o líneas, habilita la invocación rápida de comandos mediante el teclado, la edición simultánea y una amplia personalización a través de archivos de configuración JSON.

2.4.4 GITHUB

GitHub es un herramienta para almacenamiento online, que ofrece sus servicios básicos de manera gratuita y es utilizada para el desarrollo de software y el control de versiones mediante Git. Ofrece la posibilidad de controlar versiones distribuidas y gestión de código fuente, proporcionando control de acceso y varias características como el seguimiento de errores, la gestión de tareas y la integración continua.

2.4.5 DOCKER HUB

Docker Hub es un repositorio proporcionado por Docker para encontrar, subir y compartir imágenes de contenedores. Es el mayor repositorio del mundo de imágenes de contenedores, permitiendo el acceso de forma gratuita, la formación de equipos y organizaciones, la descarga de imágenes oficiales de calidad y la compilación automática de imágenes de contenedores desde GitHub [25].

Capítulo 3

Diseño del entorno

En este capítulo se presenta la parte de diseño del entorno en cuestión. Se pondrán en común los requisitos y decisiones de diseño, donde se logrará observar la estructura monolítica de PLICA, la solución propuesta de acuerdo con los objetivos y requisitos del proyecto y las etapas a seguir para conseguirlo.

3.1 REQUISITOS Y DECISIONES DE DISEÑO

Como se ha visto en el primer capítulo, la arquitectura actual del proyecto PLICA se encuentra en un estado monolítico, lo que hace de este un sistema rígido y difícil de adaptar debido a que sus tecnologías de virtualización y orquestación están desactualizadas. Entrando en detalle, este entorno está actualmente desarrollado como una única entidad global que se virtualiza y arranca mediante las herramientas de VirtualBox y Vagrant, lo que complica su despliegue, su portabilidad a otros sistemas, su escalabilidad y su securización. Además, se trata de un entorno de baja resiliencia, el cual es necesario reiniciar por completo cada vez que se produce algún error o se necesita modificar algún archivo.

El entorno que se acaba de mencionar se puede observar en la siguiente figura, donde se ven cada una de las tecnologías empleadas y la conexión entre ellas:

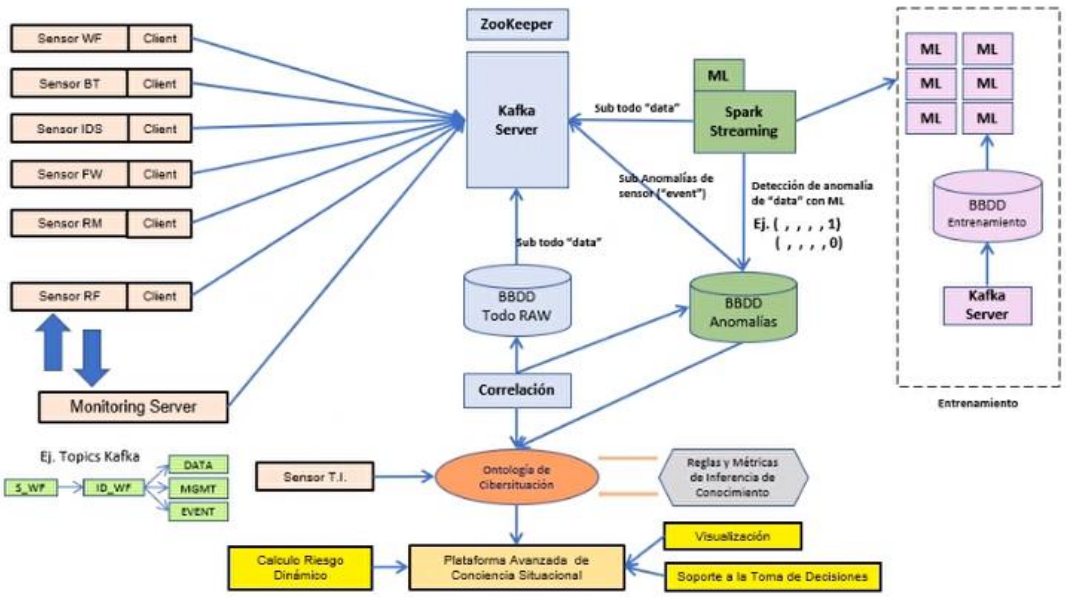

Figura 13. Arquitectura monolítica del proyecto PLICA

Como se puede percibir, a la entrada cuenta con una serie de sensores que captan información y son monitorizados y procesados por un servidor Kafka con *topics* para cada caso y orquestados por ZooKeeper. Tras esto, se procesan los datos en tiempo real mediante Spark, con algoritmos aprendizaje automático y el resultado, donde se han podido detectar posibles anomalías, se almacena con Elasticsearch. Además, estas anomalías se correlan con unas reglas mediante ElastAlert, tras lo que se aplica una ontología y se envía a una consola de visualización, donde se observan los sensores afectados y cuáles son los posibles ataques que está recibiendo.

Visto desde el contexto de la ciberseguridad, todo el conjunto puede verse como un SIEM (Gestión de Información y Eventos de seguridad). Es un sistema que recolecta los eventos de múltiples fuentes, identifica la información analizando su estructura y la normaliza y correla, detectando patrones y posibles anomalías. Sirven de apoyo a la monitorización proporcionando detección y notificación de incidentes de seguridad y facilita su trazabilidad.

Una vez se ha estudiado como está diseñado actualmente el entorno, se presentan una serie de requisitos y requerimientos recogidos en la siguiente tabla:

Id	**Requisitos**
RD1	El entorno debe de poder ejecutarse en cualquier tipo de sistema, tanto a nivel de red local como estar habilitado para despliegues en la nube y ser compatible con los sensores utilizados en el proyecto. Sin embargo, se requiere que sea diseñado para un despliegue completo a nivel local, sin conexión a internet
RD2	Los servicios y tecnologías deben virtualizarse independientemente y permitir su posterior despliegue de manera conjunta, para conseguir mayor resiliencia.
RD3	El sistema a diseñar debe estar securizado para evitar posibles intrusiones y accesos no autorizados y permitir de manera simple su monitorización.
RD4	Debe permitir la escalabilidad de los sistemas, dando la opción de que se puedan crear dinámicamente nuevas instancias de los servicios en caso de ser necesario.
RD5	Ha de reducir el tiempo de arranque y el espacio de almacenamiento, haciendo que el entorno sea lo más ligero posible.

Tabla 2. Requisitos de diseño del nuevo entorno

A través del estudio presentado en el segundo capítulo sobre diferentes tecnologías de virtualización y orquestación, se ha de tomar la decisión de cuales de ellas son las mejores para lograr cumplir con los requisitos. En la siguiente figura, se muestra una comparativa a nivel estructural entre un contenedor Docker y una máquina virtual.

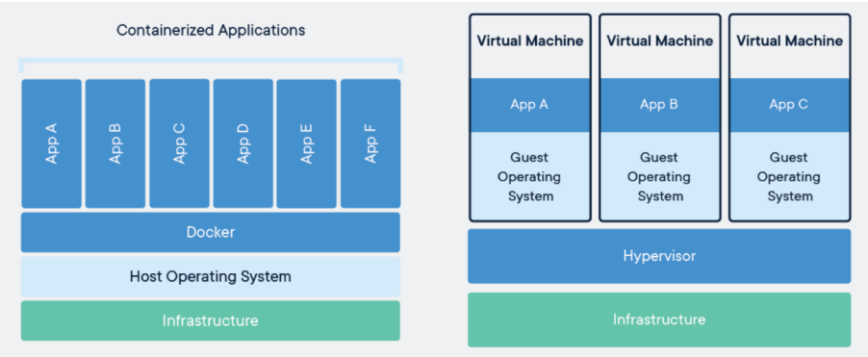

Figura 14. Estructura de un contenedor Docker y una máquina virtual [26]

Atendiendo a esta, podemos extraer las diferencias de Docker con respecto a las máquinas virtuales. Mientras que estas últimas son una abstracción del hardware del servidor donde están construidas y deben incluir una copia completa de un sistema operativo, aplicaciones, ficheros binarios y librerías, los contenedores son una abstracción en la capa de aplicación, que empaqueta código y dependencias juntas, haciendo que si se crean varios iguales sólo uno de ellos ocupe espacio en el servidor. Todo esto supone que las VM lleguen a alcanzar del orden de GB en almacenamiento haciendo a Docker ventajoso a estas, debido a que los contenedores son mucho más manejables, a la gran diferencia de tamaño, al tiempo de arranque y a la escalabilidad que aporta Docker. Además, la tecnología de contenedores está orientada y facilita la automatización, la integración y desarrollo continuos.

De manera similar, se puede comparar el uso de Vagrant con Docker. Dado que esta primera sirve para gestionar VMs, aunque permite una gestión más fácil, mayor escalabilidad y otras muchas características, el hecho de que funcione con VMs le genera numerosas limitaciones. A continuación se muestra una tabla comparativa entre ambas tecnologías:

Características	**Docker**	**Vagrant**
Tipo de virtualización	Contenedores	Máquina virtual
Nivel de aislamiento	Alto	Muy alto
Tiempo de creación	Inferior a 10 minutos	Superior a 10 minutos
Tamaño del despliegue	Del orden de MB	Del orden de GB
Tiempo de arranque	Segundos	Minutos
Impacto en el sistema	Muy bajo	Alto
Disponibilidad simultánea	Superior a 50	Inferior a 10
Garantiza recursos en OS	No	Si

Tabla 3. Comparativa Docker vs Vagrant

Teniendo en cuenta los requisitos *RD1*, *RD2*, *RD4* y *RD5*, se toma la decisión de que la mejor tecnología de virtualización para el diseño y desarrollo del entono en cuestión es Docker y por tanto se hará uso de esta herramienta para cumplir con las especificaciones.

En cuanto a las tecnologías de orquestación para contenedores Docker, hay que tomar una decisión entre Docker Compose y Kubernetes. La gestión de Kubernetes requiere de una mayor especialización, es menos sencillo de configurar y desplegar que Compose y es frecuente la necesidad de uso de herramientas adicionales y practicas más complejas para gestionar completamente el acceso, la identidad, la gobernanza y la seguridad. Además, Compose está orientado a un despliegue completamente local en un único *host*, el cual es uno de los requerimientos de diseño *(RD1)*, sin tener porque ser en el mismo *host*, pero sí a nivel local y permite escalar de manera sencilla todos los contenedores, así como securizarlos y utilizar variables de entorno y volúmenes. Sin embargo, para despliegues en la nube es altamente recomendable el uso de Kubernetes, ya que es capaz de autobalancear la carga de trabajo entre sus *pods* y está mejor preparado para despliegues de esta naturaleza.

Como conclusión y dada la naturaleza de despliegue local del entorno, se toma la decisión de utilizar Docker Compose para el orquestado de los contenedores de los diferentes servicios. Sin embargo, en casos futuros que necesiten despliegues en la nube, se toma la resolución de hacer uso de la herramienta de Kubernetes.

3.2 SOLUCIÓN PROPUESTA

Como se menciona en la sección anterior, se van a utilizar las tecnologías virtualización y orquestación de Docker y Docker Compose, por lo que se propone un entorno basado en contenedores, en el que cada uno de los servicios se despliegue individualmente para realizar posteriormente un despliegue conjunto de todas ellas, permitiendo su interconexión, tanto en un mismo entorno como en entornos distribuidos.

De este modo, se virtualizarán de manera independiente Kafka, ZooKeeper, Spark, Elasticsearch, ElastAlert, Fuseki y Pandora FMS mediante imágenes Docker, utilizando volúmenes para guardar los datos de manera local y tener persistencia. Dado que Kafka y ZooKeeper funcionan conjuntamente se conectarán a la misma red, al igual que ocurre con los servicios de Elasticsearch y Kibana. Por otra parte, el servicio de Spark se dividirá en varios microservicios formando un clúster, donde cada uno de sus nodos serán virtualizados de manera independiente en distintos contenedores. En cuanto a los servicios de ElastAlert y Fuseki, ambos tendrán una red propia exclusiva para ellos. Pandora FMS, por su parte, irá virtualizado dentro de todos estos servicios anteriormente mencionados. Atendiendo a la configuración de red, se reservará la primera dirección de cada una para hacer de *gateway* de cada una de ellas y todas utilizarán el mismo *bridge* para interconectarse unas con otras, dentro de la red que proporciona Docker a nivel local del equipo donde se está ejecutando.

Al utilizar Docker se deberá definir un *Dockerfile* para cada uno de los servicios descritos. Se necesita una imagen base de la que parten todas las imágenes de los servicios y sobre la que se construirán los contenedores. Para todos ellos se ha decidido utilizar la imagen *alpine:3.10*. La selección de esta se debe a que es la imagen más ligera soportada por Docker, ocupando tan solo 5.6 MB. Además, contiene todo lo necesario como base para desplegar contenedores de esas tecnologías, ocupando el menor espacio posible. Se puede encontrar en Docker Hub en la dirección especificada en [27], de donde se puede descargar.

En cuanto a la orquestación, se propone la realización de varios *docker-compose.yml* para el despliegue de aquellos servicios que se encuentran en el mismo direccionamiento red, lo que permita hacer pruebas funcionales de los microservicios a menor escala y ejecutarlos de manera distribuida y la realización de otro global, que permita desplegar el entorno al completo. Además, se realizará por cada uno de ellos un *.env*, en el cual se almacenará toda la información relevante a versiones de los servicios y otras variables de entorno como usuarios, contraseñas, direccionamiento y elementos específicos para cada microservicio. Se ha tomado esta decisión como solución final, ya que permite gestionar todas las variables de todo el entorno desde un único fichero, facilitando la automatización y simplificando su portabilidad.

