Universitat Oberta de Catalunya

uoc.edu

# Diseño y securización de sistemas de información web basados en contenedores y microservicios

**Nombre del estudiante:** Fraga Martín-Arroyo, Santiago
Nombre del Programa: M1.887, Seguridad y contenedores
Área de trabajo final: Seguridad empresarial

**Nombre Tutor/a de TF:** Albós Raya, Amadeu
**Nombre Profesor/a responsable de la asignatura:** Garcia Font, Victor

Fecha Entrega: 10 de enero de 2023

**FICHA DEL TRABAJO FINAL**

| | |
|---|---|
| **Título del trabajo:** | *Seguridad y en contenedores* |
| **Nombre del autor:** | *Fraga Martín-Arroyo, Santiago* |
| **Nombre del consultor/a:** | *Albós Raya, Amadeu* |
| **Nombre del PRA:** | *Garcia Font, Victor* |
| **Fecha de entrega (mm/aaaa):** | *01/2023* |
| **Titulación o programa:** | Ciberseguridad y Privacidad |
| **Área del Trabajo Final:** | *Seguridad empresarial* |
| **Idioma del trabajo:** | *Castellano* |
| Palabras clave | *Contenedores, seguridad, microservicios* |


**Resumen del Trabajo**

Actualmente los ciberataques contra empresas e instalaciones gubernamentales han aumentado enormemente provocando serios problemas a la actividad de las empresas y a las infraestructuras de países.

La finalidad de este proyecto es desarrollar un plan de transformación de un sistema de información tradicional a un modelo basado en microservicios y contenedores, para de esta forma poder aislar y mejorar la seguridad de los distintos aplicativos.

Para llevar a cabo este objetivo se comenzará analizando el sistema el cual queremos migrar para posteriormente proporcionar una solución adecuada haciendo uso de herramientas que nos faciliten la creación y administración de contenedores. Por otro lado, se estudiará la viabilidad de la utilización de orquestadores de contenedores y en qué casos es recomendable su uso.

Además, se realizará un análisis de la ejecución de contenedores en las plataformas cloud y las ventajas que se pueden obtener al usar estos servicios.

Para finalizar, se realizará una prueba de concepto aplicando la metodología descrita.

**Abstract**

Currently, cyber-attacks on companies and government have increased enormously, causing serious problems to the activity of companies and the infrastructure of countries.

The purpose of this project is to develop a transformation plan from a traditional information system of a company or government to a model based on containers and microservices, to isolate and improve the security of the

different applications.

To complete this objective, we will begin by analyzing the system which we want to migrate to make a solution using tools that facilitate the creation and administration of containers. On the other hand, we will analyze of using containers administrator will be studied and in which cases its use is recommended.

In addition, an analysis of the possible cloud platforms where containers can be run and the advantages.

Finally, the development of the application will be carried out.

# Índice

# Lista de figuras

# 1. Introducción

## 1.1. Contexto y justificación del trabajo

Los ciberataques se han convertido en una de las mayores preocupaciones para las empresas, los estados e incluso para las familias.

Hoy en día, cualquier empresa por pequeña que sea cuenta con un sistema de información que usa para almacenar la información de facturación, datos de sus clientes, programas informáticos, etc. Debido a esto, muchos cibercriminales ponen el ojo en este tipo de empresas, para una vez realizado el ciberataque comenzar a extorsionarles, como es el caso de los ransomware.

Otra forma de extorsión sería la de inutilizar la plataforma e-commerce de la empresa para de esta forma parar toda su actividad.

Durante el desarrollo de este trabajo fin de máster nos centraremos en la **securización y modernización del sistema de información** de una empresa para poder pasar de un sistema anticuado monolítico a un sistema con una arquitectura basada en **microservicios y contenedores,** poniendo el foco en la **seguridad de dichos contenedores**.

Uno de los aspectos más relevantes de la arquitectura que se desarrollará será el **uso de microservicios**, los cuales, gracias a su gran modularidad facilita en gran medida el desarrollo, reduciendo las dependencias y los posibles errores que se pueden cometer al mantener una arquitectura monolítica.

Además del uso de contenedores, se aplicarán otras **medidas de seguridad complementarias** para asegurar el correcto funcionamiento del aplicativo. Siguiendo en esta misma línea **buscaremos posibles vulnerabilidades** en las imágenes que se creen, así como **verificar la autoría** de estas. Este paso es muy importante ya que nos asegurará que nuestro contenedor ejecuta una imagen segura.

Por otro lado, se estudiará la posibilidad de llevar toda esta infraestructura al Cloud para poder **ceder la gestión y la seguridad de los servicios a un tercero.** Ya que, para empresas pequeñas no sería rentable encargarse ellas mismas de la seguridad de sus servicios debido a la gran inversión que esto conlleva.

Finalmente se investigará la viabilidad de usar una **plataforma de orquestación de contenedores** y ver cómo esta puede mejorar la seguridad de nuestro sistema.

## 1.2. Objetivos del trabajo

El objetivo principal del presente proyecto es la **securización de un sistema con una arquitectura basada en microservicios y desarrollado mediante contenedores**.

Para llevar a cabo dicho objetivo nos apoyaremos en la siguiente lista de metas:

- Analizar las amenazas en sistemas basados en contenedores.
- Diseñar un modelo seguro de implementación de un sistema basado en contenedores.
- Validar el diseño seguro desarrollado mediante una prueba de concepto.

## 1.3. Impacto en sostenibilidad, ético-social y de diversidad

El presente proyecto tiene en consideración las indicaciones del compromiso ético y global. En primer lugar, respecto a la **sostenibilidad**, el Objetivo de Desarrollo Sostenible 2030 (Naciones Unidas, 2022) número 9: *INDUSTRIA, INNOVACIÓN E INFRAESTRUCTURA* es el marco en el que se basa este proyecto ya que se pretende promover la industrialización sostenible (objetivo 9.2) y,

> *Modernizar la infraestructura y reconvertir las industrias para que sean sostenibles, utilizando los recursos con mayor eficacia y promoviendo la adopción de tecnologías y procesos industriales limpios y ambientalmente racionales, y logrando que todos los países tomen medidas de acuerdo con sus capacidades respectivas* (Objetivo 9.4 ODS; Naciones Unidas, 2022).

El presente trabajo cumple el objetivo de sostenibilidad ya que, gracias al uso de contenedores para la construcción del sistema de información, se está haciendo una utilización mucho más eficiente y sostenible de los recursos de los que se dispone.

Respecto al **compromiso ético y la responsabilidad social**, este proyecto se rige por el Objetivo de Desarrollo Sostenible 2030 de las Naciones Unidas (2022) número 8: *TRABAJO DECENTE Y CRECIMIENTO ECONÓMICO*. Más concretamente, se propone que se pueda aumentar la productividad económica de las empresas a través de un uso más innovador y moderno de la tecnología (objetivo 8.2 de la ODS). Optimizando recursos se contribuye al bien común de la sociedad ya que se produce un ahorro energético; esto se traduce en una ventaja social y un aumento de la competitividad de las empresas que pudieran beneficiarse de este proyecto.

Esto se consigue gracias a que con el uso de contenedores securizados correctamente, una empresa que siga las pautas citadas en el presente trabajo pueda conseguir un uso optimizado de los recursos de los que dispone obteniendo un aumento de la productividad económica.

Este proyecto respeta los principios deontológicos universales tales como la legalidad y la confidencialidad. *"Debemos ser especialmente escrupulosos con el tratamiento y utilización que damos a la información que manejamos y cumplir el deber de secreto, el cual constituye un derecho y un deber básico de la profesión"*. Se asumen durante la elaboración de este Trabajo Final de Máster, los deberes que se estipulan en el Código Deontológico de la Ingeniería Informática (CCII, 2022).

Teniendo en cuenta la **diversidad y los derechos humanos**, el presente TF respeta los ODS 5: *IGUALDAD DE GÉNERO* y 10: *REDUCCIÓN DE LAS DESIGUALDADES*, asegurando la transversalidad de la perspectiva de género en todas las etapas del proyecto y velando porque éste respete todos los Derechos Humanos (Naciones Unidas, 2022).

Es por ello por lo que durante todo el desarrollo del presente trabajo Fin de Máster no se discriminan las aportaciones de ningún autor o autora y se evitan los sesgos o estereotipos que pudieran provocar desigualdades sociales. Además, se hace uso del lenguaje inclusivo y se mantiene presente en todo momento la sensibilidad hacia las discriminaciones.

## 1.4. Estado del arte

En el siguiente apartado describiremos algunos de los trabajos relacionados con temas parecidos al tratado en el presente Trabajo Fin de Máster.

**Tabla 1. Estado del arte**

| **Título-Autor/a** | **Resumen** |
|---|---|
| Seguridad en Docker<br>González Hernández, Beatriz | El trabajo consiste en el desarrollo de una guía de uso de Docker, así como consejos de seguridad para su uso correcto (González, 2018). |
| Análisis de seguridad de arquitecturas basadas en Kubernetes<br>Simón Nóvoa, Manuel | Durante el desarrollo del proyecto se habla sobre securización de un clúster haciendo uso de Kubernetes (Simón, 2021). |
| Seguridad en Docker<br>Gamboa Fernández, Juan | Este trabajo Fin de Máster consiste en demostrar si la ejecución de contenedores en una infraestructura puede mantener un nivel de seguridad aceptable (Gamboa, 2018). |
| Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua<br>Aladrén García, Julio | La finalidad del trabajo Fin de Máster es la de investigar los mecanismos de securización de Docker y Kubernetes desde el punto de vista de los hipervisores (Aladrén, 2020). |

Fuente: elaboración propia a partir de González (2018), Simón (2021), Gamboa (2018) y Aladrén (2020).

Tal y como podemos observar en la tabla superior, existen otros trabajos Fin de Máster los cuales hablan de las herramientas **Docker y/o Kubernetes** y de su securización, es decir, basan todo el trabajo en ambas herramientas.

En contraposición, el presente TFM se basa en el concepto de **contenedor independientemente de la herramienta utilizada** y centra el foco del estudio en la **securización de un sistema de información** completo aplicando una arquitectura basada en **microservicios** y al uso de **herramientas de gestión de contenedores** como son **Podman** y **Docker**.

## 1.5. Enfoque y método seguido

El enfoque que se le ha dado a este trabajo ha sido el de adaptar un sistema tradicional existente a una arquitectura basada en microservicios y de contenedores haciendo énfasis en su configuración de forma segura.

El motivo por el cual se ha elegido dicha solución es gracias a sus ventajas como la modularidad, escalabilidad y aislamiento de los servicios. Además, con el uso de contenedores, el despliegue y seguridad de los distintos servicios mejora tal y como se puede visualizar en el resultado del presente TFM.

La metodología que seguiremos durante todo el desarrollo de este trabajo fin de máster está compuesta por las siguientes etapas:

**Plan de trabajo**
En esta primera etapa se define el contexto en el cual se desarrolla el trabajo, así como los objetivos que se quieren alcanzar. Además, se estudiará el impacto en sostenibilidad, ético-social y de diversidad.
Por último, se realizará una planificación incluyendo las tareas a realizar y un diagrama de Gantt.

**Análisis de las principales amenazas en sistemas de información**
Donde describiremos las principales amenazas a las que se enfrentan las aplicaciones web y cuales podrían llegar a ser sus consecuencias.

**Análisis del sistema actual**
En esta etapa elaboraremos un sistema de información el cual simulará un aplicativo de una empresa y estudiaremos tanto su arquitectura como su funcionamiento. Además, veremos como de crítico puede ser para la empresa si dicho sistema quedase inutilizado.

**Estudio de contenedores software**
Se realizará un estudio de en qué consiste un contenedor y cómo podemos usarlo para construir aplicaciones basadas en contenedores.

**Metodología para pasar una aplicación a un contenedor**
Explicaremos los pasos que debemos seguir para crear un contenedor con nuestro aplicativo.

**Diseño seguro de contenedores**
Se investigará posibles soluciones a las amenazas detectadas en la etapa "**Análisis de las principales amenazas en sistemas de información**", así como consejos para que nuestro contenedor sea lo más seguro posible.

**Diseño seguro de orquestadores**
Estudiaremos en que consiste un orquestador de contenedores y porqué es una buena opción para sistemas en los cuales tenemos una gran cantidad de contenedores que manejar.

**Estudio de plataformas cloud**
En este apartado estudiaremos las distintas alternativas que tenemos para llevar nuestro sistema basado en contenedores a la nube y cuál nos ofrece un mejor servicio.