A continuación, se muestra una figura en la que se pueden observar las diferentes tecnologías contenedorizadas, así como las relaciones entre ellas, sus direcciones y puertos dentro de una red para un despliegue, bien con un único *docker-compose.yml* en un mismo entorno o bien cada subsistema en entornos distribuidos, rompiendo la arquitectura monolítica de la que se partía inicialmente:

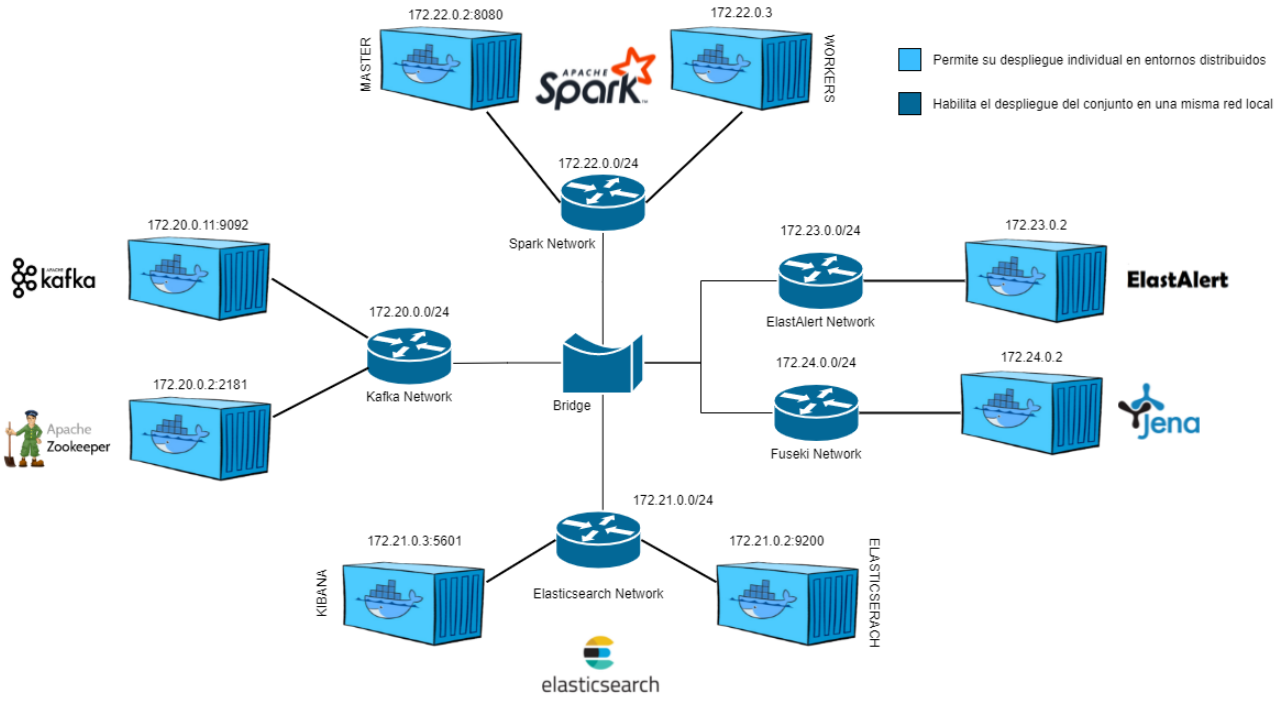

Figura 15. Solución de diseño propuesta para el entorno de PLICA

3.3 ETAPAS DE DISEÑO

Una vez propuesta la solución de diseño que se va a desarrollar en el siguiente capítulo, se establecen una serie de etapas en las que se divide dicho proceso. Para ello, se seguirá un proceso secuencial comenzando desde la selección de la imagen base de un servicio hasta la finalización del *docker-compose.yml*, con el que se desplegará dicho microservicio en conjunción con algún otro. Así, el proceso de diseño seguirá la siguiente secuencia:

Id	Etapas de diseño
ED1	Establecimiento de la imagen base de Alpine Linux.
ED2	Selección de argumentos necesarios para la construcción de la imagen.
ED3	Etiquetado de la imagen para identificar servicio, versiones y autor.
ED4	Inclusión de todos los ficheros y archivos imprescindibles para el funcionamiento
ED5	Realización de todas las instalaciones y configuraciones.
ED6	Exposición de puertos utilizados por el contenedor.
ED7	Elección del fichero o comando de arranque del contenedor.
ED8	Creación del archivo .env con todas las variables de entorno del servicio.
ED9	Generación del servicio en el docker-compose.yml, indicando su nombre, la ruta al Dockerfile, las variables de entorno, los volúmenes y mapeos de puertos, en caso de tenerlos y la definición de la red a la que se conectan, asignándole su correspondiente dirección y nombre.

Tabla 4. Etapas de diseño del nuevo entorno

Como resultado de este proceso, se obtendrá el diseño de un servicio completo, como uno de los mostrados en la *Figura 15*, a partir de lo cual se reiterarán las etapas mencionadas para cada uno de los servicios, completándose finalmente el esquema propuesto.

Capítulo 4

Desarrollo del entorno

En este capítulo se ofrece una visión completa del desarrollo global del entorno y de cada uno de los servicios a desplegar de manera individual. Se mostrará de manera secuencial, siguiendo el orden establecido en las etapas de diseño del capítulo anterior.

4.1 INTRODUCCIÓN

En esta sección se va a hacer una breve introducción al marco del desarrollo del entorno. En primer lugar, las tecnologías y herramientas utilizadas durante el capítulo son: Docker y Docker Compose, para la virtualización y la orquestación; Portainer, para la simplificación de la gestión de imágenes, volúmenes, redes y contenedores Docker; GitHub y Docker Hub, como repositorios de versiones del desarrollo y las imágenes Docker; y Sublime Text, para la escritura de todo el código del proyecto.

Dada la secuencia que sigue en el sistema de PLICA, desde que un mensaje es captado en un sensor hasta que es pasado por la ontología, se toma la decisión de desarrollar las tecnologías en dicho orden. Por tanto, las tecnologías seguirán la siguiente sucesión durante el desarrollo: Apache Zookeeper y Apache Kafka, Apache Spark, Elasticsearch, ElastAlert, Apache Jena Fuseki y Pandora FMS, habilitando *SSH* en todos ellos. Se dividirán en subsistemas en función del proceso que realizan dentro del propio entorno.

Dado el gran número de archivos desarrollados y la extensión del código, a excepción del *docker-compose.yml* y el *.env* globales, los cuales se presentan en el *Anexo C* debido a su gran valor, se opta por proporcionar el resto de manera externa a este documento. Todo este desarrollo se puede encontrar en el repositorio de GitHub en la siguiente dirección:

https://github.com/acallejasz/TFG

4.2 SUBSISTEMA 1 - GESTIÓN DE FLUJOS

Este subsistema constituye la primera parte de este desarrollo, en el que se van a dockerizar las tecnologías Apache Kafka y Apache ZooKeeper. Como se mencionó anteriormente, este módulo se utilizará para procesar los datos de distintos sensores en tiempo real para ser enviados al siguiente módulo de procesamiento de datos.

Para que esto se haga de manera segura, se ha realizado un bastionado de los contenedores para que nadie externo pueda modificar el entorno y al mismo tiempo se han securizado todas las comunicaciones mediante la aplicación de autorización entre Kafka y ZooKeeper con *SASL* y autenticación de conexiones inter e intra-Kafka y Zookeeper con el protocolo *SSL*, cumpliendo así con el *RD3*. El objetivo por el cual se ha decidido realizar este despliegue con esas configuraciones es para evitar cualquier ataque de *MiTM* (Man in he Middle), el cual consiste en alguien anónimo llamado “X” se sitúa entre dos interlocutores, en nuestro caso los sensores y Apache Spark, e intercepta los mensajes de A hacia B, conociendo la información y a su vez dejando que el mensaje continúe su camino.

De este modo y con esta configuración, en caso de que alguien intercepte los mensajes que se envían de los sensores a través de Kafka porque la red de trabajo quede expuesta, estos van cifrados y no supone la pérdida de la confidencialidad ni integridad de estos. A continuación, se va a detallar cada una de las imágenes por separado, con su proceso de construcción y configuración siguiendo las etapas de diseño.

4.2.1 APACHE ZOOKEEPER

Para la virtualización de esta tecnología, se ha desarrollado un *Dockerfile* a partir de la documentación de ZooKeeper [28] y se ha utilizado para la configuración de seguridad con *SSL* y otras características la documentación referenciada en [29], [30] y [31]. Partiendo de la imagen de Alpine (*ED1*), se establecen como argumentos la versión (3.5.9), el directorio raíz para sus archivos, su dirección IP, el directorio y contraseña del almacén de claves y el de certificados (*ED2*) y un etiquetado que incluye el nombre de la tecnología, una descripción, su versión y el autor (*ED3*). Los archivos copiados a la imagen para su puesta a punto y configuración son: *zookeeper_jaas.conf, zookeeper_certs.sh, zkEnv.sh y zoo.cfg (ED4),* tras lo que será necesaria la instalación de librerías Java y otros requisitos de esta tecnología, como paquetes para la gestión de claves *SSL* y del *SSH (ED5).*

Tiene los puertos 2181, 2281, 2888 y 3888 expuestos, que son lo que se indican en la documentación, siendo los dos primeros los usados en este desarrollo como puerto del cliente y puerto seguro del cliente, respectivamente *(ED6).* Finalmente, se utilizará el archivo *zkServer.sh* y el comando *start-foreground*, para el arranque del servicio *(ED7)*. El tamaño final de la imagen es de 143.7 MB y el resultado se puede encontrar en el enlace de GitHub.

4.2.2 APACHE KAFKA

En esta ocasión, se ha desarrollado un *Dockerfile* a partir de la documentación de Kafka [32] y se ha utilizado para la configuración de seguridad con *SSL* y otras características la documentación referenciada en [33] y [34]. Partiendo de la imagen de Alpine (*ED1*), se establecen como argumentos la versión de Kafka (3.5.9), de Scala (2.13) y de Glibc (2.31-r0), el directorio raíz para sus archivos, la dirección IP del *broker*, el número total de estos y el directorio y contraseña del almacén de claves y el de certificados (*ED2*). A continuación, se establece un etiquetado con el nombre de la tecnología, una descripción, su versión y el autor (*ED3*) y se copian los archivos desarrollados: *start-kafka.sh, start-broker.sh, producer_ssl.properties, kafka_server_jaas.conf, consumer_ssl.properties, kafka_certs.sh, producer_ssl.properties (ED4*), algunos de ellos esenciales para la generación de certificados.

Tras esto, es necesaria la instalación de librerías Java y otros requisitos de esta tecnología, como paquetes para la gestión de claves *SSL* y del *SSH (ED5)*. Además, se expone el puerto seguro para autenticación y autorización, el 9093 y para pruebas se usará el puerto configurado para texto plano, el 9092 *(ED6)*. Finalmente, se utilizará el archivo *start-kafka.sh*, para el arranque del servicio al crearse el contenedor *(ED7)*. El tamaño final de la imagen es de 462.2 MB y el resultado del *Dockerfile* se puede ver en el enlace de GitHub.

Finalmente, tras la realización de ambos *Dockerfile*, se realiza el archivo *.env* donde se indican y se da valor a todas las variables de entorno de ambas tecnologías, tales como IPs, puertos, versiones, localización de ficheros, usuarios y contraseñas, entre otros (*ED8*). A continuación, se crea el *docker-compose.yml* para la orquestación, en el que definen dos servicios: *zookeeper* y *kafka*. Cuentan con su configuración de nombres, variables, puertos y volúmenes y se conectan a la red *PLICA*, con direcciones en la subred 172.20.0.0/24 y *gateway* 172.20.0.1, y las direcciones 172.20.0.11 y 172.20.0.2 para cada servicio (*ED9*).

En este punto, como se observa en el diagrama presentado, se pueden gestionar los flujos de todos los sensores a través de su *topic*, mediante uno o varios *broker* Kafka, con el correspondiente intercambio de certificados y comunicaciones encriptadas con *SSL*.

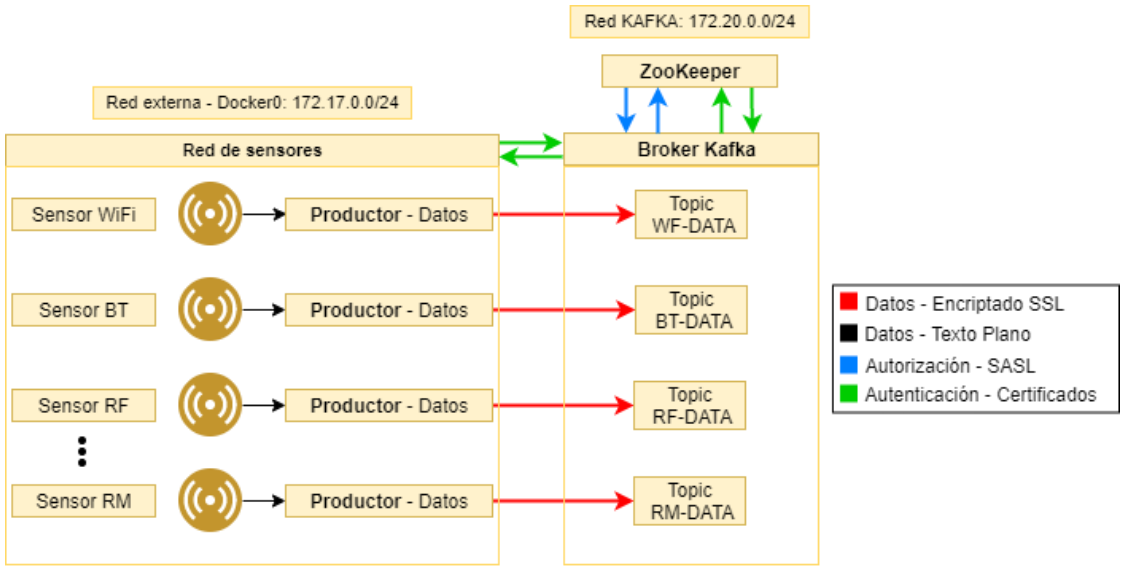

Figura 16. Diagrama de flujos del subsistema 1

4.3 SUBSISTEMA 2 – PROCESAMIENTO DE DATOS CON ML

Este subsistema es la segunda parte de este desarrollo, en el que se va a dockerizar la tecnología Apache Spark. El objetivo de este es procesar los datos recogidos en el módulo anterior en tiempo real mediante Spark, utilizando algoritmos de aprendizaje automático, uno por cada uno de los sensores, dando como resultado la propia trama de datos con un parámetro adicional que indica la presencia o no de una posible anomalía en los mismos.

Para que se haga de manera segura, se ha realizado un bastionado de los contenedores para que nadie externo pueda modificar el entorno y al mismo tiempo se han securizado las comunicaciones con autenticación con el protocolo *SSL*, cumpliendo así con el *RD3*.

4.3.1 APACHE SPARK

Para la virtualización de Apache Spark, se han desarrollado tres *Dockerfile* a partir de la documentación de Spark [35], [36] y se ha utilizado para la configuración de seguridad con *SSL* la documentación referenciada en [37]. Inicialmente, se plantea la realización del primero para la construcción de una imagen base, debido a que al funcionar a nivel de clúster, es necesario desplegar contenedores con arranque y configuración diferentes, dependiendo de si se trata del *master* del clúster o si es uno de los *workers* que trabaja como nodo ejecutor.

Partiendo de la imagen de Alpine (*ED1*), se establecen como argumentos la versión de Spark (2.4.4) y la de Hadoop (2.7) (*ED2*) y un etiquetado que incluye el nombre de la tecnología, una descripción, su versión y el autor (*ED3*). El archivo copiado a la imagen para su puesta a punto y configuración es *requirements.txt*, donde se indican todas las librerías de Python a instalar *(ED4).* A continuación, se requiere la instalación de Java y Python, junto con otros programas para gestión de sus librerías y otros requisitos de esta tecnología, como paquetes para la gestión de *SSL* y del *SSH (ED5).* Esta imagen base no tiene puertos expuestos ni fichero de arranque, ya que esto se realizará de manera individual en el siguiente paso *(ED6) (ED7).* El tamaño de la imagen es de 1.2 GB y el resultado final del *Dockerfile* se puede ver en el enlace de GitHub proporcionado.

Posteriormente, se realiza el *Dockerfile* para el *master* en el que, partiendo de la imagen que se compile según lo expuesto en el anterior parrafo, se establecen como argumentos la versión de Spark (2.4.4) y la de Hadoop (2.7) (*ED2*) y un etiquetado que incluye el nombre de la tecnología, una descripción, su versión y el autor (*ED3*). Los archivos copiados a la imagen para su puesta a punto y configuración son: *master.sh* y *template.sh*, el cual permite la ejecución de la aplicación de aprendizaje automático del sensor que se indique *(ED4).* No requiere de instalaciones adicionales, ya que todas fueron realizadas en la base *(ED5).* En esta ocasión, se exponen los puertos 8080, 7077 y 6066, tal y como se indica en la documentación, siendo el primero de ellos donde se encuentra la interfaz web de Spark *(ED6).* Finalmente, se utilizará en el arranque el archivo *master.sh*, que permite la ejecución del contenedor como nodo gestor del clúster *(ED7).* El tamaño de la imagen es de 1.2 GB y el resultado final del *Dockerfile* se puede ver en el enlace de GitHub proporcionado.

Como último paso en la virtualización, se realiza el mismo proceso para el *Dockerfile* del *worker*. Este se diferencia únicamente con el anterior en las etapas *ED4, ED6* y *ED7*, ya que en este caso se copia y utiliza como fichero de arranque *worker.sh*, que permite la ejecución del contenedor como nodo ejecutor de tareas del clúster y se expone el puerto 8081.

Finalmente, tras la realización de estos tres *Dockerfile*, se realiza el archivo *.env* donde se indican y se da valor a todas las variables de entorno tanto para la base, como para el *master* y los *workers*, tales como IPs, puertos, versiones, localización de ficheros y logs y memoria dada a cada servicio (*ED8*). Después, se crea el *docker-compose.yml* para la orquestación, en el que crean tres servicios: *spark-base* y *spark-master* y *spark-worker*, que cuentan con sus pertinentes configuraciones de nombres, variables, puertos y volúmenes. Son especialmente importante estos últimos, ya que será aquí donde se incluya la carpeta con la aplicación (*PLICA_V6.0*), que contiene los algoritmos necesarios para el procesamiento de datos y la configuración *SSL* y se conectan a la red *SPARK*, con direcciones en la subred 172.22.0.0/24 y *gateway* 172.22.0.1, y las direcciones 172.22.0.2 y 172.22.0.3 para el *master* y el primer *worker*, respectivamente (*ED9*).

Este desarrollo incluye además un *docker-compose.yml* adicional, el cual permite crear únicamente y de manera dinámica nuevos *worker* en el clúster en caso de ser necesario. Cabe destacar también, que la configuración realizada permite el balanceo automático de la carga del clúster entre los diferentes *worker* y que es tolerante a fallos, ya que si uno se cae, almacena el estado de la aplicación y los lleva a un nuevo nodo cuando este sea creado.

En este punto, al subsistema anterior se le añade un clúster de Spark para el procesado en tiempo real de los datos gestionados por Kafka mediante algoritmos de aprendizaje automático, cifrando todas las comunicaciones. El resultado se puede observar como sigue:

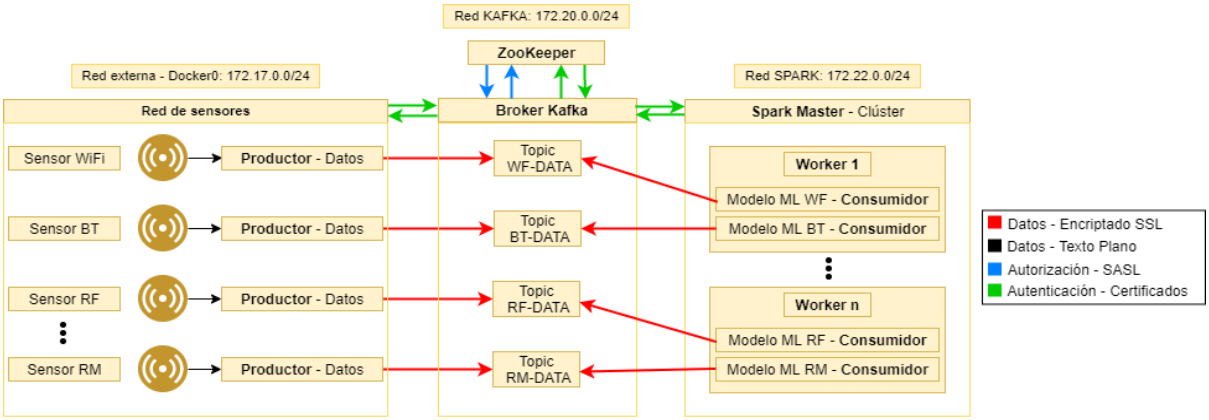

Figura 17. Diagrama de flujos de los subsistemas 1 y 2

4.4 SUBSISTEMA 3 - ALMACENAMIENTO DE DATOS

Este subsistema es la tercera parte de este desarrollo, en el que se va a dockerizar la tecnología Elasticsearch, en conjunto con su interfaz visual Kibana. El objetivo de este módulo es el de almacenar todos los datos procesados en el módulo anterior por Spark, permitiendo su visualización y la realización de peticiones para la extracción de resultados.

4.4.1 ELASTICSEARCH

Para la virtualización de Elasticsearch, se han desarrollado dos *Dockerfile* a partir de su documentación [38]. Dado que esta tecnología cuenta con imágenes Docker oficiales, se hace uso de estas como imagen base (*ED1**), centrándose el desarrollo en el proceso de configuración y seguridad de las imágenes. Se establecen como argumentos la versión de Elasticsearch y Kibana (7.9.2) (*ED2*) y un etiquetado que incluye el nombre de la tecnología, una descripción, su versión y el autor (*ED3*).

No es necesario la inclusión de ningún archivo adicional, pero si la realización de dos ficheros de configuración *.yml,* en los cuales se establecen direcciones de red, credenciales del sistema y el formato del clúster, entre otros *(ED4).* Tampoco requiere de ninguna instalación adicional *(ED5)* ni de fichero de arranque *(ED7)* y los puertos expuestos son 9200 y el 5601, respectivamente para Elasticsearch y Kibana *(ED6).* El tamaño de la imagen de Elasticsearch es de 762.9 MB y la de Kibana de 1.2 GB y el resultado final de estos *Dockerfile* y ficheros de configuración se puede ver en el enlace de GitHub proporcionado.

Finalmente, se realiza el archivo *.env* donde se indican y se da valor a todas las variables de entorno, tales como IPs, puertos, versiones, credenciales y valores de ejecución limite permitida, entre otros (*ED8*). Posteriormente, se crea el *docker-compose.yml* para la orquestación y despliegue de este módulo, en el que crean dos servicios: *elasticsearch* y *kibana.* Ambos cuentan con sus configuraciones de nombres, variables, puertos y volúmenes y adicionalmente, límites de memoria máximos permitidos y se conectan a la red *ELASTISEARCH,* con direcciones en la subred 172.21.0.0/24 y *gateway* 172.21.0.1, y las direcciones 172.21.0.2 y 172.21.0.3 para cada servicio respectivamente (*ED9*).

4.5 SUBSISTEMA 4 - CORRELACIÓN DE EVENTOS

Este subsistema presenta la cuarta parte de este desarrollo, en el que se va a dockerizar la tecnología ElastAlert. El propósito de este módulo es la correlación sobre los datos almacenados en Elasticsearch, generando alertas de anomalías u otros patrones de interés, viendo así las incoherencias en los datos almacenados casi en tiempo real y de manera fiable.

4.5.1 ELASTALERT

Para la virtualización de ElastAlert, se ha desarrollado un *Dockerfile* a partir de su documentación [39]. Partiendo de la imagen de Alpine (*ED1*), se establecen como argumentos la versión de ElastAlert (0.2.4) y dos variables para la gestión de Python (*ED2*). A continuación, se fija un etiquetado con el nombre de la tecnología, una descripción, su versión y el autor (*ED3*) y se copia el archivo *start_elastalert.sh (ED4).*

Para la siguiente etapa, se requiere la instalación de Java, Python y Docker, junto con otros programas para gestión de sus librerías y otros requisitos de esta tecnología, como paquetes para la gestión del *SSH (ED5)*. No es necesaria la exposición de ningún puerto *(ED6)* y se utilizará el *start_elastalert.sh* para el arranque del servicio al crearse el contenedor *(ED7)*. El tamaño final de la imagen es de 760.1 MB y el resultado final del *Dockerfile* se puede ver en el enlace de GitHub proporcionado.

Finalmente, se realiza el archivo *.env* donde se indican y se da valor a todas las variables, tales como IPs, puertos y versiones (*ED8*). Seguidamente, se crea el *docker-compose.yml* para la orquestación y despliegue de este módulo, en el que crea un servicio: *elastalert*. Cuenta con su configuración de nombres, variables, puertos y volúmenes, siendo muy importantes en este caso, ya que es aquí donde se incluyen la configuración y reglas de correlación que se van a aplicar y se conecta a la red *ELASTALERT*, con direcciones en la subred 172.23.0.0/24 y *gateway* 172.23.0.1, y la dirección 172.23.0.2 para el servicio (*ED9*).

4.6 SUBSISTEMA 5 - APLICACIÓN DE ONTOLOGÍAS

En este subsistema se exhibe la realización de la penúltima etapa del desarrollo en el que se va a dockerizar la tecnología de Apache Jena Fuseki. La intención de este es la aplicación de diversas ontologías sobre los datos almacenados en Elasticsearch mediante *querys* realizadas desde dicho servidor Fuseki y el almacenamiento de sus resultados, todo en tiempo real.

4.6.1 APACHE JENA FUSEKI

Para la virtualización de Fuseki, se ha desarrollado un *Dockerfile* partiendo de la imagen de Alpine (*ED1*), estableciéndose como argumentos la versión de Fuseki (3.16.0) (*ED2*) y realizando un etiquetado con el nombre de la tecnología, una descripción, su versión y el autor (*ED3*). Además, se copia todo el contexto donde se encuentra dicho archivo, entre lo que se encuentra una carpeta de ontologías, con todos los ficheros de configuración e instalación necesarios *(ED4)*. Uno de ellos, *indexCheck.sh*, evita la ejecución del servicio de ontologías hasta que todos los índices especificados han sido creados, previniendo así errores y haciendo comprobaciones cada cierto tiempo.

A continuación, se requiere la instalación de Java y otros requisitos de esta tecnología, como paquetes para la gestión del *SSH (ED5)*, se expone el puerto 3030, donde se encontrará la interfaz web del servidor *(ED6)*, y se utiliza el fichero el *start_fuseki.sh* para el arranque del servicio al crearse el contenedor *(ED7)*. El tamaño final de la imagen es de 435 MB y el resultado final del *Dockerfile* se puede ver en el enlace de GitHub proporcionado.

Finalmente, se realiza el archivo *.env* donde se indican y se da valor a todas las variables de entorno, tales como IPs, puertos y versiones, indicando también los índices de Elasticsearch sobre los que aplicar las ontologías (*ED8*). Seguidamente, se crea el *docker-compose.yml* para la orquestación y despliegue de este módulo, en el que crea un servicio: *fuseki*. Cuenta con su configuración de nombres y variables, y se conecta a la red *FUSEKI*, con direcciones en la subred 172.24.0.0/24 y *gateway* 172.24.0.1, y la dirección 172.24.0.2 para el servicio (*ED9*). En este punto del desarrollo, la interacción entre todos los subsistemas está completa, con todas sus interfaces de red, nombres, conexiones y securización realizados. Así, se presenta la arquitectura global de todos estos subsistemas:

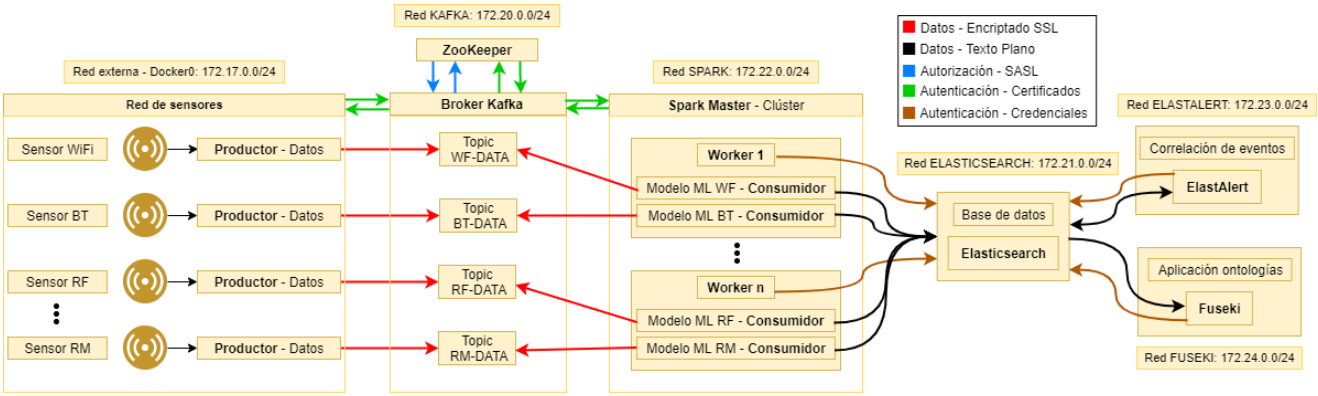

Figura 18. Diagrama de flujos de los subsistemas 1, 2, 3, 4 y 5

4.7 SUBSISTEMA 6 - MONITORIZACIÓN Y ACCESO REMOTO

Este subsistema presenta la parte final del desarrollo del proyecto, en la que se va a virtualizar la tecnología de Pandora FMS y se va a habilitar el acceso remoto a los contenedores con *SSH*. El propósito de este es permitir la monitorización de la gran mayoría de los contenedores que componen este desarrollo, de manera que puedan detectarse posibles errores y caídas y se puedan realizar configuraciones y automatizaciones ante estos casos de error o incluso detectar malos funcionamientos de alguna aplicación. Tras esto, se concederá la opción de acceder remotamente a los diferentes contenedores mediante la configuración y puesta a punto del protocolo *SSH*, permitiendo de este modo una gestión más simple y segura, ya que esta se realizará mediante claves RSA y se bloqueará el acceso por contraseña.