**Prueba de concepto**
Se realizará una implementación de lo investigado en los apartados anteriores para de esta forma demostrar que los planteamientos que se han expuesto son los correctos.

**Conclusiones y trabajos futuros**
Donde definiremos las posibles mejoras que se podrían realizar en proyectos futuros y que han quedado fuera de nuestro alcance en el contexto del presente TFM y, analizaremos las conclusiones a las que hemos llegado durante el desarrollo del trabajo.

Gracias al uso de esta metodología, la cual consta de una primera fase de análisis y estudio, y una última de prueba de concepto, se logra aplicar las medidas de seguridad sobre el sistema de forma correcta, alcanzando por ende los objetivos marcados.

## 1.6. Planificación del trabajo

En la siguiente tabla se muestra un listado de las tareas que se irán desarrollando a lo largo del TFM:

**Tabla 2 Planificación de las tareas**

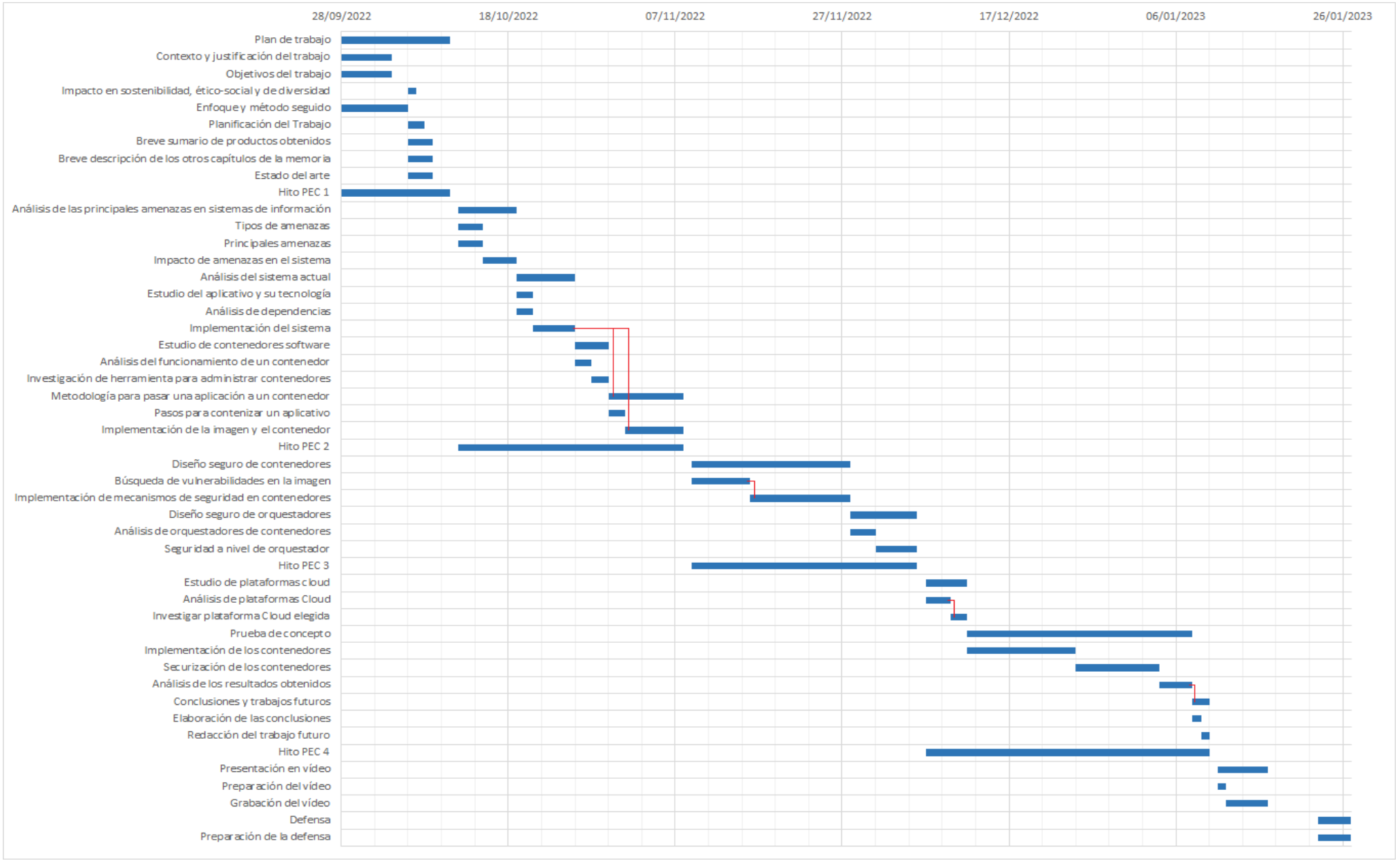
| Código | Tarea | Inicio | Fin | Duración |
|---|---|---|---|---|
| **1** | **Plan de trabajo** | **28/09/2022** | **11/10/2022** | **13** |
| 1.1 | Contexto y justificación del trabajo | 28/09/2022 | 04/10/2022 | 6 |
| 1.2 | Objetivos del trabajo | 28/09/2022 | 04/10/2022 | 6 |
| 1.3 | Impacto en sostenibilidad, ético-social y de diversidad | 06/10/2022 | 07/10/2022 | 1 |
| 1.4 | Enfoque y método seguido | 28/09/2022 | 06/10/2022 | 8 |
| 1.4 | Planificación del trabajo | 06/10/2022 | 08/10/2022 | 2 |
| 1.6 | Breve sumario de productos obtenidos | 06/10/2022 | 09/10/2022 | 3 |
| 1.7 | Breve descripción de los otros capítulos de la memoria | 06/10/2022 | 09/10/2022 | 3 |
| 1.8 | Estado del arte | 06/10/2022 | 09/10/2022 | 3 |
| 1.9 | **Hito** PEC 1 | 28/09/2022 | 11/10/2022 | 13 |

| | | | | |
|---|---|---|---|---|
| **2** | **Análisis de las principales amenazas en sistemas de información** | **12/10/2022** | 19/10/2022 | 7 |
| 2.1 | Tipos de amenazas | 12/10/2022 | 15/10/2022 | 3 |
| 2.2 | Principales amenazas | 12/10/2022 | 15/10/2022 | 3 |
| 2.3 | Impacto de amenazas en el sistema | 15/10/2022 | 19/10/2022 | 4 |
| **3** | **Análisis del sistema actual** | 19/10/2022 | 26/10/2022 | 7 |
| 3.1 | Estudio del aplicativo y su tecnología | 19/10/2022 | 21/10/2022 | 2 |
| 3.2 | Análisis de dependencias | 19/10/2022 | 21/10/2022 | 2 |
| 3.3 | Implementación del sistema | 21/10/2022 | 26/10/2022 | 5 |
| **4** | **Estudio de contenedores software** | 26/10/2022 | 30/10/2022 | 4 |
| 4.1 | Análisis del funcionamiento de un contenedor | 26/10/2022 | 28/10/2022 | 2 |
| 4.2 | Investigación de herramienta para administrar contenedores | 28/10/2022 | 30/10/2022 | 2 |
| **5** | **Metodología para pasar una aplicación a un contenedor** | 30/10/2022 | 08/11/2022 | 9 |
| 5.1 | Pasos para contenizar un aplicativo | 30/10/2022 | 01/11/2022 | 2 |
| 5.2 | Implementación de la imagen y el contenedor | 01/11/2022 | 08/11/2022 | 7 |
| 5.3 | **Hito** PEC 2 | 12/10/2022 | 08/11/2022 | 27 |
| **6** | **Diseño seguro de contenedores** | 09/11/2022 | 28/11/2022 | 19 |
| 6.1 | Búsqueda de vulnerabilidades en la imagen | 09/11/2022 | 16/11/2022 | 7 |
| 6.2 | Implementación de mecanismos de seguridad en contenedores | 16/11/2022 | 28/11/2022 | 12 |
| **7** | **Diseño seguro de orquestadores** | 28/11/2022 | 06/12/2022 | 8 |
| 7.1 | Análisis de orquestadores de contenedores | 28/11/2022 | 01/12/2022 | 3 |
| 7.2 | Seguridad a nivel de orquestador | 01/12/2022 | 06/12/2022 | 5 |
| 7.3 | **Hito** PEC 3 | 09/11/2022 | 06/12/2022 | 27 |
| **8** | **Estudio de plataformas cloud** | 07/12/2022 | 12/12/2022 | 5 |
| 8.1 | Análisis de plataformas Cloud | 07/12/2022 | 10/12/2022 | 3 |
| 8.2 | Investigar plataforma Cloud elegida | 10/12/2022 | 12/12/2022 | 2 |
| **9** | **Prueba de concepto** | 12/12/2022 | 08/01/2023 | 27 |
| 9.1 | Implementación de los contenedores | 12/12/2022 | 25/12/2022 | 13 |
| 9.2 | Securización de los contenedores | 25/12/2022 | 04/01/2023 | 10 |
| 9.3 | Análisis de los resultados obtenidos | 04/01/2023 | 08/01/2023 | 4 |
| **10** | **Conclusiones y trabajos futuros** | 08/01/2023 | 10/01/2023 | 2 |
| 10.1 | Elaboración de las conclusiones | 08/01/2023 | 09/01/2023 | 1 |
| 10.2 | Redacción del trabajo futuro | 09/01/2023 | 10/01/2023 | 1 |
| 10.3 | **Hito** PEC 4 | 07/12/2022 | 10/01/2023 | 34 |
| **10** | **Presentación en vídeo** | **11/01/2023** | **17/01/2023** | 6 |
| 10.1 | Preparación del vídeo | 11/01/2023 | 12/01/2023 | 1 |
| 12.2 | Grabación del vídeo | 12/01/2023 | 17/01/2023 | 5 |
| **11** | **Defensa** | **23/01/2023** | **27/01/2023** | 4 |
| 11.1 | Preparación de la defensa | 23/01/2023 | 27/01/2023 | 4 |

Fuente: Elaboración propia

En la siguiente página se muestra el diagrama de Gantt de las tareas.

**Figura 1 Diagrama de Gantt que muestra la ejecución de las tareas para el correcto desarrollo del trabajo Fin de Máster**



## 1.7. Breve sumario de productos obtenidos

Tras finalizar todas las entregas programadas, los productos obtenidos serán los siguientes:

- Análisis de la seguridad en contenedores.
- Diseño seguro de un sistema basado en contenedores.
- Validación de la securización del diseño seguro de un sistema basado en contenedores mediante el desarrollo de una prueba de concepto.

## 1.8. Breve descripción de los otros capítulos de la memoria

Los capítulos en los que se divide el presente trabajo Fin de Grado se encuentran detallados a continuación:

- **Análisis de amenazas en un sistema de información:** Donde se investigan las amenazas a las que se enfrenta un sistema de información que se encuentra accesible desde internet.
- **Análisis sistema actual:** Se comenta el sistema de información sobre el cual trabajaremos durante todo el desarrollo del trabajo Fin de Máster.
- **Aplicaciones y contenedores:** En esta sección se analiza el funcionamiento de los contenedores y su uso para albergar aplicaciones.
- **Diseño seguro en contenedores:** Nos centraremos en aplicar medidas de seguridad sobre contenedores.
- **Plataformas cloud:** En este apartado se analizan las plataformas cloud existentes y sus ventajas respecto a sistemas locales.
- **Prueba de concepto:** Donde aplicamos de forma práctica todo lo estudiado anteriormente.
- **Conclusiones y trabajo futuro:** Se reflejan las conclusiones a las que hemos llegado y se propondrán futuras mejoras y aplicaciones del proyecto.

## 1.9. Recursos necesarios y presupuesto del proyecto

En la siguiente tabla se detallan los recursos y sus costes asociados para la realización del trabajo Fin de Máster.