Llegados a este punto del desarrollo, se plantea una problemática con la imagen base de Alpine Linux, debido a que el software de monitorización es incompatible con este OS y no permite su correcta instalación en el sistema, lo que fuerza a buscar otra alternativa. Tras la realización de un estudio de la tecnología y de las diversas posibilidades para su virtualización, teniendo en cuenta el desarrollo anteriormente realizado, se toma la decisión de reemplazar la imagen base de Alpine por la imagen de Ubuntu 18.04.

Esto se debe a que es el sistema operativo más ligero, que más se parece a la anterior distribución de Alpine. Debido a esto, los anteriores subsistemas realizados pasarán a tener como imagen base la que se realiza aquí sobre Ubuntu y con el software de Pandora FMS y el *SSH* ya instalados y configurados *(ED1*)*, lo que hace que el único cambio necesario a realizar en todos sea en la *ED5*, ya que la instalación y configuración en este OS es distinta a la realizada con la imagen de Alpine. De este modo, se creará una nueva carpeta, llamada *PLICA-Ubuntu_based*, sobre la que se continuará este nuevo desarrollo con los cambios aquí mencionados. Sin embargo, todo el anterior desarrollo sigue disponible y se habilita en él también el *SSH*, para comparar ambos sistemas en cuanto a espacio de almacenamiento y velocidad de despliegue, pero sin contar el de Alpine con el software de Pandora FMS.

4.7.1 PANDORA FMS Y *SSH*

Para la virtualización de Pandora FMS, se ha desarrollado un *Dockerfile* partiendo de la imagen de Ubuntu:18.04 (*ED1**), sin argumentos adicionales (*ED2*) y con un etiquetado con el nombre de la tecnología, una descripción, su versión y el autor (ED3). Para esta tecnología, y para poder habilitar el *SSH*, se copian las carpetas y archivos: */pandoraAgents*, *enable_ssh.sh* y *id_rsa.pub*, siendo este último en el que se encuentra la clave pública del sistema *(ED4)*. A continuación, se requiere su instalación y paquetes para la gestión del *SSH (ED5)*, se expone el puerto 22, que es el habilitado para el acceso remoto *(ED6)*, y no se utiliza fichero para su arranque, ya que esta se usa como imagen base *(ED7)*. El tamaño final de la imagen es de 258 MB y el resultado del *Dockerfile* se puede ver en el enlace de GitHub.

Como no se desplegará como contenedor y no tiene variables, no se realiza el fichero .env ni el *docker-compose.yml* en este caso (*ED8*) (*ED9*). Sin embargo, será necesario iniciar la ejecución de ambos servicios en los subsistemas anteriores, por lo que en su fichero de arranque se incluyen las ordenes: */usr/sbin/sshd* y *service pandora_agent_daemon start*. Finalmente, se incluye la figura del entorno completo con todas las interacciones existentes:

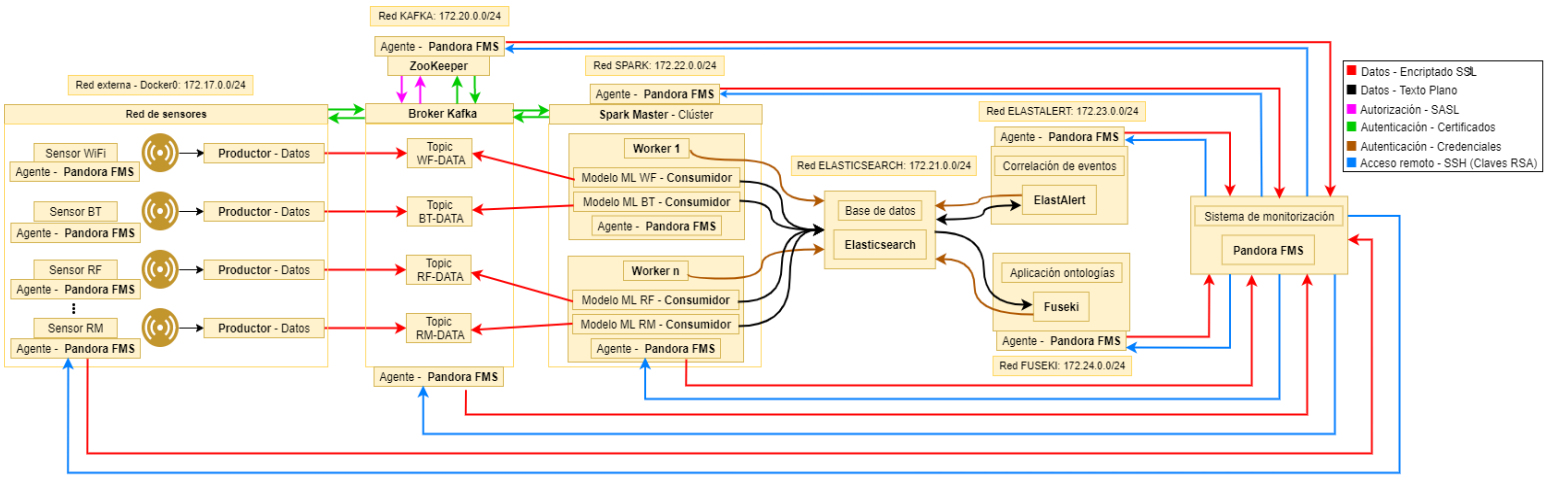

Figura 19. Diagrama de flujos del desarrollo completo

4.8 ORQUESTACIÓN CONJUNTA DE LOS SERVICIOS

Una vez se ha realizado con éxito todo el desarrollo y se cuenta con los ficheros *docker-compose.yml* y *.env* para la orquestación y despliegue de manera individual de cada uno de los subsistemas por separado para su despliegue en entornos distribuidos, se opta por la realización de estos dos archivos para la orquestación conjunta de todos los servicios con un solo comando. Esta sección se ha realizado de igual manera tanto para el desarrollo basado en Alpine Linux sin el servicio de Pandora FMS como para el desarrollo basado en Ubuntu con dicho servicio, para su posterior comparativa en el siguiente capítulo.

De esta manera, se escribe un fichero *.env* en el cual se encuentran todas las variables de entorno del proyecto completo, tanto versiones, como direcciones, de red, credenciales, rutas a ficheros y otras variables necesarias. Con esto se consigue que todas las variables que necesiten ser modificadas en cualquiera de las aplicaciones, incluso algunas que son compartidas entre servicios, como pueden ser los certificados, se encuentren todas en un mismo sitio, permitiendo además una mejora de la portabilidad a cualquier sistema u organización, ya que permite cambiar todo el entorno sin necesidad de preocuparse individualmente de los servicios, en caso de que así se desee.

Por otra parte, se realiza un *docker-compose.yml* global que cuenta con los siguientes servicios: *base, zookeeper, kafka, elasticsearch, kibana, spark-base, spark-master, worker, elastalert* y *fuseki*. Estos cuentan con sus pertinentes configuraciones de nombres, variables, puertos y volúmenes, así como con el establecimiento de las dependencias unos de otros, para garantizar el inicio de aquellos que se requiera en primer lugar. Además, se definen las redes para cada uno de los servicios tal y como se vio en su desarrollo individual, respetándose todas las direcciones y nombres asignados.

Adicionalmente, en este desarrollo se realiza el fichero *forget_remote_host.sh* para facilitar los accesos mediante *SSH*. Este sirve para que la máquina desde la que se acceda por *SSH* pueda olvidar las claves ECDSA de los contenedores a los que se conectan, ya que aunque su IP no cambie, si este se destruye y se despliega de nuevo produce un fallo ya que esta clave cambia, lo que no permite la conexión si no se realiza este proceso.

Finalmente, el despliegue de todo el entorno desarrollado puede iniciarse con simplemente hacer en una consola de comandos, situada en el directorio raíz de este proyecto, el siguiente comando: ***docker-compose up –scale base=0 –scale spark-base=0***, ya que estos dos servicios solo son utilizados como imagen base para otras imágenes y no necesitan ser desplegados en contenedores. Quedaría así finalizado el desarrollo, con los requisitos y etapas de diseño establecidas en el capítulo anterior, resultando en un despliegue como el de la *figura 15*.

Capítulo 5

Resultados y validación

En este capítulo se manifiestan los resultados alcanzados tras la finalización del desarrollo y su validación con pruebas de diferente índole. Además, se incluye una revisión donde se plantea si se ha logrado la consecución de los objetivos planteados en el primer capítulo.

5.1 EJECUCIÓN Y PRUEBA GLOBAL DE ENTORNO

En primer lugar, para comprobar el funcionamiento de todos los servicios, se realiza una prueba global del sistema, desplegándolo mediante el comando: ***docker-compose up –scale base=0 –scale spark-base=0.*** Para esta prueba, se hace uso de los siguientes archivos que han sido desarrollados con este objetivo y que están disponibles en GitHub en /*scripts_pruebas*:

- *crear_topics.sh.* Este nos permite la generación de *topics* Kafka, uno por cada uno de los sensores del proyecto y acepta dos parámetros, el primero de ellos el número de réplicas y en segundo lugar, el número de particiones de cada uno.
- *crear_productor.sh.* Permite la creación de un contenedor Kafka como *productor*, lo que permite simular el envío de mensajes de uno de los sensores y que toma como primer parámetro el nombre del *topic* sobre el que va a enviar los datos.
- *crear_consumidor.sh.* Permite la creación de un contenedor Kafka como *consumidor*, tomando como primer parámetro el nombre del *topic* del que lee los datos.
- *startPy.sh.* Inicia la ejecución en el *spark-master* de la rutina de aprendizaje máquina que se desee, tomando como parámetro el nombre de la aplicación a iniciar.
- *ingresar_datos.sh.* Crea un nuevo índice en Elasticsearch y añade datos.
- *tramas_x.* Cuenta con ejemplos de datos de los sensores de varios tipos, tanto anómalos como procesados sin encontrar amenazas.

Una vez entrado en contexto, hay que comprobar inicialmente que todos los contenedores están desplegados y corriendo en la máquina en los puertos esperados. Esto se puede comprobar mediante el comando *service docker status*, que devuelve un resultado satisfactorio de acuerdo con los servicios que deben estar ejecutándose y las direcciones y puertos que se propusieron para estos. Este resultado se muestra a continuación:

```
acallejasz@acallejasz:~/Escritorio/TFG/PLICA-Ubuntu_based_pandoraFMS$ service docker status
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2021-05-16 10:32:08 CEST; 10h ago
     Docs: https://docs.docker.com
 Main PID: 926 (dockerd)
    Tasks: 138
   CGroup: /system.slice/docker.service
           ├─  926 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
           ├─ 1469 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 9000 -container-ip 172.17.0.2 -container-port 9000
           ├─22126 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 2181 -container-ip 172.20.0.11 -container-port 2181
           ├─22143 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 23 -container-ip 172.20.0.11 -container-port 22
           ├─22167 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 8080 -container-ip 172.22.0.2 -container-port 8080
           ├─22215 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 9200 -container-ip 172.21.0.2 -container-port 9200
           ├─22252 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 25 -container-ip 172.21.0.2 -container-port 22
           ├─22451 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 7077 -container-ip 172.22.0.2 -container-port 7077
           ├─22478 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 26 -container-ip 172.22.0.2 -container-port 22
           ├─22625 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 9092 -container-ip 172.20.0.2 -container-port 9092
           ├─22668 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 24 -container-ip 172.20.0.2 -container-port 22
           ├─22690 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 5601 -container-ip 172.21.0.3 -container-port 5601
           ├─23565 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 27 -container-ip 172.22.0.3 -container-port 22
           ├─23945 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 28 -container-ip 172.23.0.2 -container-port 22
           ├─23970 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 3030 -container-ip 172.24.0.2 -container-port 3030
           └─24045 /usr/bin/docker-proxy -proto tcp -host-ip 0.0.0.0 -host-port 29 -container-ip 172.24.0.2 -container-port 22
```

Figura 20. PID, direcciones y puertos del despliegue

Posteriormente al arranque, se inicia la comprobación del correcto funcionamiento del acceso remoto mediante *SSH* por medio de claves RSA, que están en el host y el contenedor. Esto se realiza de este modo, dado que el acceso por contraseña ha sido deshabilitado para evitar ataques de fuerza bruta a los contenedores. Concretamente, se conecta al contenedor *spark-master*, donde además se comprueba también que el proceso de Pandora FMS está en ejecución. En este caso, se ejecutarán los comandos: *ssh root@172.22.0.2 y service pandora_agent_daemon status*, cuyo resultado puede verse en la siguiente figura:

Figura 21. Acceso por *SSH* a un contenedor y PID de Pandora FMS

Prosiguiendo con la prueba de funcionalidad, se utiliza el comando *./crear_topics.sh 1 1*, mediante el que se consigue la creación de un *topic* por cada uno de los sensores con una réplica y una partición y se despliegan cuatro nuevos contendores de Kafka mediante *./crear_productor x*, que simularán ser cada uno, uno de los sensores, en este caso el de WiFi, el de bluetooth, el de radio frecuencia y el de redes móviles.