**Tabla 3 Presupuesto para realizar el proyecto**

| Recursos | Descripción | Coste |
|---|---|---|
| Ordenador | Se necesita un ordenador en el cuál poder realizar el trabajo Fin de Máster incluyendo la memoria y los casos prácticos que se desarrollarán | 800€-1200€ |
| Microsoft 356 | Software de ofimática para poder realizar la memoria | Al pertenecer a la comunidad educativa su coste es de 0€, pero si no fuese así el coste aproximado es de 69€ por |

| | | |
|---|---|---|
| | | año |
| Windows | Sistema operativo usado | Al pertenecer a la comunidad educativa su coste es de 0€, pero si no fuese así el coste aproximado es de 145€ desde la página oficial |
| Software libre | Para el desarrollo del trabajo usaremos otros programas como Virtual Box y un sistema operativo Linux como Ubuntu. | Gratis |
| Costes varios | En este apartado se incluye el coste asociado a internet, luz, agua, gas, etc. | 200€ |
| | TOTAL | 1500€ |

Fuente: Elaboración propia

## 1.10. Análisis de riesgos

Existen una serie de riesgos que pueden poner en peligro la elaboración del trabajo Fin de Máster:

**Riesgo 1: Longitud del trabajo demasiado extenso**
**Descripción:** Existe un riesgo por el cual no se lograría terminar en su totalidad el trabajo Fin de Máster debido a un intento de abarcar demasiadas cosas y abordar temas que se salen del alcance de este.
**Mitigación:** Se debe ser consciente del tiempo que se tiene para el desarrollo del trabajo y planificar en una fase temprana del mismo los hitos que se quieren cumplir.

**Riesgo 2: Problema a la hora de implementar el caso práctico**
**Descripción:** Pueden surgir multitud de problemas a la hora de desarrollar el caso práctico debido a que no se esté familiarizado con la tecnología propuesta o se intente realizar una configuración demasiado compleja.
**Mitigación:** Acotar las herramientas y la configuración que se realizará de cada una de ellas.

**Riesgo 3: Sistema de información demasiado complejo**
**Descripción:** A la hora de implementar el sistema de información para comenzar la prueba de concepto hay que tener especial cuidado con la dimensión de dicho sistema, ya que un sistema demasiado complejo dificultaría su transformación a un sistema basado en contenedores.
**Mitigación:** Debemos ser conscientes del tiempo que tenemos para desarrollar el trabajo por lo que el sistema de información debe ser muy sencillo de construir ya que el objetivo del trabajo TFM no es el de desarrollar un sistema desde cero.

**Riesgo 4: Desastres naturales y/o guerras**
**Descripción:** Hay ciertas circunstancias que no podemos controlar como son los desastres naturales y las guerras, pero no por ello debemos de ignorarlas.

**Mitigación:** Hacer una planificación en la cual se cuente con cierto tiempo de margen para que en el caso de encontrar alguna situación mayor que imposibilite su elaboración durante un período de tiempo determinado, el desarrollo de este no se vea afectado.

# 2. Análisis de riesgos y amenazas de seguridad

## 2.1. Definición y contexto

Una **amenaza es un hecho potencialmente peligroso** para el sistema que puede ocurrir, y de ser así provocaría daños sobre este. No todas las amenazas tienen el mismo nivel de importancia o el mismo origen, por lo que en algunos casos es óptimo no implementar salvaguardas para estas (Ministerio de Hacienda y Administraciones Públicas, 2012).

En la actualidad cualquier empresa por pequeña que sea tiene un sistema de información para poder desempeñar su trabajo, ya sea ofreciendo servicios de e-commerce, gestionando su inventario o administrando su plantilla. Esto quiere decir que la cantidad de **sistemas de información** existentes y **que presentan amenazas críticas es muy elevado**.

Estos problemas de seguridad no afectan solo a las empresas dueñas de estos sistemas, también afectan a los clientes o usuarios de los servicios que ofrecen dichas empresas, ya que si se produce un ataque por parte de un cibercriminal y este logra tener acceso a los datos del sistema **la información de los clientes y usuarios puede verse comprometida**.

Si se realiza una búsqueda rápida de ciberataques ocurridos en España podemos darnos cuenta de que se producen una gran cantidad de denuncias debido a este motivo, un ejemplo lo encontramos en un proveedor de Orange el cual sufrió un incidente de seguridad y datos sensibles de clientes (nombre, apellidos, DNI, IBAN…) fueron comprometidos con el riesgo que eso conlleva (Salcedo, 2022).

Por otro lado, el SEPE (Servicio Público de Empleo Estatal) también sufrió un ciberataque provocando en este caso el colapso de los servicios de ayuda. Esto hizo que muchas personas se quedasen sin recibir su prestación (Bailón, 2021).

Es por ello por lo que los sistemas de información actuales deben tener especial cuidado con las amenazas críticas y **proteger adecuadamente la información** que tienen almacenada para evitar en la medida de lo posible una fuga de información.

## 2.2. Tipos de amenazas

A continuación, detallaremos los tipos de amenazas existentes según su origen (Ministerio de Hacienda y Administraciones Públicas, 2012):

- **De origen natural:** Son amenazas causadas por la naturaleza, como terremotos, inundaciones, etc.
- **De origen industrial:** Este tipo de amenazas tiene su origen en la industria como puede ser la contaminación, apagones eléctricos, etc.

- **Defectos de las aplicaciones:** Estas amenazas son producidas por un defecto a la hora de implementar un sistema, dichos defectos son llamados vulnerabilidades y pueden comprometer el sistema.
- **Causadas por las personas de forma accidental:** Las personas que tienen acceso al sistema pueden cometer errores y ocasionar un mal funcionamiento de este, como el borrado accidental de registros en una base de datos.
- **Causadas por las personas de forma deliberada:** Las personas con acceso al sistema pueden, de forma intencionada, eliminar registros de la base de datos como venganza.

Durante el desarrollo del presente trabajo Fin de Máster nos centraremos en los 3 últimos tipos de amenazas. A continuación, se detalla un breve ejemplo de cada una de las tipologías de amenazas.

- **Defectos de las aplicaciones**: Respecto a la tipología de **defectos de las aplicaciones** nos podemos encontrar con imágenes cuyas dependencias tienen vulnerabilidades, lo cual produciría un incidente de seguridad.

- **Causadas por las personas de forma accidental:** Por otro lado, las **amenazas provocadas por personas de forma accidental** abarcarían por ejemplo un fallo no intencionado de implementación o por una mala configuración del sistema host o de la herramienta que se esté usando para gestionar los contendores (Docker o Podman).

- **Causadas por las personas de forma deliberada:** Por último, se encuentran las **amenazas provocadas por personas** de forma deliberada, la cual consistiría en un cibercriminal que intentaría explotar alguna vulnerabilidad del sistema o en el uso de **malware** (virus, gusano, etc) que se ejecutaría en el sistema provocando algún perjuicio al usuario.

## 2.3. Análisis de las amenazas en sistemas de información

Existen una infinidad de amenazas que afectan a un sistema de información, pero en este apartado nos centraremos en las más importantes que se detallan en el documento OWASP TOP 10 (OWASP, 2022):

- **Pérdida de control de acceso:** Esta amenaza provoca que una persona consiga acceder a sitios de forma no autorizada.
- **Fallas criptográficas:** Si no se cifran bien los datos sensibles como cuentas bancarias, registros médicos, etc. estos pueden ser leídos con facilidad por cibercriminales.
- **Inyección:** Esta amenaza provoca la ejecución de scripts por parte del atacante en la base de datos.

- **Diseño inseguro:** En este caso nos centramos en la seguridad vista desde el punto de vista arquitectónico y del propio diseño del aplicativo.
- **Configuración de seguridad incorrecta:** Este riesgo se basa en la incorrecta configuración del aplicativo por ejemplo al hacer uso de cuentas predeterminadas.
- **Componentes vulnerables y desactualizados:** Este riesgo se produce al utilizar librerías antiguas o software de terceros con vulnerabilidades.
- **Fallos de identificación y autenticación:** Este caso es provocado por una mala política de seguridad al administrar los usuarios como permitir contraseñas débiles, almacenar las credenciales en texto plano, no usar un 2FA, etc.
- **Fallos en el software y en la integridad de los datos:** Esto se produce cuando no tenemos un control sobre software que usamos en nuestro aplicativo de terceros y estos provienen de fuentes no confiables. Esto puede producir la creación de nuevas vulnerabilidades.
- **Fallos en el registro y monitoreo:** Este riesgo consiste en la no detección de una brecha de seguridad motivado por la no monitorización y análisis de registros del sistema.
- **Falsificación de solicitudes del lado del servidor:** Este riesgo tiene su origen cuando la aplicación web obtiene un recurso remoto sin realizar una validación de la URL proporcionada por el usuario.

Como podemos observar, existen una gran cantidad de amenazas que debemos tener en cuenta a la hora de construir un sistema de información.

A pesar de que este trabajo Fin de Máster se centra en la seguridad vista desde la perspectiva de un contenedor no podemos olvidar que dentro de dicho contenedor tenemos un aplicativo web, el cual puede sufrir los ataques mencionados en el OWASP TOP 10.

## 2.4. Análisis de las amenazas en contenedores

Además de las amenazas que existen sobre un sistema de información, al hacer uso de una arquitectura basada en contenedores existen otra serie de amenazas que afectan directamente a esta arquitectura. En este apartado se detallan las amenazas a las que se tiene que enfrentar los contenedores y/o imágenes (Tara Seals, 2022; y, OWASP, 2022).

- **Privilegios del usuario:** Si el aplicativo que se encuentra dentro del contenedor **se ejecuta con permisos de root**, se le está dando al atacante una **herramienta para poder comprometer la seguridad** del contenedor y en el peor de los casos aprovechar una **vulnerabilidad** del **kernel** de Linux para poder acceder al host y de esta forma tener acceso a todos los contenedores que se estén ejecutando. Debido a esto es bueno seguir el principio de dar al usuario los privilegios mínimos necesarios.
- **Gestión de actualizaciones:** Las aplicaciones/servicios que se estén usando dentro del contenedor por regla general tendrán vulnerabilidades

que son parcheadas con actualizaciones, debido a esto tenemos que asegurarnos de que todo el software del contenedor se encuentre actualizado ya que de lo contrario podría convertirse en un vector de ataque.

- **Segmentar red y cortafuegos:** Se deben exponer los servicios de contenedores que serán consumidos y no todos, además es buena práctica tener la red segmentada para de esta forma aislar a ciertos contenedores de un sistema. De no aplicar estas medidas un atacante podría comprometer todos los contenedores del host con más facilidad.
- **Configuración predeterminada:** Se debe realizar una configuración segura y hacer uso solo de los componentes necesarios para el funcionamiento del contenedor, de no ser así podríamos usar componentes que no necesitamos pero que tienen alguna vulnerabilidad que puede ser explotada por un atacante.
- **Contextos distintos en el mismo host:** Se debe mantener una diferencia notoria a la hora de hacer uso de varios contextos como Desarrollo, Preproducción y Producción, esto quiere decir que en un mismo host no debe coexistir más de un contexto ya que de ser así podríamos estar creando una puerta trasera a producción.
- **Proteger los secretos:** Debemos proteger adecuadamente cualquier token, clave privada, contraseña, etc. Ya que si un atacante logra acceder a dichos datos tendrá acceso a información y/o servicios privados.
- **Protección de los recursos:** Todos los contenedores de un mismo host comparten los recursos de cpu, ram, red y discos, esto quiere decir que un contenedor de forma accidental o intencionadamente puede agotar todos los recursos del host, debido a este problema es una buena práctica limitar los recursos de los que dispone cada contenedor.
- **Integridad y origen de la imagen del contenedor:** A la hora de montar un contenedor debemos asegurarnos de que la imagen que usamos no haya sido manipulada y que su origen es el correcto ya que si desplegamos una imagen malintencionada podemos poner en riesgo todo el host y llegar a afectar a todos los contenedores que se estén ejecutando.
- **Seguir el paradigma inmutable:** Debemos priorizar el uso de contenedores de solo lectura ya que de esta forma limitaremos las acciones que pueden hacer y reduciremos la cantidad de vectores de ataque disponibles para el atacante.
- **Registros:** Se debe tener un sistema de recopilación de registros de los contenedores, para de esta forma analizar dicha información y poder detectar un posible ataque.

Tal y como acabamos de observar, los **contenedores ofrecen una forma mejorada de aislamiento a nivel de sistema operativo** a diferencia de un servicio tradicional que se debería complementar con otros métodos como **AppArmor o SELinux**.

Esto es debido a que cada contenedor tiene todo lo necesario para la ejecución de una aplicación en su interior, por lo que, gracias a esto, no

necesita tener acceso a las dependencias del host, **exceptuando el Kernel de Linux** el cual es compartido.

Esto provoca que las aplicaciones que se encuentren ejecutándose dentro de un contenedor tengan un nivel de **aislamiento superior** a aplicaciones tradicionales. No obstante, se debe destacar que a pesar de tener un mejor aislamiento los contenedores **no están exentos de riesgos** y siguen estando expuestos a ataques y a problemas de aislamiento.

Por otro lado, los **sistemas tradicionales basados en microservicios** se estarían ejecutando en el sistema host **sin estar aislados unos de otros** y compartiendo dependencias, librerías, recursos, etc. En este caso el aislamiento que tendrían está ligado a su **nivel de ejecución en el sistema host**.

# 3. Análisis de sistemas basados en contenedores

## 3.1. Definición y características de los contenedores

Los **contenedores** software son un tipo de **tecnología de virtualización**, eso quiere decir que son capaces de crear pequeñas “máquinas virtuales” donde **alojar aplicaciones**. El contenedor es capaz de proveer de todos los recursos necesarios para el correcto funcionamiento del aplicativo (NetApp, 2022; y, Caparrós, Cubero, Fernández y Guijarro, 2022).

**La principal diferencia entre un contenedor y una máquina virtual** la podemos encontrar en la forma en que tienen de ejecutarse.
Una máquina virtual necesita de un **hipervisor** para funcionar, además necesita de su propio sistema operativo lo cual las hace muy pesadas y lentas.

Por otro lado, los contenedores **son capaces de ejecutarse en el propio sistema operativo host** sin la necesidad de hacer uso de un hipervisor. Gracias a esto los contenedores son menos pesados y más rápidos a la hora de ejecutarse.

Otra característica de los contenedores es su dependencia del **núcleo de Linux** (**Kernel**) para su ejecución. Gracias a esta característica les permite tener un mayor rendimiento que una máquina virtual pero también provoca que al hacer uso del Kernel de Linux, **una vulnerabilidad sobre dicho núcleo** tendría **consecuencias en los contenedores**.

Gracias a estas características podemos enumerar las siguientes ventajas (NetApp, 2022; y, Caparrós, Cubero, Fernández y Guijarro, 2022):

- **Ligereza:** Los contenedores requieren de menos recursos para su funcionamiento.
- **Portabilidad:** Debido a que dentro del contenedor se encuentran todos los recursos/dependencias que el aplicativo necesita no necesitamos volver a configurar nada al ejecutar dicho contenedor en otra máquina con otro sistema operativo.
- **Agilidad:** Al ser tan portable y rápido de ejecutar permite realizar despliegues de aplicaciones de forma rápida y sencilla. Esto es un gran punto a favor si el equipo de desarrollo trabaja con metodologías agiles.
- **Replicación:** Se puede replicar un contenedor fácilmente y en pocos segundos.
- **Dependencias:** Gracias a los contenedores podemos aislar las dependencias de una aplicación del sistema host, como puede ser la versión de java.

Un sistema basado en contenedores consta principalmente de dos elementos, una imagen y un contenedor (Garzas, 2015):

- **Imagen:** Es una representación estática del servicio y de la configuración/dependencias del aplicativo a ejecutar.
- **Contenedor:** Son instancias en ejecución de una imagen, gracias a los contenedores podemos ejecutar imágenes y exponerlas en algún puerto.

Los pasos que se deben seguir para la implementación de un contenedor con **Docker** de forma adecuada son los siguientes:

1. Creación del fichero con la configuración o definición del servicio, comúnmente llamado Dockerfile.
2. Ejecución del fichero para la creación de la imagen.
3. Ejecución del contenedor haciendo uso de la imagen creada.

Por otro lado, si se opta por hacer uso de **Podman** los pasos a seguir serían muy parecidos (Ameijeiras, 2022):

1. Creación de un fichero de configuración para la creación de la imagen. En este caso se podría usar el mismo que el Dockerfile pero modificando algunos comandos.
2. Ejecución del fichero para la creación de la imagen.
3. Ejecución del contenedor haciendo uso de la imagen creada.

Por otro lado, al contrario que Docker, **Podman** tiene una característica que lo hace especial y es la capacidad de **poder ejecutar Pods** tal y como hace **Kubernetes**.

Existen multitud de herramientas para manejar contenedores entre las que destacan **RKT, LXC/LXD, Podman o Docker**.

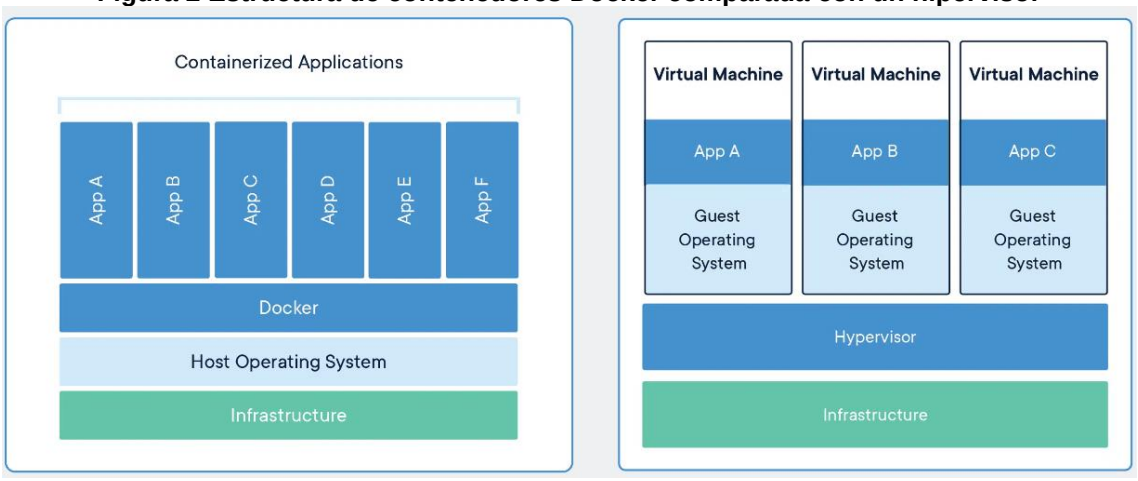
Para el desarrollo del presente TFM se ha optado inicialmente por el uso de **Docker y Podman** ya que son las herramientas más populares entre las empresas.

## 3.2. Docker

Una de las herramientas más populares y usadas es Docker, gracias al Daemon que tiene seremos capaces realizar solicitudes al Docker Engine API para de esta forma poder administrar nuestros contenedores.

Para saber mejor como está montada la estructura que usa Docker, podemos ver en la siguiente figura una comparación entre cómo se realiza la virtualización usando contenedores (Docker) y usando un hipervisor.

**Figura 2 Estructura de contenedores Docker comparada con un hipervisor**




Fuente: (Docker, 2022)

Tal y como podemos visualizar en la figura superior, Docker es capaz de ejecutar varios contenedores con sus respectivas aplicaciones sin la necesidad de usar un hipervisor o un nuevo sistema operativo.

Debido a todas estas ventajas que se han ido mencionando, la tecnología de contenedores es muy usada para sistemas que se basan en una arquitectura orientada a microservicios, ya que permite un despliegue muy rápido y aislado de cada uno de los distintos servicios que componen dicho sistema.

No obstante, esta tecnología no está libre de problemas de seguridad como puede ser la ejecución del demonio de Docker y/o contenedores con permisos de root por lo que se requiere de una configuración adicional para poder utilizarla de forma segura.

## 3.3. Podman

Podman es otra herramienta la cual está en auge y al igual que Docker nos ayuda a administrar nuestros contenedores, pero en este caso es **daemonless** por lo que esto ayuda a optimizar los recursos del sistema host (Podman, 2019).

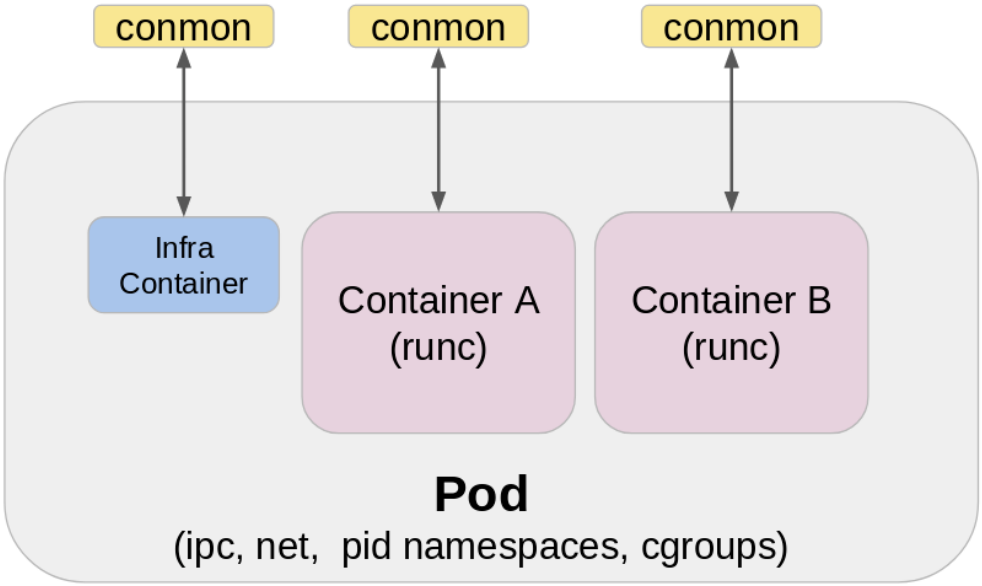
Además, Podman utiliza los mismos comandos que Docker para gestionar imágenes y/o contenedores por lo que pasar de un sistema Docker a un sistema basado en Podman es relativamente sencillo de hacer.

Una peculiaridad que tiene Podman es su capacidad de ejecutar contenedores **sin hacer uso del usuario root**, es decir, un usuario sin estos privilegios podría ejecutar un contenedor sin problema.

Por otro lado, aparece un nuevo término llamado **Pod**, los Pods son grupos de contenedores que se ejecutan juntos y son capaces de compartir los mismos recursos de una forma muy similar a como lo hace Kubernetes (Bause, 2019).

En la siguiente figura podemos visualizar la estructura de un Pod.

**Figura 3 Diagrama explicativo de un pod en Podman**




Fuente: (Bause, 2019)

Cada pod tiene un su interior un contenedor llamado **Infra Container** encargado de administrar el pod y permitir que Podman sea capaz de conectar otros contenedores.

Por otro lado, tenemos los "**common**", que son los encargados de monitorizar el estado de cada uno de los contenedores.

A nivel de seguridad un pod proporciona una **nueva capa de seguridad mejorando el aislamiento** del sistema del host.

## 3.4. Docker vs Podman

Tanto Podman como Docker son dos buenas herramientas para la administración de los contenedores pero que tienen distintas características.

En este capítulo detallaremos cuáles son esas diferencias (Dawson, 2022; Figueiredo y Gamela, 2021; y Wilson, 2021).

**Arquitectura**

**Docker** tiene un **demonio** (programa que se ejecuta en segundo plano) que sirve para la creación tanto de imágenes como de contenedores y está basado en una arquitectura cliente-servidor. En cambio, **Podman** no necesita usar un

demonio ya que su arquitectura se basa en un proceso único para administrar contenedores e imágenes.

**Privilegios root**

Podman es **rootless** lo que quiere decir que puede ejecutar contenedores sin la necesidad de darle permisos de root. Docker por defecto **ejecuta el Daemon otorgándole permisos de root**, no obstante, se puede llegar a configurar para que esto no sea así y de esta forma mitigar en cierta medida posibles vulnerabilidades.

**Modular vs Monolítico**

Mientras que con el **demonio de Docker** se pueden realizar todas las acciones que necesitemos como es la creación de imágenes, ejecución de contenedores, etc, en **Podman** tenemos una serie de módulos que realizan dichas acciones como **Buildah** que se encarga de generar los contenedores o el **runc** cuya función es la de ejecutar los contenedores.

**Seguridad**

Lo que respecta a la seguridad de ambas herramientas, Podman parte con una configuración predeterminada más segura debido a ser rootless o al evitar el uso de un Daemon tal y como si hace Docker, no obstante Docker puede ser configurado para que, por ejemplo, sus contenedores no usen un usuario con permisos de root.

Por otro lado, Podman tiene integración con SELinux (módulo de seguridad para el kernel de Linux) lo cual le provee de una mayor robustez.

**Volumenes**

Podman es capaz de acceder a volúmenes sin la necesidad de hacer uso de privilegios de root, esto hace que Podman en este aspecto sea más seguro que su homólogo Docker, ya que este último necesita dar permisos de root para poder acceder al directorio en el que se está almacenando el volumen.