Figura 22. Creación de *topics y productores* simulando sensores

Antes de poder realizar el envío de mensajes, es necesario que las aplicaciones de aprendizaje automático con Spark, correspondientes a cada uno de los sensores simulados en la prueba anterior estén activas, para que estas procesen los datos del sensor que le corresponde. Para esta tarea se hace uso de los siguientes comandos: *./startPy.sh wifi*, *./startPy.sh bluetooth*, *./startPy.sh rf* y *./startPy.sh rm*, los cuales comienzan la ejecución de estas tareas en los *worker* que estén disponibles. Dado que para este caso solo se ha desplegado uno con el Compose general, se desplegará otro adicional con el *docker-compose.yml* especificado en el desarrollo para comprobar también la funcionalidad de escalabilidad y balanceo de carga. Podemos observar que todas están ejecutándose en estos a igualdad de recursos, tal y como se encuentran definidos en el archivo *.env*, disponible en el *Anexo C*, y balanceados, lo que podemos observar en la interfaz gráfica que proporciona el servicio de Spark en la siguiente figura:

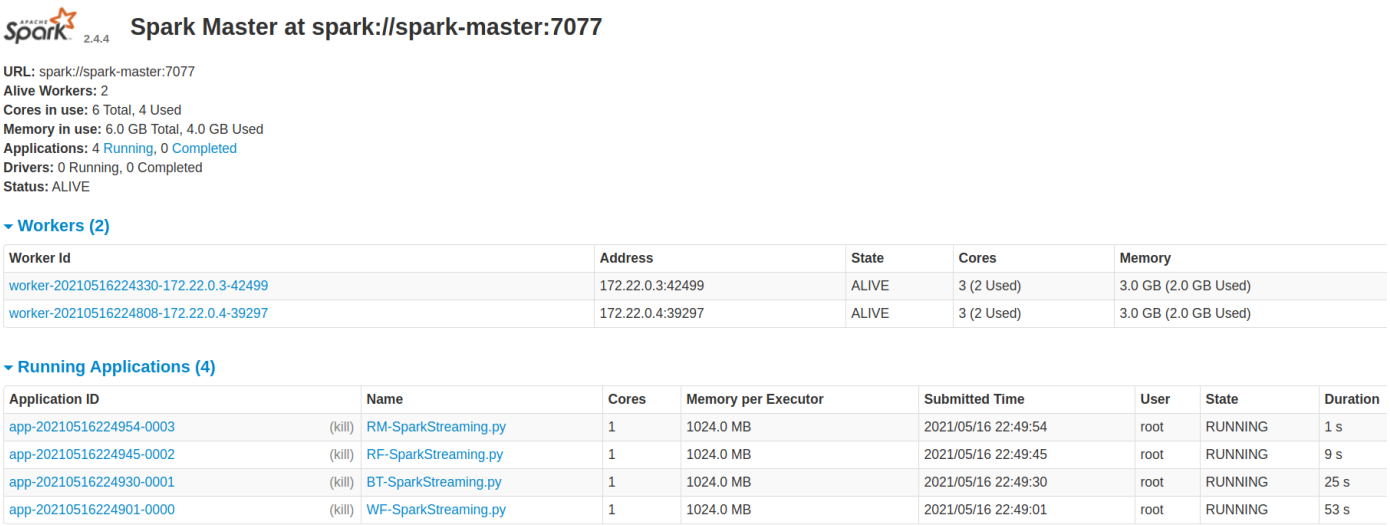

Figura 23. Clúster Spark con cuatro aplicaciones balanceadas en dos *workers*

Una vez están corriendo todas las aplicaciones, utilizando los contenedores creados en el paso anterior simulando los sensores, se envía por cada uno de ellos trazas de sus respectivos tipos, las cuales han sido obtenidas de manera real a partir de dispositivos de dicha índole. Una vez estas trazas son enviadas, su aplicación correspondiente de Spark las procesa y crea un índice en Elasticsearch con el nombre identificativo del proceso y almacena dichas trazas en su índice, donde además aparece un parámetro adicional *anomalía*, que ha sido añadido como resultado del procesado con los algoritmos de su aplicación de Spark.

Para la visualización de estas trazas almacenadas se utiliza la interfaz gráfica de Elasticsearch, Kibana, la cual se ha levantado en otro contenedor y está disponible en el puerto 5601. De este modo, se crean cuatro índices: *wifi*, *bluetooth*, *radio_frecuencia y redes_moviles.*

A continuación, se muestra un ejemplo de estos datos almacenados correctamente tras todo el proceso, concretamente los de Bluetooth. En ellos se pueden ver como la aplicación de Spark ha determinado que se puede tratar de una posible anomalía, tal y como se esperaba. Para el resto de los índices ocurre lo mismo, por lo que se omite el resultado.

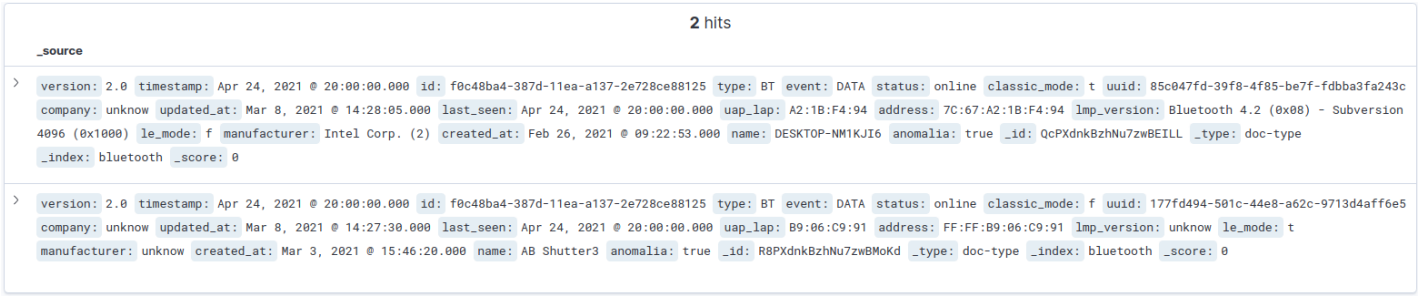

Figura 24. Resultado almacenado en Elasticsearch tras el procesamiento de tramas

Tras estar todos estos datos almacenados en Elasticsearch con sus respectivos índices, la aplicación de correlación de ElastAlert crea una serie de nuevos índices con los nombres: *elastalert_correlacion*, *elastalert_status* y variantes de estos. Si visualizamos dichos índices en Kibana podemos ver como las reglas de correlación y alertas han sido aplicadas obteniéndose los siguiente resultados:

Figura 25. Alertas y correlación almacenadas por Elastalert tras procesar tramas

Podemos observar cómo se han generado 4 alertas, una por cada tipo de sensor, indicando que se han encontrado posibles tramas anómalas con sus características y además, se da una alerta de correlación con una coincidencia de relación entre estos cuatro eventos.

Finalmente, una vez comprobado el correcto funcionamiento de todos los anteriores subsistemas, se evidencia a continuación que esto ocurre también con el subsistema de ontologías con Fuseki. En esta ocasión, todos las tramas almacenadas en Elasticsearch son tomadas por Fuseki mediante peticiones y se aplican a estas una serie de reglas ontologías para detectar cuales son los posibles ataques a los que se están enfrentando, siendo estos resultados almacenados en el servidor en la ruta */PLICA.* Esto se refleja en la siguiente figura:

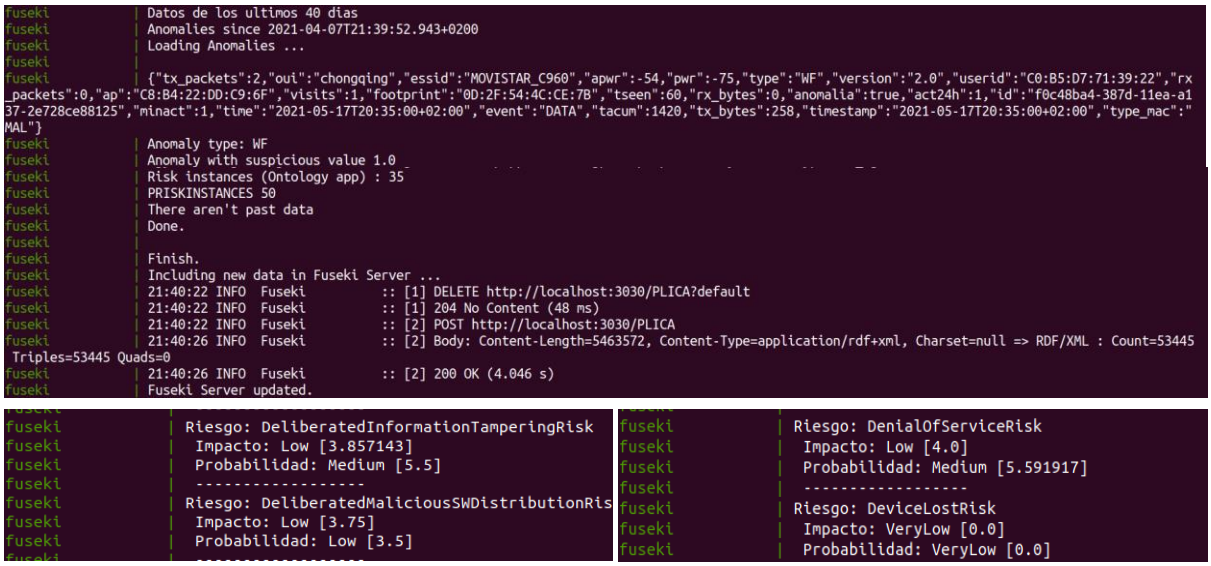

Figura 26. Aplicación de ontologías y resolución de posibles ataques

De este modo, se puede concluir como satisfactoria la prueba global de todos los subsistemas desplegados, funcionando conjuntamente, en la realización de un ciclo completo del proceso de vida, desde que una trama es detectada hasta que se le aplica una ontología. En este punto, tal y como se plantearon en el primer capítulo, se puede validar el cumplimiento de los objetivos *O1, O2, O4* y *O6,* ya que estos han sido alcanzados y comprobados.

5.2 COMPROBACIÓN DE SEGURIDAD Y CIFRADO

En esta sección se van a mostrar una serie de pruebas funcionales realizadas para la comprobación de la seguridad del entorno mediante una comparativa de tráfico capturado con la herramienta Wireshark en diferentes redes, simulando lo que ocurriría en un ataque MiTM, para tráfico de mensajes enviados a través de un *productor* Kafka y Spark, sin configuración de autorización y autenticación *SSL* y con ella. Con esto, se pretende demostrar el correcto funcionamiento y configuración de la seguridad y al mismo tiempo, justificar y entender porque es necesaria esta configuración para evitar la pérdida de la confidencialidad y de la integridad en caso de recibir un ataque.

Siguiendo los pasos de la sección anterior, cuyo resultado puede verse en la *Figura 20* y *Figura 22*, en primer lugar se despliega todo el entorno mediante el comando ***docker-compose up –scale base=0 –scale spark-base=0*** y seguidamente, se crean los *topics* y el *productor*, que para este ejemplo se hará sobre *WF-DATA*. De este modo, se va a exponer una comparativa en la captura de paquetes con la herramienta Wireshark, correspondientes al envío del mensaje "*Este es el mensaje enviado*" entre el *productor*, que simula el funcionamiento de un sensor y Spark, en primera instancia sin toda la configuración *SSL*, es decir, utilizando los puertos no seguros 2181 (ZooKeeper) y 9092 (Kafka) y Spark sin *SSL* y seguidamente utilizando toda la configuración de seguridad, con los puertos seguros 2281 (ZooKeeper) y 9093 (Kafka) y con *SSL* configurado en Spark.

Para comprender las redes donde se capturan los paquetes, hay que entender que la IP en la que se encuentra el *productor*, la 172.17.0.4, se corresponde con la red *docker0* y su bridge por defecto, ya que el *productor* no tiene por qué encontrarse dentro de la red KAFKA al ser alguien externo que va a enviar mensajes. Por su parte, la 172.20.0.1 es la que hace de *gateway* a la red KAFKA, donde se encuentran definidos todos los contenedores con los *brokers* y ZooKeeper y la 172.22.0.2 y 172.22.0.3, correspondientes al *master* y *worker* de Spark. Así, cuando se envíe un mensaje de un *productor* a través de un *broker* Kafka, este paquete se corresponderá con la IP origen 172.17.0.4 y destino 172.20.0.1, mientras que la autenticación y recepción en Spark será entre la IP origen 172.20.0.1 y destino 172.22.0.2/3.

En la siguiente figura, se aprecia la captura de un paquete en la red KAFKA, tras el envío de un mensaje en el *productor*, con origen en este y destino el *gateway*, a través del puerto 9092, sin configuración de seguridad, en texto plano. Se observa cómo en la traza marcada de azul se puede extraer información de quien es el que manda el mensaje (*console-producer*), cual es el *topic* por el que se envía (*WF-DATA*) y el mensaje (*Este es el mensaje enviado*). Por tanto, en caso de que alguien interceptara el mensaje, supondría una pérdida de la confidencialidad y el atacante descubriría información de todo el entorno, lo que puede dar lugar a que consiga un acceso mayor a nuestro sistema, siendo un peligro para la seguridad.

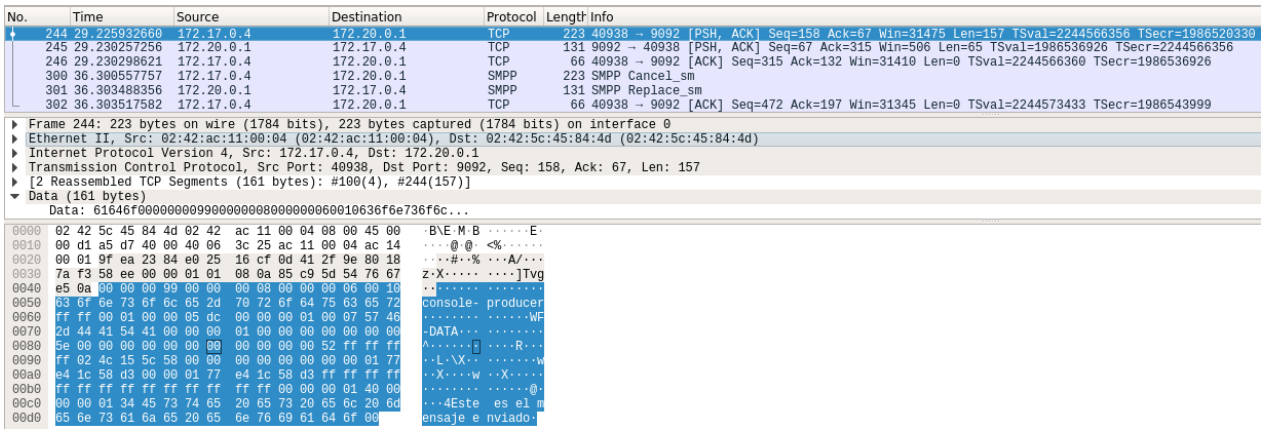

Figura 27. Captura de paquetes sin configuración de seguridad mediante Wireshark

En esta segunda imagen, se aprecia la captura de un paquete en la red SPARK, tras el envío de un mensaje en el *productor*, con origen en el *gateway* Kafka y con destino Spark, a través del puerto 9093, usando *SSL* con *TLS v1.2* y utilizando toda la configuración de seguridad en los contenedores. Podemos observar cómo inicialmente se han verificado los certificados y se ha producido un *Handsake* entre Kafka y Spark, indicando que la conexión que se está estableciendo es segura. Esto lo podemos corroborar además viendo que el mensaje está cifrado (*Encrypted Application Data*) y por tanto, en caso de que alguien interceptara el mensaje, no supondría una pérdida de la confidencialidad, haciendo al sistema seguro, y cumpliendo de este modo con el *O3* del primer capítulo. Hay que destacar, que no es posible enviar o consultar información del *broker* si no se dispone de los certificados.