Un problema de seguridad que nos podemos encontrar es cuando se hace uso de los **bind mounts** ya que en este caso estaríamos compartiendo ficheros del sistema host con el contenedor. Esto quiere decir que el contenedor puede obtener acceso a los archivos del host con el posible problema de seguridad que eso conlleva (Baretto, 2017).

Tal y como se ha comentado, **Docker y Podman** son herramientas cuya función es la de administrar las imágenes y contenedores. A pesar de tener algunas diferencias a la hora de gestionar las imágenes y contenedores **ambas son soluciones válidas** para la implementación de un sistema de información siempre y cuando **se configuren adecuadamente** y se adapten al contexto de seguridad requerido.

# 4. Sistemas basados en orquestadores

## 4.1. Introducción

A la hora de gestionar un sistema de información basado en contenedores, **se puede llegar a realizar de una forma más sencilla** que con sistemas de información tradicionales, esto es así, ya que cada uno de los servicios se encuentra aislado en un contenedor, facilitando de esta forma su manejo.

No obstante, cuando el sistema consta de multitud de contenedores, **realizar la gestión** de estos de forma manual **se vuelve una tarea muy costosa**, la cual no está exenta de **posibles errores de configuración**.

Para los casos en los cuales el sistema comienza a tener una complejidad elevada **existen herramientas de orquestación** que facilitan dicha tarea (Isaac, 2021; y Stackscale, 2021).

El uso de orquestadores tiene las siguientes ventajas (Isaac, 2021; y Stackscale, 2021):

- **Reducir la complejidad** a la hora de gestionar los contenedores.
- **Aumenta la seguridad** ya que se reducen los errores humanos gracias a la automatización de ciertas tareas.
- **Permite reiniciar y escalar los contenedores** de forma automática.
- **Mejora la distribución del tráfico** entre los distintos contenedores.

Por otro lado, las desventajas derivadas del uso de estos orquestadores son las siguientes (Isaac, 2021; y Stackscale, 2021):

- Las **migraciones** a estos sistemas son **más complejas**.
- El proceso de **instalación y posterior configuración es más complejo**.
- La **implementación de un clúster** de contenedores de forma manual es **compleja**.
- **Pueden existir vulnerabilidades propias de la herramienta** de orquestación.

A la hora de elegir una herramienta para la orquestación de contenedores no existen demasiadas alternativas, actualmente podemos elegir entre **Docker Swarm** y **Kubernetes** pero es esta última alternativa la que tiene mayor potencial y popularidad por lo que será en la que nos centraremos.

**La implementación de estas herramientas** de orquestación **queda fuera del alcance del actual proyecto**, por lo que será agregado al apartado de trabajos futuros.

## 4.2. Kubernetes

### *4.2.1. Descripción*

**Kubernetes (K8s)** es una **plataforma de código abierto** que se encarga de automatizar la implementación, el escalado y la administración de los contenedores de un sistema (Kubernetes, 2022).

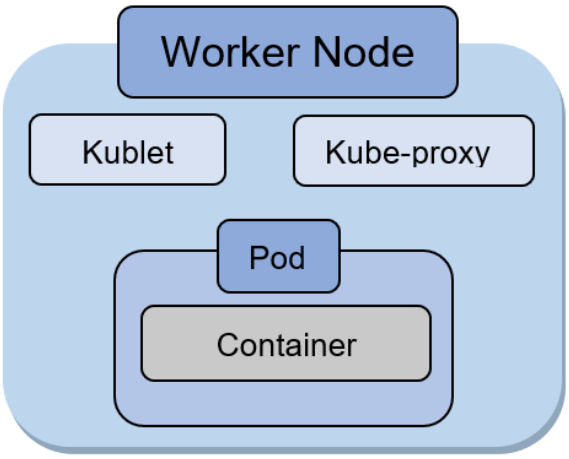
Por otro lado, la gran popularidad de Kubernetes permite la **migración sencilla** desde un **sistema on-premise** o entre distintas **plataformas de servicios Cloud** como AWS, Microsoft Azure o Google Cloud.

### *4.2.2. Arquitectura de Kubernetes*

La estructura de Kubernetes está organizada mediante distintos tipos de nodos, los cuales pueden ser **Worker Node** o **Master Node**.

Los **Worker Node** son los encargados de ejecutar las aplicaciones en uno o varios **pods**, además dentro del mismo worker node existen varios componentes como son el **Kublet** y el **Kube-proxy.**

**Figura 4 Estructura de un worker node en Kubernetes**



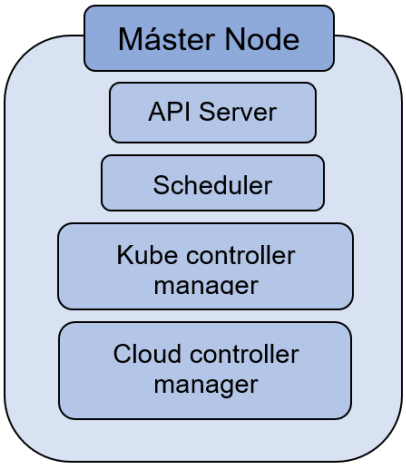

Fuente: Elaboración propia

Cada uno de los componentes mostrados en la ilustración superior tienen distintas funciones.

- **Pod:** Un pod es un grupo de uno o más contenedores que comparten entre ellos los recursos.
- **Kublet:** Se encarga de administrar los pods dentro del worker node y es capaz de comunicarse con el master node.
- **Kube-proxy:** Se encarga de gestionar la red del worker node.

Por otro lado, los **Master Node** son los encargados de administrar los worker node. A continuación, se muestra un master node con sus componentes:

**Figura 5 Estructura de un máster node en Kubernetes**




Fuente: Elaboración propia

- **API server:** Recibe peticiones/comandos para de esta forma poder administrar el clúster.
- **Schedule:** Se encarga de asignar cada uno de los pod a un worker node, identificando el ideal.
- **Kube controller manger:** Se encarga de que el estado actual del clúster sea el estado ideal, es decir, tenga cierta cantidad de pods o instancias levantadas de un servicio.
- **Cloud controller manager:** Sirve para interactuar con un Cloud provider API, es decir, permite ejecutar Kubernetes en servicios cloud como AWS.

Tanto los **worker node** como los **master node** se encuentran dentro de un **clúster**, que es una red que agrupa todos los componentes para que de esta forma se puedan comunicar unos con otros sin problemas.

Además, toda la configuración de Kubernetes y el estado final deseado se almacena en una base de datos llamada **Etcd** lo que facilita la migración del sistema clúster a un proveedor de la nube.

### *4.2.3. Seguridad en Kubernetes*

**Kubernetes** como herramienta software **no está exenta de posibles fallos de seguridad**, **vulnerabilidades o fallos en su configuración**; es por eso por lo que en este apartado se comentará brevemente los aspectos más importantes a tener en cuenta a la hora de desarrollar un sistema de información haciendo uso de la plataforma Kubernetes.

Antes de comenzar con problemas exclusivos de Kubernetes, se debe destacar que **las mismas amenazas que se aplican a sistemas basados en contenedores son aplicadas en sistemas Kubernetes** ya que esta plataforma usa contenedores para su funcionamiento. Esto quiere decir que se

debe tener en cuenta las amenazas descritas en el apartado 2.4. Análisis de las amenazas en contenedores.

A continuación, se detallan distintos tipos de amenazas y una posible solución o mitigación (Avi, 2022; Kubernetes, 2022; y RedHat, 2022).

- **Errores en a la configuración:** Kubernetes usa fichero yml para su configuración, un error en alguno de ellos puede provocar una brecha de seguridad, es por lo que se recomienda el uso de **herramientas para poder detectar malas configuraciones**, algunas de estas herramientas son **kubescape** o **kubesec**.

- **Seguridad en la red:** Es recomendable aplicar distintas **políticas de red tanto de entrada como de salida** para controlar el tipo de tráfico permitido, así como cifrar las comunicaciones dentro del clúster de Kubernetes.

- **Restringir el acceso a componentes:** Se debe **evitar** que se pueda **acceder a ciertos componentes** de Kubernetes como es el caso de la API (**API Server**), ya que desde este componente se ejecutan comandos que pueden modificar la configuración de todo el clúster provocando un problema de seguridad.

- **Autenticación: Activar la autenticación para acceder a los componentes** del cluster como por ejemplo Kube controller manager, además el certificado raíz debe estar protegido adecuadamente y los certificados medios deben tener una fecha de expiración no superior a 3 años.

- **Permisos de un pod:** Un pod debe **tener solo los permisos estrictamente necesarios** para su correcto funcionamiento, es decir, si un pod solo consulta a una base de datos, únicamente debe tener el permiso de lectura y no el de modificación.

Tal y como se puede observar al hacer uso de Kubernetes **aparecen otras amenazas además de las propias del uso de contenedores,** que se deben tener en cuentan a la hora de desarrollar el sistema de información. Estas amenazas se deben tener en consideración y mitigarlas en la medida de lo posible para evitar problemas de seguridad en el futuro.

# 5. Sistemas basados en arquitecturas cloud

Hoy en días las plataformas cloud gozan de una gran popularidad y son una gran alternativa a la hora de alojar nuestro sistema de información.

Es por eso por lo que en este apartado analizaremos las distintas alternativas que tenemos en el mercado y cuáles son los puntos fuertes y débiles de optar por una solución en el cloud.

## 5.1. Definición

El **Cloud** o **Cloud Computing** es un modelo que permite el acceso bajo demanda y a través de Internet a un conjunto de **recursos informáticos**. Estos recursos informáticos son gestionados por un proveedor y son ofrecidos al usuario (Suppi, 2022).

Dependiendo del tipo de servicio que se ofrece existen 3 categorías de entornos Cloud (Suppi, 2022).

- **Infraestructura como servicio (IaaS):** En este tipo de cloud el proveedor proporciona los recursos informáticos como el almacenamiento, red, servidores en los cuales el cliente podrá instalar su SO y sus aplicaciones, es decir, el cliente alquila la infraestructura que necesita.
- **Plataforma como servicio (PaaS):** El proveedor proporciona al usuario un entorno en el cual se puede desarrollar, gestionar y desplegar aplicaciones. El cliente en este caso no gestiona ni controla la infraestructura, pero tiene el control de las aplicaciones que en ella se despliegan.
- **Software como servicio (SaaS):** El proveedor proporciona al cliente una aplicación completamente configurada y lista para su uso. En este caso el usuario no tiene control sobre ellas.

**La implementación de los servicios cloud** mencionados en el presente apartado **queda fuera del alcance del actual proyecto**, por lo que será agregado en la sección de trabajos futuros.

## 5.2. Ventajas e Inconvenientes

A continuación, se detallarán las principales ventajas e inconvenientes que tenemos al hacer uso del Cloud para desplegar nuestro sistema (Suppi, 2022).

**Ventajas**

- **Agilidad y autoservicio:** El cliente puede obtener una gran cantidad de recursos informáticos de forma rápida y sencilla, además cuando no los necesite los puede liberar sin tener que realizar ninguna configuración.

- **Escalabilidad y elasticidad:** El cliente puede administrar los recursos informáticos para que en momentos de máxima carga se obtengan más recursos y cuando la carga disminuya dichos recursos de vayan liberando.

- **Coste:** Al hacer uso solamente de los recursos necesarios en los momentos necesarios los costes se reducen ya que si un recurso no está siendo usado puede ser liberado para de esta forma reducir los costes.

- **Seguridad:** Parte de la seguridad del sistema recaerá sobre el proveedor, el cual, al interesarle que su servicio sea seguro hará inversiones de dinero para mejorar la seguridad, esto es un gran punto a favor ya que hay empresas que no se pueden permitir el realizar una gran inversión en seguridad.

- **Reducción de riesgo y rapidez:** No se necesita realizar una gran inversión para comenzar y se reducen los riesgos ya que es el proveedor el que asumirá algunos de ellos como el de mantener la infraestructura segura.

**Desventajas**

- **Centralización:** Al tener todos los datos y las aplicaciones desplegadas en un proveedor cloud si dicho proveedor tiene algún problema y deja de estar disponible todo el sistema del cliente dejará de estar operativo.

- **Confiabilidad:** Se debe elegir un proveedor en el cual se confíe ya que dependeremos de él para que nuestro sistema funcione adecuadamente y las decisiones que dicho proveedor pueda tomar repercutirán en el cliente.

- **Disponibilidad:** Al hacer uso de una estructura cloud se necesita internet para poder acceder a ella, por lo que si el cliente tiene algún tipo de problema con su proveedor de internet este no tendrá ninguna forma de gestionar su sistema.

- **Riesgo y privacidad:** Los datos sensibles del sistema del cliente no se encuentran en sus instalaciones, en su lugar están almacenados en la base de datos del cloud y su seguridad depende del proveedor por lo que el riesgo puede ser demasiado alto.

- **Seguridad:** Debido a que la información del cliente debe pasar por distintos canales del proveedor, se produce en cada uno de ellos un

riesgo que puede desembocar en un fallo de seguridad. Además, en algunos casos el cliente no es consciente del fallo de seguridad y/o fuga de información.

- **Vendor lock-in:** Otro de los problemas derivados del cloud es la dependencia que se produce al hacer uso de un proveedor de cloud, ya que migrar todo el sistema a otro proveedor suele ser costoso y requiere de configuraciones adicionales.

## 5.3. Seguridad en el cloud y su responsabilidad

Al tener nuestro sistema en el cloud de un proveedor podemos llegar a **mejorar en algunos casos la seguridad** de este, ya que dicha responsabilidad no recaerá solo sobre el **cliente**, sino que el **proveedor** también tendrá que aplicar las **medidas de seguridad** que estime oportunas (Suppi, 2022).

Por otro lado, dependiendo del servicio que contratemos **la seguridad puede depender casi en su totalidad al proveedor** como por ejemplo si contratamos un servicio SaaS de correo electrónico.

Por ejemplo, no es lo mismo que el cliente alquile una máquina para **instalar manualmente su sistema**, que contratar un servicio de despliegue de contenedores como puede ser **Amazon ECS** (Amazon Elastic Container Service), ya que la seguridad en este último caso corresponderá en mayor medida a Amazon (AWS, 2022).

No obstante, no debemos olvidar que en el ejemplo anterior sería **Amazon** el que se encargaría de **mantener segura la infraestructura**, pero **correspondería al cliente mantener segura su aplicación**, ya que es posible que el aplicativo tenga una vulnerabilidad y esto provocaría un fallo de seguridad.

Como se ha comentado, la responsabilidad de **la seguridad no depende de un único actor** ya que en el caso del cloud se trata de un sistema de responsabilidad compartida. Dicha responsabilidad está **directamente correlacionada con el nivel de control** que tiene un actor sobre la arquitectura y/o servicio (Suppi, 2022).

Esto quiere decir que a **mayor nivel de control** tiene un actor **mayor será su nivel de responsabilidad en materia de seguridad**. En la siguiente tabla se puede ver una comparación de la responsabilidad que tiene el proveedor y el cliente dependiendo del servicio que dicho proveedor proporcione.

**Tabla 4 Responsabilidad dependiendo del tipo de servicio que ofrece el proveedor**

| | IaaS | PaaS | SaaS |
|---|---|---|---|
| Proveedor (responsabilidad) | Bajo | Medio | Alto |
| Cliente (responsabilidad) | Alto | Medio | Bajo |

Fuente: Elaboración propia

A la hora de utilizar el cloud para realizar el despliegue de un sistema de información basado en contenedores, se puede hacer uso de servicios (SaaS) que ofrecen los proveedores, pero el **cliente se tendría que adaptar a las herramientas que facilita el proveedor**, es decir, tendría que usar la plataforma de contenedores que el servicio cloud ofrece, las herramientas que el proveedor tiene, las versiones que utiliza el proveedor, etc.

Otra posibilidad sería hacer uso de un servicio del tipo **IaaS,** en este caso el **cliente tendría un mayor control** del SO que quiere instalar, las aplicaciones y la plataforma de contenedores que quiere usar. Por el contrario, también **tendría que aplicar medidas de seguridad sobre la infraestructura,** ya que en estos casos es **el cliente el que tiene mayor responsabilidad** que el proveedor.

Tal y como hemos mencionado, **a medida que el cliente tiene un mayor control sobre el sistema también tiene mayor responsabilidad** a lo que a seguridad se refiere. Esto quiere decir que es el cliente el que debe sopesar si verdaderamente necesita tener control total de la máquina que le proporciona el proveedor o si amoldarse al servicio que dicho proveedor le ofrece y adaptarse a la plataforma de contenedores que usa.

## 5.4. Plataformas cloud más populares

Debido a la gran popularidad que han ido tomando las plataformas cloud han aparecido multitud de proveedores ofreciendo dichos servicios, en el presente apartado hablaremos sobre 3 de las principales plataformas como son **AWS** (Amazon Web Services), **Microsoft Azure** y **Google Cloud**.

### Amazon Web Services

AWS es la plataforma cloud de Amazon, la cual goza de gran popularidad entre las empresas privadas ya que ofrece un sinfín de herramientas que se adaptan a las distintas necesidades de los clientes.

Algunos de los servicios que pueden ser utilizados para la implementación de un sistema basado en contenedores puede ser el **Amazon ECS** (Amazon Elastic Container Service). Este servicio proporciona una herramienta para la orquestación de contenedores la cual permite implementar, administrar y escalar el aplicativo (AWS, 2022).

Otro servicio interesante sería el **Amazon EKS** (Amazon Elastic Kubernetes Service), gracias a este servicio se puede ejecutar Kubernetes en la nube de AWS (AWS, 2022).

A nivel de seguridad AWS nos ofrece una serie de servicios que nos pueden ayudar, algunos de los cuales son (AWS, 2022):

- **Amazon GuardDuty:** Es un servicio de detección de amenazas el cual analiza las cargas de trabajo y las cuentas de AWS en busca de actividades maliciosas.
- **AWS WAF:** Es un cortafuegos para aplicaciones web y es capaz de proteger a las aplicaciones contra los principales ataques o mitigar en gran medida sus efectos.
- **Amazon Detective:** Este servicio recopila y analiza los datos procedentes de AWS para posteriormente poder realizar investigaciones más rápidas y eficaces.

## Microsoft Azure

Microsoft Azure es la plataforma **cloud de Microsoft**, la cual nos ofrece servicios para construir, desplegar y administrar los aplicativos del cliente (Azure, 2022).

Al igual que AWS, Azure ofrece sus propios servicios orientados a sistemas basados en contenedores, a continuación, definimos cada uno de ellos.

- **Azure Container Apps**: Es un servicio que nos permite administrar y desplegar aplicaciones basadas en contenedores.
- **Azure Kubernetes Service (AKS)**: Este servicio nos ofrece las herramientas necesarias para poder crear un sistema basado en contenedores pero que sea orquestado por Kubernetes.

A nivel de seguridad encontramos los siguientes servicios:

- **Azure Monitor:** Este servicio recopila y analiza los datos de los aplicativos para posteriormente detectar y diagnosticar posibles problemas que pueda tener la aplicación.
- **Application Gateway:** Gracias a este servicio las distintas aplicaciones tendrán un solo punto de acceso el cual podrá ser controlado con un firewall, balanceador de carga, SSL, etc.

## Google Cloud

Esta última plataforma cloud es la ofrecida por Google que al igual que las anteriores nos ofrece distintos servicios para poder llevar nuestro sistema de información a la nube (Google Cloud, 2022).

Los servicios que nos ofrece para poder administrar un sistema basado en contenedores son los siguientes:

- **Google Cloud Run:** Es un servicio que permite el despliegue de aplicaciones basadas en contenedores.
- **Google Kubernetes Engine (GKE):** Este servicio permite ejecutar clústeres de Kubernetes haciendo uso de la infraestructura de Google Cloud.

Desde el punto de vista de la seguridad Google nos ofrece los siguientes servicios para ayudarnos:

- **Web Security Scanner:** Este servicio permite detectar vulnerabilidades en los distintos servicios que estemos usando como Google Kubernetes Engine (GKE).
- **Cloud Key Management:** Este servicio nos ofrece un sistema de gestión de claves criptográficas seguro.
- **Cloud Monitoring:** Este servicio recopila datos, métricas y metadatos para posteriormente analizarlos y detectar posibles problemas.

A continuación, se muestra una tabla comparativa de las 3 plataformas Cloud comentadas.

**Tabla 5 Comparación de los servicios ofrecidos por AWS, Azure y Google Cloud**

| | AWS | Azure | Google Cloud |
|---|---|---|---|
| **Servicio para contenedores** | Amazon ECS | Azure Container Apps | Google Cloud Run |
| **Servicio para Kubernetes** | Amazon EKS | Azure Kubernetes Service | Google Kubernetes Engine |
| **Monitoreo** | Amazon CloudWatch | Azure Monitor | Cloud Monitoring |
| **Búsqueda de amenazas** | Amazon GuardDuty | Azure Active Directory Identity Protection | WAAP |
| **Búsqueda de vulnerabilidades** | Amazon Inspector | Azure Active Directory Identity Protection | Web Security Scanner |
| **Búsqueda de vulnerabilidades en imágenes** | Amazon ECR image scanning | Defender for Containers | Container Analysis |
| **Cortafuegos** | AWS Network Firewall | Azure Firewall | Google Cloud Firewall |
| **Administrar secretos** | AWS Secrets Manager | Key Vault | Secret Manager |

Fuente: Elaboración propia a partir de AWS (2022); Azure (2022); y Google Cloud (2022)

Tal y como podemos observar existen distintos proveedores que nos ofrecen servicios similares para que la migración de un sistema basado en contenedores a la nube sea relativamente sencilla.

Como se ha visualizado en la tabla comparativa, cada plataforma ofrece su servicio para proteger un elemento del sistema. Esto quiere decir que independientemente del cloud que se elija siempre se nos ofrecerá unos servicios de seguridad muy similares.

# 6. Análisis del sistema prototipo

## 6.1. Descripción del sistema

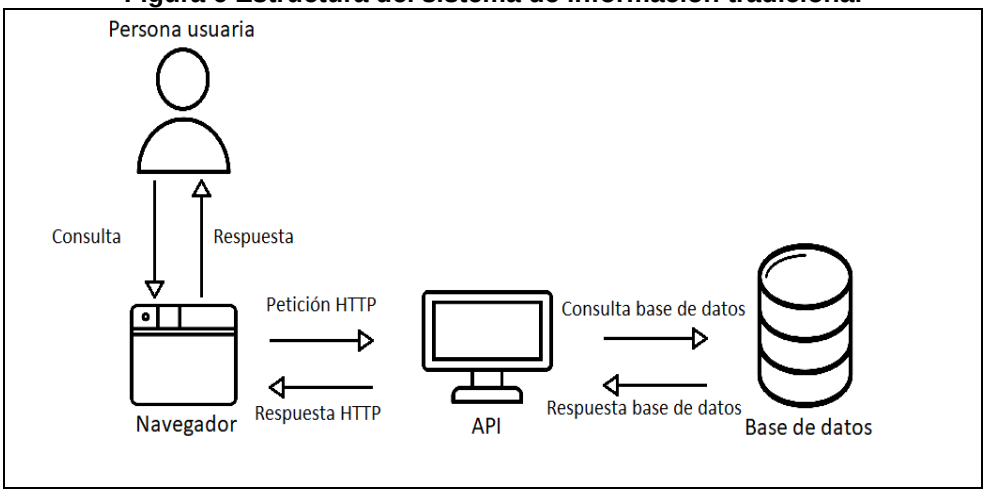
En esta sección describiremos un sistema típico de una empresa, el cual estará compuesto por una aplicación (API) que se conectará a una base de datos para proporcionar la información solicitada y/o guardar datos.

La empresa ficticia se llamará Acme-Shop y constará de un aplicativo para gestionar el inventario de productos que tiene almacenados. Esta información será guardada en una base de datos que estará conectada con la aplicación (API).

Se ha optado por elegir una **estructura basada en microservicios** ya que es la más usada por las empresas que tienen un sistema de información, por lo tanto, aportaremos mayor realismo a la prueba de concepto desarrollada con posterioridad.

En la siguiente figura podemos observar un esquema de la estructura seguida por el sistema de información, la cual se compone de los siguientes elementos.

**Figura 6 Estructura del sistema de información tradicional**




Fuente: Elaboración propia

- **Persona usuaria:** Persona que interactúa con el sistema de información.
- **Browser:** Navegador web o cliente HTTP que realiza peticiones a la API que provee el aplicativo.
- **API:** Aplicación que expone una API la cual será consultada por el cliente, además dicho aplicativo tendrá acceso a la base de datos para realizar las operaciones de CRUD.
- **Base de datos:** Plataforma en al cual se almacenarán los datos necesarios para el correcto funcionamiento del sistema de información.