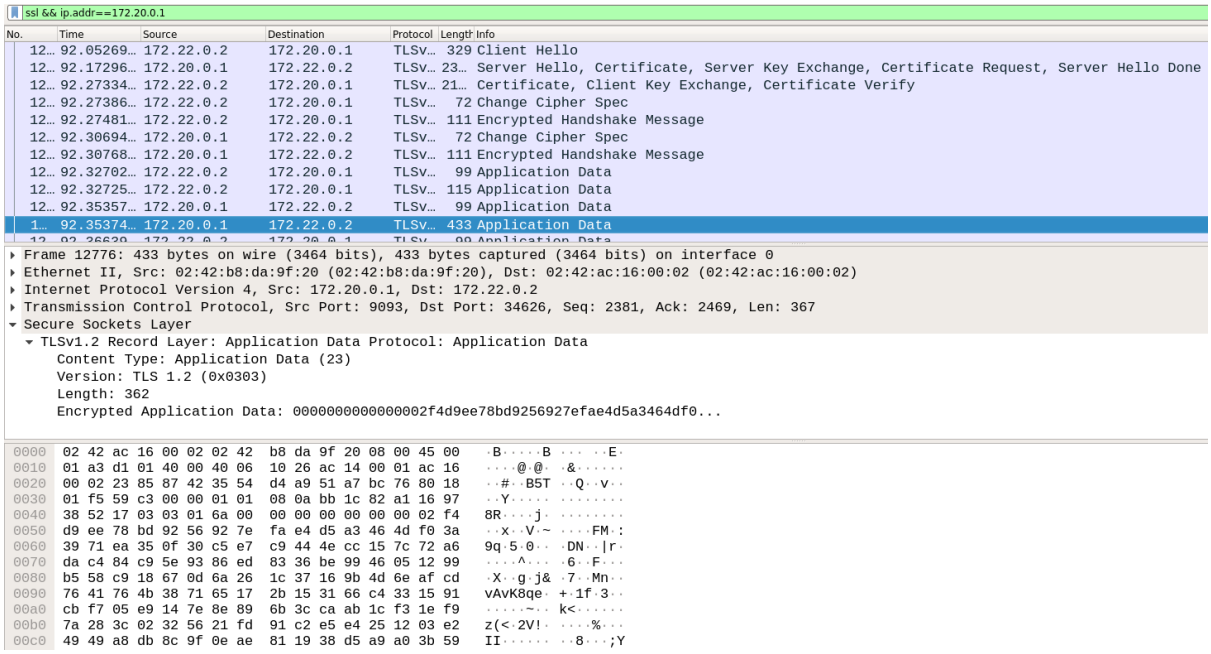

Figura 28. Captura de paquetes con configuración de seguridad mediante Wireshark

5.3 COMPARATIVA ENTRE ENTORNOS

Para finalizar este capítulo se presentarán en esta sección dos tablas comparativas. En la primera de ellas se realizará una comparativa entre el primer entorno desarrollado a partir de la imagen base de Alpine Linux y el finalmente desarrollado con todos los requerimientos a partir de la imagen de Ubuntu. En la segunda, se hará una comparativa entre el entorno monolítico a sustituir y el desarrollado en este proyecto basado en nuevas tecnologías de virtualización y orquestación. En ambas, se mostrará el espacio ocupado y los tiempos de creación de las imágenes y de despliegue de cada caso, para comprobar si ha conseguido cumplir con los objetivos expuestos en el primer capítulo relativos a estas características.

En la primera tabla se puede observar como el hecho de haber tenido que cambiar el OS base ha supuesto un incremento en la capacidad de almacenamiento necesaria, pero sin llegar a suponer un problema real, en comparación a la necesidad de este cambio para completar adecuadamente el desarrollo previsto. Por otro lado, los tiempos de compilación y despliegue de imágenes son prácticamente iguales, por lo que no supone cambio. Además, se omite los almacenamientos de Elasticsearch y Kibana por ser iguales, pero se añaden al total final.

Características		Desarrollo Alpine	Desarrollo Ubuntu
Espacio de almacenamiento (MB)	Base	5.6	63.1
	Zookeeper	143.7	511.6
	Kafka	462.2	556.4
	Spark (x3)	1200	1900
	ElastAlert	760.1	1300
	Fuseki	435	738.4
	Pandora	-	258
	Total	7369.5	11090.4
Tiempo de creación (min)		19.65	18.33
Tiempo de despliegue (min)		2.83	3.05

Tabla 5. Comparativa de métricas entre los desarrollos basados en Alpine y Ubuntu

Por último, la segunda tabla demuestra el cumplimiento del objetivo *O5*. Como se observa, el entorno monolítico queda en evidencia ante el desarrollo con Docker, dado que este le supera en todas las métricas realizadas en el servidor de PLICA, lo que se concluye en una clara justificación de la elaboración del proyecto. Así, quedan validados todos los objetivos que se propusieron al inicio de este documento y se da por finalizado el proyecto.

Características	Entorno monolítico	Entorno desarrollado
Espacio de almacenamiento (GB)	17.8	11.1
Tiempo de ejecución completa (min) (Primera vez, sin nada instalado)	20.68	18.51
Tiempo de despliegue (s)	58	8.8
Tiempo de arranque (min)	2.7	1.5

Tabla 6. Comparativa de métricas entre el entorno monolítico y el desarrollado

Capítulo 6

Conclusiones y líneas futuras

En este capítulo se exponen las diferentes conclusiones que se extraen de la realización del proyecto y las líneas futuras sobre las que se puede trabajar tras su finalización.

6.1 CONCLUSIONES

En la actualidad, más aún tras la pandemia en la que está sumida el mundo, se ha producido un incremento de la digitalización y virtualización de todos los servicios, siendo más común la aparición de nuevos ataques y brechas de seguridad que han afectado a múltiples corporaciones.

La realización de este trabajo de fin de grado consolida la idea de que los entornos monolíticos están abocados a su desaparición. Como se ha podido ver, las nuevas tecnologías de virtualización y orquestación permiten gestionar de una manera más segura los entornos al mismo tiempo que los hace más rápidos, escalables, portables y resilientes.

Concretamente, el uso de las tecnologías de Docker y Docker Compose constituyen una disrupción con respecto a sus predecesores en el mundo de la virtualización, con su gestión basada en el uso de contenedores aislados para proporcionar los servicios con independencia de la máquina o sistema operativo en el que se encuentre.

El sistema diseñado para la migración presenta grandes mejoras con respecto a su antecesor y es completamente funcional, lo que permitirá su pronta incorporación al proyecto. Además, se han logrado de manera satisfactoria todos los objetivos que se marcaron al inicio de este documento, tanto a nivel de diseño como de desarrollo y resultado final, gracias a la aplicación correcta de las pautas marcadas y los conocimientos adquiridos en su consecución.

Finalmente, quiero destacar que la realización de este proyecto ha sido un gran reto que me ha aportado muchos conocimientos en un ámbito que me era desconocido al comienzo de este año y que me ha hecho evolucionar como ingeniero en un sector cada vez más en auge.

6.2 LÍNEAS FUTURAS

Tras el desarrollo del entorno para su migración a PLICA, en esta sección se plantean posibles mejoras y líneas futuras sobre las que trabajar en el proyecto. A través de las ideas propuestas a lo largo de este documento, se continua con la intención de seguir avanzando en la securización del entorno y la mejora de su portabilidad y escalabilidad.

En concreto, las líneas a seguir serían entorno al despliegue completo en la nube del subsistema 2, el de Apache Spark, a través de la tecnología de Kubernetes, ya que es el servicio más crítico de todo el sistema a nivel de necesidad de recursos computacionales y de almacenamiento. Además, se planteará una mejora de la securización en cuanto al acceso a contenedores, mediante una autorización especifica al usuario a solo determinadas rutas, servicios y ficheros y acceso por contraseña, en el fichero *sudoers*.

Bibliografía

[1] Oracle, «Oracle VM VirtualBox,» [En línea]. Available: https://www.virtualbox.org/. [Último acceso: 2021].

[2] Oracle, «Oracle VM VirtualBox,» [En línea]. Available: https://www.virtualbox.org/manual/UserManual.html#vdidetails. [Último acceso: 2021]. [Último acceso: 2021].

[3] D. Inc, «Docker overview | Docker Documentation,» [En línea]. Available: https://docs.docker.com/get-started/overview/. [Último acceso: 2021].

[4] IBM, «What is Docker? | IBM,» [En línea]. Available: https://www.ibm.com/cloud/learn/docker. [Último acceso: 2021].

[5] U. Digital, «Que es la plataforma de contenedores Docker | Universo Digital,» [En línea]. Available: https://universo-digital.net/que-es-la-plataforma-de-contenedores-docker/. [Último acceso: 2021].

[6] HashiCorp, «Introduction | Vagrant by HashiCorp,» [En línea]. Available: https://www.vagrantup.com/intro/index. [Último acceso: 2021].

[7] HashiCorp, «Boxes | Vagrant by HashiCorp,» [En línea]. Available: https://www.vagrantup.com/docs/boxes. [Último acceso: 2021].

[8] D. Inc, «Overview of Docker Compose | Docker Documentation,» [En línea]. Available: https://docs.docker.com/compose/. [Último acceso: 2021].

[9] T. K. Authors, «What is Kubernetes? | Kubernetes,» [En línea]. Available: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Último acceso: 2021].

[10] T. K. Authors, «Kubernetes Components | Kubernetes,» [En línea]. Available: https://kubernetes.io/docs/concepts/overview/components/. [Último acceso: 2021].

[11] T. K. Authors, «Understanding Kubernetes Objects | Kubernetes,» [En línea]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/. [Último acceso: 2021].

[12] T. A. S. Foundation, «Apache ZooKeeper,» [En línea]. Available: https://zookeeper.apache.org/. [Último acceso: 2021].

[13] T. A. S. Foundation, «ZooKeeper Because Coordinating Distributed Systems is a Zoo,» [En línea]. Available: https://zookeeper.apache.org/doc/r3.7.0/zookeeperOver.html. [Último acceso: 2021].

[14] T. A. S. Foundation, «Introduction | Apache Kafka,» [En línea]. Available:

https://kafka.apache.org/intro. [Último acceso: 2021].

[15] T. A. S. Foundation, «Documentation | Apache Kafka,» [En línea]. Available: https://kafka.apache.org/documentation/. [Último acceso: 2021].

[16] T. A. S. Foundation, «Overview - Spark 3.1.1 Documentation,» [En línea]. Available: https://spark.apache.org/docs/latest/index.html. [Último acceso: 2021].

[17] T. A. S. Foundation, «Cluster Mode Overview - Spark 3.1.1 Documentation,» [En línea]. Available: https://spark.apache.org/docs/latest/cluster-overview.html. [Último acceso: 2021].

[18] E. B.V., «What is elasticsearch? | Elastic,» [En línea]. Available: https://www.elastic.co/what-is/elasticsearch. [Último acceso: 2021].

[19] Yelp, «ElastAlert - Easy & Flexible Alerting with Elasticsearch,» [En línea]. Available: https://elastalert.readthedocs.io/en/latest/elastalert.html#overview. [Último acceso: 2021].

[20] P. FMS, «Pandora:QuickGuides: What is Pandora FMS - Pandora FMS Wiki,» [En línea]. Available: https://pandorafms.com/docs/index.php?title=Pandora:QuickGuides_EN:What_Is_Pandora_FMS. [Último acceso: 2021].

[21] T. A. S. Foundation, «Apache Jena - Apache Jena Fuseki,» [En línea]. Available: https://jena.apache.org/documentation/fuseki2/. [Último acceso: 2021].

[22] Portainer, «Portainer | Open Source Container Management GUI,» [En línea]. Available: https://www.portainer.io/. [Último acceso: 2021].

[23] T. W. Foundation, «Chapter 1. Introduction,» [En línea]. Available: https://www.wireshark.org/docs/wsug_html_chunked/ChapterIntroduction.html. [Último acceso: 2021].

[24] S. HQ, «Sublime Text - A sophisticated text editor for code, makeup and prose,» [En línea]. Available: https://www.sublimetext.com/. [Último acceso: 2021].

[25] D. Inc, «Docker Hub QuickStart | Docker Documentation,» [En línea]. Available: https://docs.docker.com/docker-hub/. [Último acceso: 2021].

[26] D. Inc, «What is a container? | App Containerization | Docker,» [En línea]. Available: https://www.docker.com/resources/what-container. [Último acceso: 2021].

[27] D. Inc, «Image Layer Details - alpine:3.10,» [En línea]. Available: https://hub.docker.com/layers/alpine/library/alpine/3.10/images/sha256-25a240f2ac2dd7f7267b6e93d0c291b7e05ca63a3d455e57bc414a048ec8c3f3?context=explore. [Último acceso: 2021].

[28] T. A. S. Foundation, «ZooKeeper: Because Coordinating Distributed Systems is a

Zoo,» [En línea]. Available: https://zookeeper.apache.org/doc/r3.5.9/zookeeperStarted.html. [Último acceso: 2021].

[29] T. A. S. Foundation, «ZooKeeper: Because Coordinating Distributed Systems is a Zoo,» [En línea]. Available: https://zookeeper.apache.org/doc/r3.5.9/zookeeperAdmin.html. [Último acceso: 2021].

[30] A. C. O.-S. Project, «ZooKeeper SSL User Guide - Apache Zookeeper,» [En línea]. Available: https://cwiki.apache.org/confluence/display/ZOOKEEPER/ZooKeeper+SSL+User+Guide. [Último acceso: 2021].

[31] C. Inc, «ZooKeeper Security | Confluent Documentation,» [En línea]. Available: https://docs.confluent.io/platform/current/security/zk-security.html#encrypt-zk. [Último acceso: 2021].

[32] T. A. S. Foundation, «Apache Kafka,» [En línea]. Available: https://kafka.apache.org/quickstart. [Último acceso: 2021].

[33] C. Inc, «Security Tutorial | Confluent Documentation,» [En línea]. Available: https://docs.confluent.io/platform/current/security/security_tutorial.html. [Último acceso: 2021].

[34] C. Inc, «Encryption and Authentication with SSL | Confluent Documentation,» [En línea]. Available: https://docs.confluent.io/platform/current/kafka/authentication_ssl.html#creating-ssl-keys-and-certificates. [Último acceso: 2021].

[35] T. A. S. Foundation, «Quick Start - Spark 3.1.1 Documentation,» [En línea]. Available: https://spark.apache.org/docs/latest/quick-start.html. [Último acceso: 2021].

[36] T. A. S. Foundation, «Configuration - Spark 3.1.1 Documentation,» [En línea]. Available: https://spark.apache.org/docs/latest/configuration.html. [Último acceso: 2021].

[37] T. A. S. Foundation, «Security - Spark 3.1.1 Documentation,» [En línea]. Available: https://spark.apache.org/docs/latest/security.html. [Último acceso: 2021].

[38] E. B.V., «Configuring Elasticsearch | Elasticsearch Guide [7.12] | Elastic,» [En línea]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html. [Último acceso: 2021].

[39] Yelp, «Running ElastAlert for the first Time,» [En línea]. Available: https://elastalert.readthedocs.io/en/latest/running_elastalert.html#downloading-and-configuring. [Último acceso: 2021].

Anexo A

Aspectos éticos, económicos, sociales y ambientales

En este anexo se va a realizar una descripción de los aspectos éticos, económicos, sociales y ambientales que pueden extraerse de la realización de este trabajo de fin de grado.