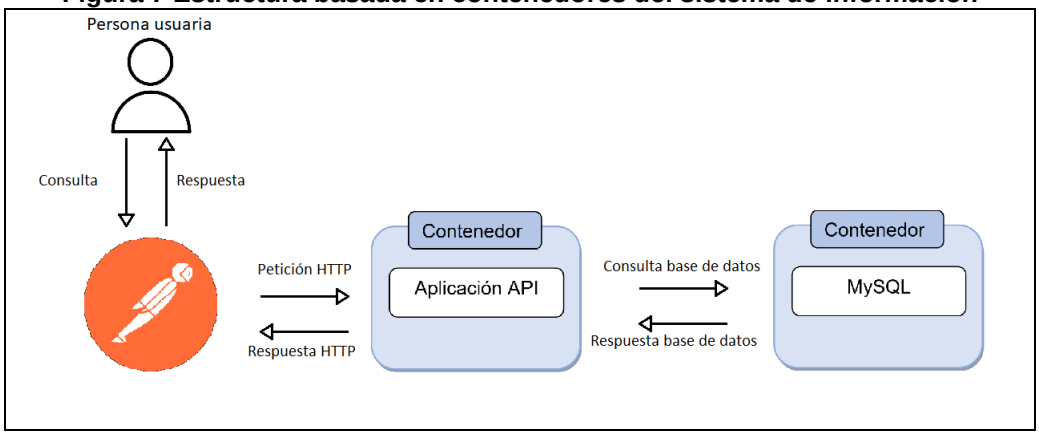
## 6.2. Sistema basado en contenedores

A continuación, realizaremos una implementación del sistema comentado en el apartado anterior, pero haciendo uso de contenedores. Para realizar dicha implementación se hará uso de Podman y Docker.

Para la conversión del sistema anteriormente mencionado a un sistema basado en contenedores haremos uso de dos contenedores, uno de ellos alojará nuestra aplicación API mientras que el otro tendrá la base de datos. Ambos contenedores se comunicarán para poder intercambiar información.

En la siguiente figura podemos visualizar cómo quedaría la nueva estructura haciendo uso de los contenedores:

**Figura 7 Estructura basada en contenedores del sistema de información**




Fuente: Elaboración propia

El sistema desarrollado tiene los siguientes elementos:

- **Persona usuaria:** Persona que interactúa con el sistema.
- **Postman:** Plataforma para realizar peticiones a la API del aplicativo.
- **Contenedor – Aplicación API:** Dicho contenedor contiene el servicio que se encarga de recibir y procesar las peticiones de las personas usuarias.
- **Contenedor – MySQL:** En este contenedor se ejecuta el servicio de base de datos utilizado que en este caso en concreto se trata de MySQL.

Cuando una persona interactúa con el sistema, el flujo que sigue es el siguiente:

1. La persona usuaria haciendo uso de **Postman realiza una petición a la API** que ofrece el contenedor “Aplicación API”.
2. **El contenedor “Aplicación API” procesa dicha petición** y si es necesario se conecta con el contenedor en el que se almacena la base de datos llamado “MySQL”.
3. **El contenedor que contiene el servicio de base de datos recupera la información** requerida y se la proporciona al servicio “Aplicación API”.

4. Finalmente, **el servicio "Aplicación API" contesta a la persona usuaria** con la información solicitada.

El sistema basado en contenedores contiene un único servicio por contenedor, esta filosofía nos proporciona numerosas ventajas como es la mejora del **aislamiento** de los servicios, lo que provoca un **aumento de la seguridad** de estos al no tener que interaccionar innecesariamente entre ellos para su correcto funcionamiento (Donohue, 2022).

Por otro lado, a nivel de diseño, al hacer uso de una arquitectura orientada a los microservicios logramos **mejorar la escalabilidad** del sistema, **mejorar su gestión**, **reutilización** y al tener servicios aislados es más fácil la **corrección de posibles errores** (Donohue, 2022).

Para conocer con más detalle la configuración usada para la implementación del sistema de información se puede acceder al anexo del presente documento (ver anexo Anexo I. Configuración del sistema de información).

## 6.3. Seguridad en el sistema de información basado en contenedores

A nivel de seguridad, un sistema de información basado en contenedores puede presentar fallos de seguridad que son convertidos en vectores de ataque para posteriormente comprometer el sistema.

Los fallos de seguridad se producen principalmente por una mala configuración de los contenedores o por utilizar la configuración por defecto.

A continuación, se detallan algunos de los mecanismos que podemos utilizar para la protección del sistema, aunque se verá con más detalle en la prueba de concepto.

- Aplicar una **política de mínimos privilegios** para el usuario o **rootless** en el caso de usar Docker. Esto quiere decir que se debe configurar tanto el demonio de Docker y la ejecución de contenedores para que no usen los privilegios del usuario root.
- Uso de herramientas para la **búsqueda de vulnerabilidades** ya que estas pueden haber sido introducidas al hacer uso de dependencias y/o servicios desactualizados.
  Otra forma muy común de que la imagen que se esté usando tenga vulnerabilidades, es porque **algunas de sus capas no haya sido actualizada,** por lo que los fallos de programación que tienen no han sido reparados.
- Aislar los contenedores con el uso de **AppArmor o SELinux**, para evitar la posible propagación del malware de un contenedor a otro.
- Uso de **cortafuegos** para evitar intentos de conexiones o escaneos de puertos.

- Uso de un **proxy inverso** para redirigir el tráfico hacia el endpoint del aplicativo. Al hacer uso de un proxy inverso se puede realizar un escaneo y/o petición de credenciales al usuario que se intenta conectar a nuestra API.
- **Protección de los recursos utilizados por los contenedores** para de esta forma evitar que un contenedor que no esté funcionando correctamente consuma todos los recursos del host en el que esté, provocando una ralentización de la ejecución de los otros contenedores que se estén ejecutando en el mismo host.
- Comprobación de la **integridad y origen de las imágenes** que usamos en nuestro sistema. Es muy común que se usen imágenes públicas para de esta forma ahorrar trabajo, pero estas imágenes pueden contener vulnerabilidades o ser imágenes creadas por cibercriminales.
- Uso de un **sistema de registros para monitorizar los contenedores** para de esta forma disminuir el tiempo de actuación ante una incidencia.

Existen otros mecanismos que se pueden utilizar a la hora de proteger el sistema de información, pero durante el desarrollo del presente trabajo Fin de Máster nos centraremos en los más importantes.

# 7. Prueba de concepto

## 7.1. Introducción

En el apartado actual se realizará una prueba de concepto aplicando mecanismos de seguridad al sistema basado en contenedores que se ha configurado y desarrollado en el punto Anexo I. Configuración del sistema de información (ver anexo).

Para ello el apartado se divide en distintas secciones en las cuales se irán aplicando medidas de seguridad para reducir o eliminar amenazas que afecten al sistema.

A continuación, se muestra un resumen de las distintas secciones:

- **Política de mínimos privilegios:** En este apartado hablaremos sobre los privilegios a la hora de ejecutar un contenedor y nos centraremos en el modo rootless del demonio de Docker y en el uso de usuarios sin privilegios de root a la hora de crear una imagen.

- **Aislamiento de contenedores:** En esta sección se mejorará el aislamiento de los contenedores y se evitará que puedan hacer ciertas llamadas a ciertos recursos del sistema host. Primero se hará uso del módulo de seguridad **AppArmor** para luego implementar definitivamente **Seccomp**.

- **Limitación de recursos:** En este caso aplicaremos restricciones de recursos a los contenedores haciendo uso de las herramientas que vienen ya implementadas tanto en Docker como en Podman.

- **Búsqueda de vulnerabilidades:** Para mejorar la seguridad de los contenedores en este apartado utilizaremos la herramienta **Trivy** para la búsqueda de posibles vulnerabilidades en las imágenes de los contenedores.

- **Implementación de cortafuegos:** En esta sección se implementará un firewall para proteger a los contenedores de conexiones no seguras. Para ello se hará uso de **UFW**, un cortafuegos que viene junto con la distribución de Ubuntu.

- **Proxy inverso:** Para proteger mejor el sistema de información del exterior en este apartado se implementará un proxy inverso con **Nginx**, el cual hará de filtro para detectar posibles conexiones no seguras al sistema. Además, se implementará un sencillo sistema de autenticación con **Http Basic Authentication.**

- **Uso de pods en Podman:** En este apartado se verá el uso de **pods con Podman** y como su uso mejora a nivel de seguridad el aislamiento del sistema basado en contenedores.

## 7.2. Política de mínimos privilegios

### *7.2.1. Demonio de Docker*

Esta política de privilegios es muy importante aplicarla ya que de no ser así le estaríamos dando al atacante armas con las que comprometer el sistema.

Uno de los principales problemas a la hora de hacer uso de Docker es su Daemon, ya que por defecto este se ejecuta con permisos de root lo cual es innecesario y puede provocar un problema de seguridad.

Debido a esto se debe configurar como rootless y para ello se debe ejecutar el comando "**dockerd-rootless-setuptool.sh install**" el cual nos habilitará la opción de poder ejecutar el demonio de Docker en modo rootless.

**Figura 8 Resultado tras la ejecución del comando "dockerd-rootless-setuptool.sh install"**


Fuente: Elaboración propia

Para obtener más detalles de dicha configuración se puede consultar los anexos del presente documento Política de mínimos privilegios.

Gracias a esta configuración conseguimos reducir el área de ataque de un usuario malintencionado.

Por otro lado, en el caso de Podman **no es necesario hacer ninguna configuración adicional**, ya que afortunadamente Podman no requiere de un demonio para su funcionamiento. Gracias a esta característica de Podman, dicha herramienta logra ser algo **más segura con la configuración que trae por defecto**.

### *7.2.2. Permisos dentro del contenedor*

Además de configurar el demonio de Docker para que funcione sin permisos de root se debe configurar el **contenedor** para usar un **usuario sin privilegios de root**. Este paso debemos hacerlo tanto en **Docker** como en **Podman** pero la forma de configurarlo es la misma en ambos casos.

Para poder ejecutar un contenedor sin privilegios de root se debe agregar el comando "**RUN adduser -D norootuser**" al fichero Dockerfile tal y como se puede observar en la siguiente figura.

**Figura 9 Modificarción del dichero Dockerfile para ejecutar el servicio con un usuario no root**

```
1 ARG MSVC_NAME=msvc-items
2 FROM openjdk:18-jdk-alpine as mavenBuilder
3 ARG MSVC_NAME
4
5 WORKDIR /app/$MSVC_NAME
6
7 COPY ./pom.xml /app
8 COPY ./$MSVC_NAME/.mvn ./.mvn
9 COPY ./$MSVC_NAME/mvnw .
10 COPY ./$MSVC_NAME/pom.xml .
11
12 RUN ./mvnw clean package -Dmaven.test.skip -Dmaven.main.skip -Dspring-boot.repackage.skip && rm -r ./target/
13
14 COPY ./$MSVC_NAME/src ./src
15
16 RUN ./mvnw clean package -DskipTests
17
18 FROM openjdk:18-jdk-alpine
19 ARG MSVC_NAME
20 WORKDIR /app
21 RUN mkdir ./logs
22 COPY --from=mavenBuilder /app/$MSVC_NAME/target/msvc-items-0.0.1-SNAPSHOT.jar .
23
24 RUN adduser -D norootuser
25 USER norootuser
26
27 EXPOSE 8001
28 CMD ["java", "-jar", "./msvc-items-0.0.1-SNAPSHOT.jar"]
```

Fuente: Elaboración propia

De esta forma cuando se cree la imagen y posteriormente el contenedor, éste se ejecutará con privilegios de usuario. Para más información al respecto se puede consultar el anexo Política de mínimos privilegios.

Una vez realizada la configuración tanto para Docker y/o Podman a la hora de ejecutar los contenedores estos lo harán haciendo uso de **usuarios sin privilegios de root** lo que aumentará la seguridad de dichos contenedores.

Gracias a esto si un contenedor ve su seguridad comprometida el cibercriminal no tendrá bajo su control un usuario root lo que **disminuirá** en gran medida **el alcance de su ataque**.