A.1 INTRODUCCIÓN

Este trabajo de fin de grado está integrado en el proyecto PLICA de la ETSIT, con el objetivo de solucionar una necesidad real de mejora de características, tanto a nivel de velocidad de despliegue, resiliencia y portabilidad, como a nivel de seguridad del entorno monolítico utilizado en el grupo para el desarrollo diario de su trabajo.

Para la identificación de los impactos relacionados con el proyecto se presenta el siguiente contexto. Desde el punto de vista social, los grupos de interés que están asociados o que se ven afectados por la realización de este proyecto se encuentran en el ámbito universitario de la ETSIT en primera instancia, aunque más adelante se verán en detalle otros grupos derivados de la actividad del proyecto. Respecto al aspecto económico y medioambiental, el desarrollo realizado en este trabajo aporta valor en ambas secciones, por su carácter de suplir una necesidad real de cambio y por ser un producto software virtualizado. En relación al aspecto legal, el hecho de estar securizado, como se verá en la siguiente sección, aporta numerosas ventajas gracias a la confidencialidad e integridad de los datos en tránsito.

A.2 IMPACTOS RELACIONADOS CON EL PROYECTO

En primer lugar, el nuevo entorno diseñado y desarrollado supone un incremento de la productividad del equipo del proyecto, lo que podrá tener una repercusión económica a largo plazo. Dado que este nuevo entorno cuenta con encriptado de las comunicaciones y por ende un incremento significativo de la seguridad en los procesos a realizar, supone que la institución directamente afectada, la UPM, en caso de tener algún tipo de fallo de seguridad, minimice las posibilidades de enfrentarse a problemas legales que puedan resultar del ataque, gracias al mantenimiento de la confidencialidad y de la integridad de los datos utilizados y en tránsito derivados de la utilización del entono desarrollado en este proyecto.

En el aspecto medioambiental, el entorno evita la compra innecesaria de dispositivos específicos para su empleo y uso. Esto se debe a que se trata de un sistema completamente virtualizado y que se puede desplegar por completo en la nube, siendo compatible con cualquier tipo de ordenador, independientemente de su sistema operativo. Esto supone una reducción de residuos a largo plazo, fomentando la reducción de la huella ambiental.

Adicionalmente, el hecho de que el proyecto PLICA esté ligado al programa de gobierno COINCIDENTE hace que este sea de alta importancia de cara a la sociedad española, dado que los resultados derivados del éxito del proyecto podrían tener una repercusión muy positiva en la ciberdefensa de España, mejorando la calidad de vida de las personas.

A.3 ANÁLISIS DE ALGUNO DE LOS PRINCIPALES IMPACTOS

Esta sección se va a centrar en el análisis detallado de uno de los impactos más influyentes de entre los presentados en el apartado anterior, el impacto económico-legal asociado a la seguridad del entorno. El simple hecho de contar con un nivel de seguridad mayor es motivo más que suficiente para que las instituciones que hagan uso del entorno cumplan con requisitos legales de cara a la protección de la integridad y confidencialidad de los datos tratados, que en su defecto podría llegar a acarrear sanciones económicas y la toma de medidas legales por incumplimiento en el tratamiento de los datos.

A nivel empresarial, esta mejora también es muy importante ya que consigue que la corporación mantenga y no pierda valor y reputación empresarial de cara a la sociedad y frente a otras instituciones, lo que supondría pérdidas económicas.

A.4 CONCLUSIONES

A lo largo del proyecto se han tomado diversas decisiones sobre requisitos y objetivos a cumplir para el entorno a diseñar y desarrollar, que han permitido alcanzar un ecosistema que favorece al desarrollo sostenible y a la mejora de la sociedad en general y del ecosistema de la universidad en particular.

Además, se presenta un ciclo de vida de crecimiento económico del proyecto en base al aumento de la productividad, dando una alternativa a la economía de usar y tirar, ofreciendo un compromiso con la integridad y confidencialidad de los datos tratados.

De esta manera, puede concluirse que todas las personas y organizaciones implicadas en el proyecto obtienen un beneficio e impactos positivos a raíz del trabajo aquí expuesto.

Anexo B

Presupuesto económico

En este anexo se va a detallar una estimación del coste y el esfuerzo económico que están asociados a la realización de este trabajo de fin de grado y que han sido necesarios para haber podido realizar con éxito la migración del entorno desarollado al proyecto PLICA. Para ello, se tendrán en cuenta los materiales utilizados y la mano de obra, sobre lo que se estimarán los costes directos e indirectos como parte del coste total del proyecto.

- ***Costes de mano de obra***

Para el cálculo del coste asociado a la mano de obra se realiza una aproximación del número de horas empleadas en las diferentes etapas realizadas durante el trabajo, cuyo desglose se puede ver en la siguiente tabla, seguida de otra donde se incluye el coste.

Ocupaciones	**Horas**
Estudio de las diferentes tecnologías (Docker, Docker-compose, Kafka, Spark, ...)	70
Estudio del entorno monolítico, sus funcionalidades y nuevos requisitos	30
Diseño del entorno basado en Docker	50
Desarrollo de los diferentes subsistemas	150
Integración de los subsistemas para su despliegue	50
Pruebas	20
Elaboración de la memoria	70
Total	440

Tabla 7. Desglose de las ocupaciones llevadas a cabo en el proyecto y su coste en horas

Coste de mano de obra		
Horas	Precio/hora	**Total**
440	13.1€	**5764€**

Tabla 8. Costes de mano de obra

Cabe destacar que este cálculo se extrae a partir del salario medio anual de un Ingeniero de Telecomunicación en España con una edad inferior a 24 años, que se estima de 23.553€ según el Informe Socioprofesional del COIT: https://bit.ly/3ou8NuW. De este modo, si consideramos que hay un total aproximado de 45 semanas laborales anuales, trabajando un total de 40 horas semanales, lo que supone 1800 horas anuales, la estimación resulta en que el salario medio anual es de 13.1€/hora.

- ***Costes de material***

Dentro de los costes materiales asociados a este proyecto solo se encuentra el ordenador personal con el que se han realizado todas las ocupaciones anteriormente planteadas, ya que el resto de los programas utilizados son de licencia libre. El ordenador portátil se trata de un MSI PS24 8RB, con un procesador Intel Core i7, 16GB de memoria RAM y un disco duro SDD de 512 GB, con un coste asociado de 1200€.

Coste de material			
Material	Precio compra	Amortización	**Total**
Ordenador portátil	1200€	20%	**240€**

Tabla 9. Costes materiales

- ***Costes indirectos***

Los costes indirectos fruto de la realización de este proyecto, como luz, agua o internet, se estiman en un 15% sobre los costes directos anteriormente calculados, suma de los costes de mano de obra y material.

Costes indirectos		
Coste directo	Estimación	**Total**
6004€	15%	**900.6€**

Tabla 10. Costes indirectos

- ***Costes totales***

Finalmente, se presenta el cálculo de los costes totales asociados al proyecto, en donde se incluyen el beneficio industrial y la aplicación del IVA.

Coste total		
Concepto		**Total**
Costes directos	*Mano de obra*	5764€
	Material	240€
	Total	6004€
Costes indirectos		900.6€
Total antes de beneficios e impuestos		6904.6€
Beneficio industrial (10%)		690.46€
Total antes de impuestos		7595.1€
IVA (21%)		1594.96
Total presupuesto		**9190.1€**

Tabla 11. Coste total asociado al proyecto

Anexo C

Ficheros de código: *docker-compose.yml* y *.env*

En este anexo se van a presentar dos de los archivos realizados durante el desarrollo del proyecto, dada su gran relevancia. En concreto, el fichero *.env*, donde se encuentran todas las variables de entorno del sistema, y el archivo *docker-compose.yml*, que permite la creación, orquestación y despliegue de todos los contenedores desarrollados con un solo comando.

- ***Docker-compose.yml***

```yaml
version: '3.9'
services:

  base:
    build:
      context: PandoraFMS/
    image: ubuntu-pandora:base
    container_name: (Use --scale base=0)
    networks:
      BASE:

  zookeeper:
    build:
      context: Kafka-ZooKeeperSSL/zookeeper/
      args:
        ZOOKEEPER_VERSION: ${ZOOKEEPER_VERSION}
        ZK_HOME: ${ZK_HOME}
        ZOOKEEPER_KEYSTORE_FILENAME: ${ZOOKEEPER_KEYSTORE_FILENAME}
        ZOOKEEPER_TRUSTSTORE_FILENAME: ${ZOOKEEPER_TRUSTSTORE_FILENAME}
        ZOOKEEPER_STORE_WORKING_DIRECTORY: ${ZOOKEEPER_STORE_WORKING_DIRECTORY}
        ZOOKEEPER_IP: ${ZOOKEEPER_IP}
        ZOOKEEPER_TRUSTSTORE_PASSWORD: ${ZOOKEEPER_TRUSTSTORE_PASSWORD}
    image: zookeeper:${ZOOKEEPER_VERSION}
    container_name: zookeeper
    environment:
      ZK_HOME: ${ZK_HOME}
      ZOOKEEPER_IP: ${ZOOKEEPER_IP}
      ZOOKEEPER_SECURE_PORT: ${ZOOKEEPER_SECURE_PORT}
      ZOOKEEPER_KEYSTORE_LOCATION: ${ZOOKEEPER_KEYSTORE_LOCATION}
      ZOOKEEPER_KEYSTORE_PASSWORD: ${ZOOKEEPER_KEYSTORE_PASSWORD}
      ZOOKEEPER_TRUSTSTORE_LOCATION: ${ZOOKEEPER_TRUSTSTORE_LOCATION}
      ZOOKEEPER_TRUSTSTORE_PASSWORD: ${ZOOKEEPER_TRUSTSTORE_PASSWORD}
    ports:
      - "2181:2181"
      - "2281:2281"
      - "23:22"
    networks:
      KAFKA:
        ipv4_address: 172.20.0.11
    volumes:
      - certs_zookeeper:/var/ssl/private/zookeeper
```

```
kafka:
  build:
    context: Kafka-ZooKeeperSSL/kafka/
    args:
      KAFKA_VERSION: ${KAFKA_VERSION}
      SCALA_VERSION: ${SCALA_VERSION}
      GLIBC_VERSION: ${GLIBC_VERSION}
      KAFKA_HOME: ${KAFKA_HOME}
      KAFKA_ADVERTISED_HOST_NAME: ${KAFKA_ADVERTISED_HOST_NAME}
      KAFKA_KEYSTORE_FILENAME: ${KAFKA_KEYSTORE_FILENAME}
      KAFKA_KEYSTORE_CLIENT_FILENAME: ${KAFKA_KEYSTORE_CLIENT_FILENAME}
      KAFKA_TRUSTSTORE_FILENAME: ${KAFKA_TRUSTSTORE_FILENAME}
      KAFKA_CLIENT_TRUSTSTORE_FILENAME: ${KAFKA_CLIENT_TRUSTSTORE_FILENAME}
      KAFKA_STORE_WORKING_DIRECTORY: ${KAFKA_STORE_WORKING_DIRECTORY}
      KAFKA_TRUSTSTORE_PASSWORD: ${KAFKA_TRUSTSTORE_PASSWORD}
      BROKERS_NUMBER: ${BROKERS_NUMBER}
  image: kafka:${KAFKA_VERSION}
  container_name: kafka
  environment:
    KAFKA_ADVERTISED_HOST_NAME: ${KAFKA_ADVERTISED_HOST_NAME}
    KAFKA_ZOOKEEPER_CONNECT: ${KAFKA_ZOOKEEPER_SECURE_CONNECT}
    KAFKA_ADVERTISED_PORT: ${KAFKA_SECURE_PORT}
    KAFKA_SECURE_PORT: ${KAFKA_SECURE_PORT}
    KAFKA_HOME: ${KAFKA_HOME}
    BROKERS_NUMBER: ${BROKERS_NUMBER}
    KAFKA_KEYSTORE_LOCATION: ${KAFKA_KEYSTORE_LOCATION}
    KAFKA_KEYSTORE_PASSWORD: ${KAFKA_KEYSTORE_PASSWORD}
    KAFKA_TRUSTSTORE_LOCATION: ${KAFKA_TRUSTSTORE_LOCATION}
    KAFKA_TRUSTSTORE_PASSWORD: ${KAFKA_TRUSTSTORE_PASSWORD}
    KAFKA_KEYSTORE_CLIENT_FILENAME: ${KAFKA_KEYSTORE_CLIENT_FILENAME}
    KAFKA_CLIENT_TRUSTSTORE_FILENAME: ${KAFKA_CLIENT_TRUSTSTORE_FILENAME}
    KAFKA_STORE_WORKING_DIRECTORY: ${KAFKA_STORE_WORKING_DIRECTORY}
    ZOOKEEPER_KEYSTORE_LOCATION: ${ZOOKEEPER_KEYSTORE_LOCATION}
    ZOOKEEPER_KEYSTORE_PASSWORD: ${ZOOKEEPER_KEYSTORE_PASSWORD}
    ZOOKEEPER_TRUSTSTORE_LOCATION: ${ZOOKEEPER_TRUSTSTORE_LOCATION}
    ZOOKEEPER_TRUSTSTORE_PASSWORD: ${ZOOKEEPER_TRUSTSTORE_PASSWORD}
  ports:
    - "9092:9092"
    - "9093:9093"
  networks:
    KAFKA:
      ipv4_address: 172.20.0.2
  depends_on:
    - zookeeper
  volumes:
    - certs_zookeeper:/var/ssl/private/zookeeper
    - certs_kafka:/var/ssl/private/kafka
    - kafka_config:/opt/kafka/config
    - /var/run/docker.sock:/var/run/docker.sock

elasticsearch:
  build:
    context: Elasticsearch-Kibana/elasticsearch/
    args:
      EK_VERSION: ${EK_VERSION}
      ELASTIC_CREDENTIALS: ${ELASTIC_CREDENTIALS}
  image: elasticsearch:${EK_VERSION}
  container_name: elasticsearch
```

```
    environment:
      ELASTIC_CREDENTIALS: ${ELASTIC_CREDENTIALS}
      ELASTIC_USERNAME: ${ELASTIC_USERNAME}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
      ELASTIC_CLUSTER_NAME: ${ELASTIC_CLUSTER_NAME}
      ES_JAVA_OPTS: -Xmx${ELASTICSEARCH_HEAP} -Xms${ELASTICSEARCH_HEAP}
      discovery.type: single-node
    volumes:
      - elasticsearch:/usr/share/elasticsearch/data
      - ./Elasticsearch-Kibana/elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml
    ports:
      - "9200:9200"
    networks:
      ELASTICSEARCH:
        ipv4_address: 172.21.0.2

  kibana:
    build:
      context: Elasticsearch-Kibana/kibana/
      args:
        EK_VERSION: ${EK_VERSION}
    image: kibana:${EK_VERSION}
    container_name: kibana
    volumes:
      - ./Elasticsearch-Kibana/kibana/config/:/usr/share/kibana/config:ro
    environment:
      ELASTIC_USERNAME: ${ELASTIC_USERNAME}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
      ELASTIC_HOST_PORT: ${ELASTICSEARCH_HOST}:${ELASTICSEARCH_PORT}
    depends_on:
      - elasticsearch
    ports:
      - "5601:5601"
    networks:
      ELASTICSEARCH:
        ipv4_address: 172.21.0.3

  spark-base:
    build:
      context: Spark-Hadoop/base/
      args:
        SPARK_VERSION: ${SPARK_VERSION}
        HADOOP_VERSION: ${HADOOP_VERSION}
    image: spark_base
    container_name: (Use --scale spark-base=0)
    environment:
      PYTHONHASHSEED: ${PYTHONHASHSEED}
    networks:
      SPARK:

  spark-master:
    build:
      context: Spark-Hadoop/master/
      args:
        SPARK_VERSION: ${SPARK_VERSION}
        HADOOP_VERSION: ${HADOOP_VERSION}
    image: spark-master:${SPARK_VERSION}-hadoop${HADOOP_VERSION}
    container_name: spark-master
    environment:
      SPARK_MASTER_PORT: ${SPARK_MASTER_PORT}
      SPARK_MASTER_WEBUI_PORT: ${SPARK_MASTER_WEBUI_PORT}
      SPARK_MASTER_LOG: ${SPARK_MASTER_LOG}
      SPARK_MASTER_NAME: ${SPARK_MASTER_NAME}
      SPARK_MASTER_PORT: ${SPARK_MASTER_PORT}
      SPARK_SUBMIT_ARGS: ${SPARK_SUBMIT_ARGS}
```

```
      #Kafka
      KAFKA_BROKER: ${KAFKA_SECURE_BROKER}
      WF_TOPIC: ${WF_TOPIC}
      BT_TOPIC: ${BT_TOPIC}
      FW_TOPIC: ${FW_TOPIC}
      RF_TOPIC: ${RF_TOPIC}
      RM_TOPIC: ${RM_TOPIC}
      SM_TOPIC: ${SM_TOPIC}
      #Elastic
      ELASTICSEARCH_IP: ${ELASTICSEARCH_IP}
      ELASTICSEARCH_PORT: ${ELASTICSEARCH_PORT}
      ELASTIC_USERNAME: ${ELASTIC_USERNAME}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
      #Security
      KEYSTORE: ${KEYSTORE}
      TRUSTSTORE: ${TRUSTSTORE}
      KEY: ${KEY}
      PASSWORD: ${PASSWORD}
      #Python
        #Files
      SPARK_APPLICATION_PYTHON_LOCATION_WF: ${SPARK_APPLICATION_PYTHON_LOCATION_WF}
      SPARK_APPLICATION_PYTHON_LOCATION_BT: ${SPARK_APPLICATION_PYTHON_LOCATION_BT}
      SPARK_APPLICATION_PYTHON_LOCATION_FW: ${SPARK_APPLICATION_PYTHON_LOCATION_FW}
      SPARK_APPLICATION_PYTHON_LOCATION_RF: ${SPARK_APPLICATION_PYTHON_LOCATION_RF}
      SPARK_APPLICATION_PYTHON_LOCATION_RM: ${SPARK_APPLICATION_PYTHON_LOCATION_RM}
      SPARK_APPLICATION_PYTHON_LOCATION_SM: ${SPARK_APPLICATION_PYTHON_LOCATION_SM}
        # Logs
      SPARK_APPLICATION_LOG_WF: ${SPARK_APPLICATION_LOG_WF}
      SPARK_APPLICATION_LOG_BT: ${SPARK_APPLICATION_LOG_BT}
      SPARK_APPLICATION_LOG_FW: ${SPARK_APPLICATION_LOG_FW}
      SPARK_APPLICATION_LOG_RF: ${SPARK_APPLICATION_LOG_RF}
      SPARK_APPLICATION_LOG_RM: ${SPARK_APPLICATION_LOG_RM}
      SPARK_APPLICATION_LOG_SM: ${SPARK_APPLICATION_LOG_SM}
    ports:
      - "8080:8080"
      - "7077:7077"
      - "25:22"
    networks:
      SPARK:
        ipv4_address: 172.22.0.2
    volumes:
      - spark_logs:/logs
      - certs_kafka:/app/certificados_kafka
      - ./Spark-Hadoop/base/PLICA_V6/:/app

  worker:
    build:
      context: Spark-Hadoop/worker/
      args:
        SPARK_VERSION: ${SPARK_VERSION}
        HADOOP_VERSION: ${HADOOP_VERSION}
    image: spark-worker:${SPARK_VERSION}-hadoop${HADOOP_VERSION}
    container_name: worker
    depends_on:
      - spark-master
    environment:
      SPARK_MASTER: ${SPARK_MASTER}
      SPARK_WORKER_WEBUI_PORT: ${SPARK_WORKER_WEBUI_PORT}
      SPARK_WORKER_LOG: ${SPARK_WORKER_LOG}
      SPARK_WORKER_CORES: ${SPARK_WORKER_CORES}
      SPARK_WORKER_MEMORY: ${SPARK_WORKER_MEMORY}
```

```
      #Security
      KEYSTORE: ${KEYSTORE}
      TRUSTSTORE: ${TRUSTSTORE}
      KEY: ${KEY}
      PASSWORD: ${PASSWORD}
    ports:
      - "26:22"
    networks:
      SPARK:
    volumes:
      - ./Spark-Hadoop/base/PLICA_V6/:/app
      - certs_kafka:/app/certificados_kafka

  elastalert:
    build:
      context: Elastalert/
      args:
        ELASTALERT_VERSION: ${ELASTALERT_VERSION}
    image: elastalert:${ELASTALERT_VERSION}
    container_name: elastalert
    depends_on:
      - elasticsearch
      - kibana
      - kafka
      - zookeeper
      - spark-master
      - spark-base
    environment:
      TZ: ${TZ}
      ELASTICSEARCH_IP: ${ELASTICSEARCH_IP}
      ELASTICSEARCH_PORT: ${ELASTICSEARCH_PORT}
      ELASTIC_USERNAME: ${ELASTIC_USERNAME}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
    ports:
      - "27:22"
    networks:
      ELASTALERT:
        ipv4_address: 172.23.0.2
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - ./Elastalert/configFiles/:/opt/config
      - ./Elastalert/rules:/opt/rules

  fuseki:
    build:
      context: Fuseki/
      args:
        FUSEKI_VERSION: ${FUSEKI_VERSION}
    image: fuseki:${FUSEKI_VERSION}
    container_name: fuseki
    depends_on:
      - elasticsearch
      - kibana
      - kafka
      - zookeeper
      - spark-master
      - spark-base
    environment:
      FUSEKI_VERSION: ${FUSEKI_VERSION}
      FUSEKI_IP: ${FUSEKI_IP}
      ELASTICSEARCH_IP: ${ELASTICSEARCH_IP}
      ELASTICSEARCH_PORT: ${ELASTICSEARCH_PORT}
```

```
      ELASTIC_USERNAME: ${ELASTIC_USERNAME}
      ELASTIC_PASSWORD: ${ELASTIC_PASSWORD}
      ELASTICSEARCH_INDEX: ${ELASTICSEARCH_INDEX}
    ports:
      - "3030:3030"
      - "28:22"
    networks:
      FUSEKI:
        ipv4_address: 172.24.0.2

networks:
  KAFKA:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.20.0.0/24
          gateway: 172.20.0.1
  ELASTICSEARCH:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.21.0.0/24
          gateway: 172.21.0.1
  SPARK:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.22.0.0/24
          gateway: 172.22.0.1
  ELASTALERT:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.23.0.0/24
          gateway: 172.23.0.1
  FUSEKI:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.24.0.0/24
          gateway: 172.24.0.1
  BASE:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.0.0.0/24
          gateway: 172.0.0.1
volumes:
  certs_zookeeper:
  certs_kafka:
  kafka_config:
  elasticsearch:
  spark_logs:
```

- ***.env***

```
# General settings

COMPOSE_PROJECT_NAME=PLICA

        # Versions

        ZOOKEEPER_VERSION=3.5.9
        KAFKA_VERSION=2.7.0
        SCALA_VERSION=2.13
        GLIBC_VERSION=2.31-r0
        SPARK_VERSION=2.4.4
        HADOOP_VERSION=2.7
        EK_VERSION=7.9.2
        ELASTALERT_VERSION=0.2.4
        FUSEKI_VERSION=3.16.0

# ZooKeeper Variables

        # Config

        ZK_HOME=/opt/zookeeper
        ZOOKEEPER_IP=172.20.0.11
        ZOOKEEPER_SECURE_PORT=2281

        # Security

        ZOOKEEPER_KEYSTORE_LOCATION=/var/ssl/private/zookeeper/zookeeper.server.keystore.jks
        ZOOKEEPER_KEYSTORE_PASSWORD=xxxxxxxx
        ZOOKEEPER_TRUSTSTORE_LOCATION=/var/ssl/private/zookeeper/zookeeper.server.truststore.jks
        ZOOKEEPER_TRUSTSTORE_PASSWORD=xxxxxxxx

        ZOOKEEPER_KEYSTORE_FILENAME="zookeeper.server.keystore.jks"
        ZOOKEEPER_TRUSTSTORE_FILENAME="zookeeper.server.truststore.jks"
        ZOOKEEPER_STORE_WORKING_DIRECTORY="/var/ssl/private/zookeeper"
# Kafka Variables

        # Config

        KAFKA_HOME=/opt/kafka
        KAFKA_PORT=9092
        KAFKA_SECURE_PORT=9093
        KAFKA_ADVERTISED_HOST_NAME=172.20.0.1
        KAFKA_ZOOKEEPER_CONNECT=zookeeper:2181
        KAFKA_ZOOKEEPER_SECURE_CONNECT=zookeeper:2281
        BROKERS_NUMBER=1

        # Security

        KAFKA_KEYSTORE_FILENAME=kafka.server.keystore.jks
        KAFKA_KEYSTORE_CLIENT_FILENAME=kafka.client.keystore.jks
        KAFKA_TRUSTSTORE_FILENAME=kafka.server.truststore.jks
        KAFKA_CLIENT_TRUSTSTORE_FILENAME=kafka.client.truststore.jks
        KAFKA_STORE_WORKING_DIRECTORY=/var/ssl/private/kafka

        KAFKA_KEYSTORE_LOCATION=/var/ssl/private/kafka/kafka.server.keystore.jks
        KAFKA_KEYSTORE_PASSWORD=xxxxxxxx
        KAFKA_TRUSTSTORE_LOCATION=/var/ssl/private/kafka/kafka.server.truststore.jks
        KAFKA_TRUSTSTORE_PASSWORD=xxxxxxxx

# Spark Variables

        # Base

        PYTHONHASHSEED=1

        # Master

        SPARK_MASTER_PORT=7077
        SPARK_MASTER_WEBUI_PORT=8080
        SPARK_MASTER_LOG=/spark/logs
```

```
            # Kafka

            KAFKA_BROKER=172.20.0.1:9092
            KAFKA_SECURE_BROKER=172.20.0.1:9093
            WF_TOPIC=WF-DATA
            BT_TOPIC=BT-DATA
            FW_TOPIC=FW-DATA
            RF_TOPIC=RF-DATA
            RM_TOPIC=RM-DATA
            SM_TOPIC=SM-DATA

            # Elastic -> Elasticsearch Variables

            # Security

            KEYSTORE=/app/certificados_kafka/kafka.client.keystore.jks
            TRUSTSTORE=/app/certificados_kafka/kafka.client.truststore.jks
            KEY=/app/certificados_kafka/ca_key_kafka
            PASSWORD=xxxxxxxx

    # Worker

    SPARK_MASTER="spark://spark-master:7077"
    SPARK_WORKER_WEBUI_PORT=8081
    SPARK_WORKER_LOG=/spark/logs
    SPARK_WORKER_CORES=3
    SPARK_WORKER_MEMORY=3G

    # Python

    SPARK_MASTER_NAME=spark-master
    SPARK_MASTER_PORT=7077
    SPARK_SUBMIT_ARGS= --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.4.4,org.elasticsearch:elasticsearch-hadoop:7.9.2

            # Files

            SPARK_APPLICATION_PYTHON_LOCATION_WF=/app/wifi/StructuredStreaming/DistanceKMeans/structuredWf.py
            SPARK_APPLICATION_PYTHON_LOCATION_BT=/app/bluetooth/StructuredStreaming/DistanceKMeans/structuredBt.py
            SPARK_APPLICATION_PYTHON_LOCATION_FW=/app/fw/StructuredStreaming/DistanceKmeans/structuredFw.py
            SPARK_APPLICATION_PYTHON_LOCATION_RF=/app/rf/StructuredStreaming/DistanceKmeans/structuredRf.py
            SPARK_APPLICATION_PYTHON_LOCATION_RM=/app/rm/StructuredStreaming/DistanceKmeans/structuredRm.py
            SPARK_APPLICATION_PYTHON_LOCATION_SM=/app/siem/StructuredStreaming/DistanceKmeans/structuredSm.py

            # Logs

            SPARK_APPLICATION_LOG_WF=logs/WF_logs.txt
            SPARK_APPLICATION_LOG_BT=logs/BT_logs.txt
            SPARK_APPLICATION_LOG_FW=logs/FW_logs.txt
            SPARK_APPLICATION_LOG_RF=logs/RF_logs.txt
            SPARK_APPLICATION_LOG_RM=logs/RM_logs.txt
            SPARK_APPLICATION_LOG_SM=logs/SM_logs.txt

# Elasticsearch Variables

    # Resources

    ELASTICSEARCH_HEAP=1024m

    # Hosts and Ports

    ELASTICSEARCH_IP=172.21.0.1
    ELASTICSEARCH_HOST=elasticsearch
    ELASTICSEARCH_PORT=9200

    KIBANA_HOST=kibana
    KIBANA_PORT=5601
```

```
      # Credientals
      # Username & Password for Admin Elasticsearch cluster.
      # This is used to set the password at setup, and used by others to connect to Elasticsearch at runtime.
      # Do not change the ELASTIC_USERNAME, it's required to be elastic. Otherwise there will be an error.

      ELASTIC_CREDENTIALS=false
      ELASTIC_USERNAME=elastic
      ELASTIC_PASSWORD=xxxx

      # Cluster

      ELASTIC_CLUSTER_NAME="elasticdocker-cluster"
# Elastalert variables

       TZ="UTC"

# Fuseki

FUSEKI_IP=172.24.0.2

       # Elastic connect

       ELASTICSEARCH_INDEX=wifi,radio_frecuencia
```