## 7.3. Aislamiento de contenedores

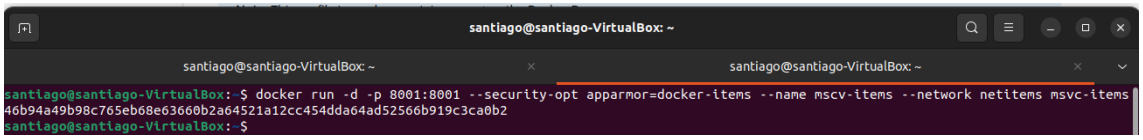
La tecnología de contenedores es capaz de aislar los servicios del sistema host de una forma sencilla, pero con solo usar dicha tecnología no se consigue un aislamiento suficiente.

Esto quiere decir que, a pesar de tener nuestro servicio en un contenedor, un atacante podría saltar del contenedor al sistema host si no se ha realizado una configuración adecuada.

Para llevar a cabo esta tarea se hará uso de AppArmor y de Seccomp aunque tal y como se puede observar en el apartado Aislamiento de contenedores **se recomienda el uso de Seccomp** ya que no da tantos problemas como su homólogo.

Para utilizar **AppArmor** nos apoyaremos en la herramienta **bane** la cual nos generará un perfil, el cual cargaremos en nuestro sistema con el comando "**sudo apparmor_parser -r /etc/apparmor.d/containers/docker-items**" para posteriormente ejecutar el contendor haciendo uso del flag **"--security-opt apparmor=docker-items**".

**Figura 10 Ejecución del contenedor haciendo uso de AppArmor para mejorar el aislamiento**

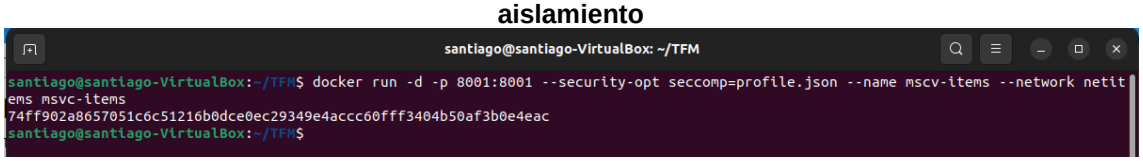



Fuente: Elaboración propia

En el caso de **Seccomp** también necesitaremos un perfil el cual podemos obtener de la siguiente ruta **https://github.com/moby/moby/blob/master/profiles/seccomp/default.json.**

Una vez descargado el perfil lo usaremos haciendo uso del flag **"--security-opt seccomp=profile.json**" a la hora de ejecutar el contenedor.

**Figura 11 Ejecución del contenedor haciendo uso de Seccomp para mejorar el aislamiento**




Fuente: Elaboración propia

Una vez implementada la solución, se confiere **mayor robustez al aislamiento de los contenedores** evitando que puedan acceder a recursos del sistema host que no son necesarios.

## 7.4. Limitación de recursos

Cuando se ejecuta un contenedor este **consume recursos del sistema host** y dependiendo del tipo de contenedor y del servicio que esté prestando estos recursos pueden aumentar o disminuir.

A simple vista puede parecer que no implica ninguna amenaza, pero se puede dar la situación que un contenedor tenga un bug o haya sido programado para

**consumir una gran cantidad de recursos** del sistema (contenedor creado por un atacante).

Esto puede provocar que el **sistema host se quede sin recursos** y ralentizar todos los contenedores que se encuentren en él alojados.

Para evitar esta situación se debe **limitar la cantidad de recursos** a los que puede acceder el contenedor. En este apartado se aplicarán una serie de restricciones a nivel de **procesador y RAM** sobre el contenedor "**msvc-items**" para evitar que consuma más recursos de los necesarios.

Para realizar dicha tarea se debe ejecutar los contenedores con los flags "**--cpus=1"** y **"--memory=512m**" tal y como se puede observar en la siguiente ilustración.

**Figura 12 Ejecución del contenedor limitando los recursos**


Fuente: Elaboración propia

Para ampliar la información se puede consultar el anexo del presente documento Limitación de recursos.

Gracias a esta configuración en los contenedores, nos aseguramos de que ninguno de ellos puede sobrepasar los recursos que le hemos asignado y por consiguiente **evitaremos que un contenedor sea capaz de consumir todos los recursos del host**.

## 7.5. Búsqueda de vulnerabilidades

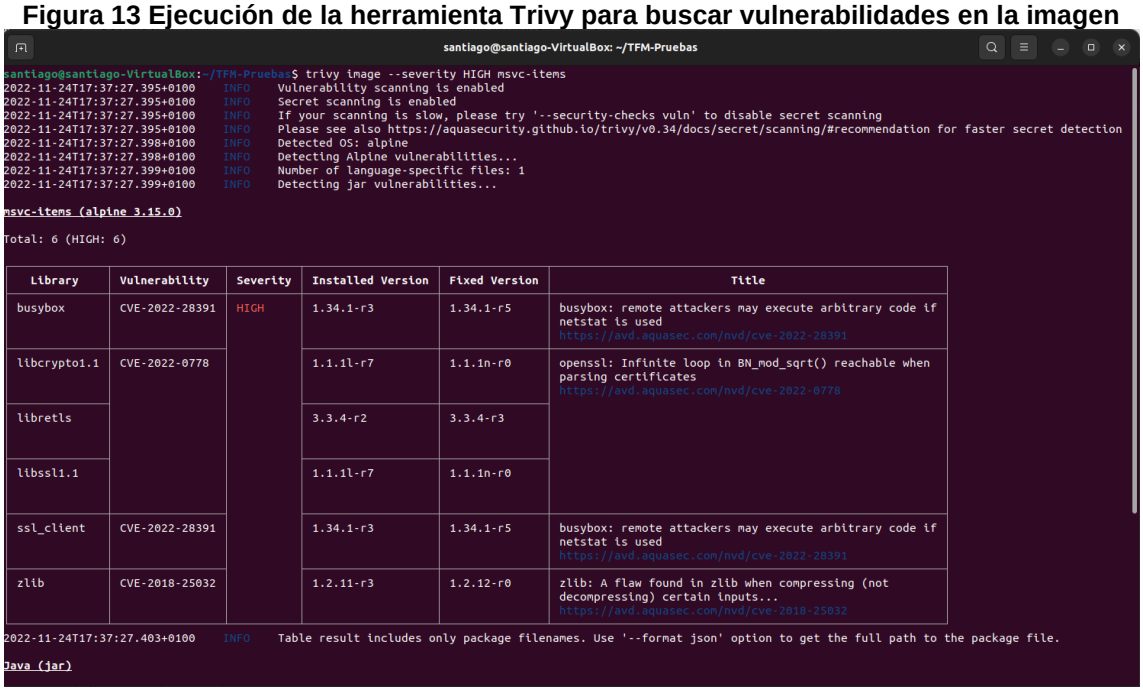
A pesar de que hayamos realizado una configuración segura de nuestro sistema basado en contenedores, existe la posibilidad de que las imágenes que se estén usando **contengan vulnerabilidades**. Esto quiere decir que cuando levantemos los contenedores, el servicio es susceptible de recibir un ataque aprovechando dicha vulnerabilidad.

Debido a esto **es muy importante que se tengan las imágenes actualizadas** con las últimas dependencias y/o librerías para de esta forma evitar las vulnerabilidades conocidas.

Para llevar a cabo esta tarea existen herramientas que son capaces de **escanear las imágenes en busca de vulnerabilidades** y es en este apartado donde vamos a aplicar una de estas herramientas.

La herramienta elegida es **Trivy** y para escanear las imágenes se debe ejecutar el siguiente comando "**trivy image --severity HIGH msvc-items**". Tras su ejecución la herramienta nos proporciona un informe con los resultados obtenidos.

**Figura 13 Ejecución de la herramienta Trivy para buscar vulnerabilidades en la imagen**




Fuente: Elaboración propia

Para más detalles se puede consultar el anexo del presente documento Búsqueda de vulnerabilidades.

Gracias al uso de este tipo de herramientas podemos buscar vulnerabilidades conocidas dentro de nuestras imágenes, para posteriormente solventar dicho problema. Con esto se consigue que las imágenes que se usan para desarrollar el sistema de información sean **más robustas y seguras**.

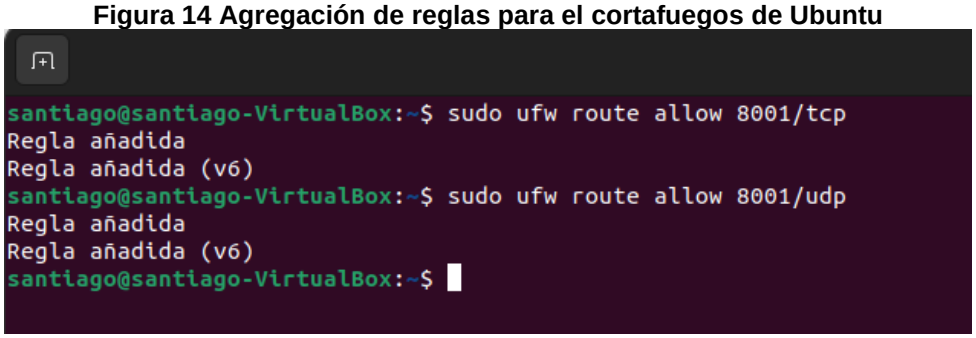
## 7.6. Implementación de cortafuegos

En un sistema de información existen una gran cantidad de conexiones las cuales pueden ser legítimas o no, eso quiere decir que tendremos una parte de estas cuyo objetivo es el de comprometer el sistema.

Es por ello, por lo que es buena práctica implementar un firewall que solo permita las conexiones que necesite el sistema de información para su correcto funcionamiento y evite las conexiones no deseadas.

Para implementar dicho firewall se hará uso de **UFW** el cual viene instalado por defecto en Ubuntu. Para su correcta configuración usaremos los comandos “**ufw route allow 8001/tcp**” y “**ufw route allow 8001/udp**” para de esta forma solo permitir conexiones a dichos puertos.

**Figura 14 Agregación de reglas para el cortafuegos de Ubuntu**




Fuente: Elaboración propia

Para más detalles sobre dicha configuración se puede visualizar el anexo Implementación de cortafuegos.

Gracias al uso de un cortafuegos se puede exponer solo los puertos necesarios para el correcto funcionamiento del sistema de información, con lo que se **reduciría el área de ataque** de un cibercriminal y por ende mejoraría la seguridad.

## 7.7. Proxy Inverso

A la hora de montar un sistema de información es **buena práctica** la implementación de un **proxy inverso** para acceder al mismo, gracias a esto conseguimos que los usuarios de fuera de nuestro sistema solo vean y tengan acceso al endpoint que nosotros configuremos en el proxy inverso aumentando de esta forma la seguridad (DockerHub, 2022; Ellingwood, 2015; Nginx, 2022; Nginx, 2022; y Ravoof, 2022).

El uso de un proxy inverso tiene las siguientes ventajas:

- **Balancear la carga:** Esto quiere decir que el proxy inverso es capaz de balancear las peticiones que recibe el aplicativo, por ejemplo, si tenemos dos instancias de nuestro servicio API el balanceador de carga dividirá de forma óptima las peticiones entre las dos instancias.
- **Mejora de seguridad:** El proxy inverso es capaz de ocultar la dirección IP y algunas características del servicio API ya que el usuario solo verá la IP del proxy inverso.
  Además, se pueden aplicar filtros en el proxy inverso para no dejar pasar cierto tipo de peticiones o usarlo como cortafuegos
- **Mejora de rendimiento:** El servidor proxy puede almacenar en su cache todo el contenido estático del aplicativo por lo que a la hora de ofrecer información al usuario este puede recibirla de forma más rápida.

Para realizar esta prueba de concepto se hará uso del servidor proxy **Nginx** e implementaremos un **proxy inverso** para acceder al servicio API del aplicativo.

Para realizar esta función se debe modificar el fichero “**default.conf**” y configurarlo para redirigir las peticiones que le lleguen a nuestro servicio.

Para aportar más seguridad aplicaremos **HTTP Basic Authentication** al servidor de **Nginx** para que de esta forma los usuarios necesiten introducir sus credenciales para poder acceder al servicio API.

Para implementar el uso de credenciales crearemos un fichero llamado "**.htpasswd**" el cual contendrá la información de dichas credenciales.

Posteriormente introduciremos este fichero en el servidor de **Nginx** y activaremos el uso de credenciales modificando de nuevo el archivo "**default.conf**".

La configuración agregada al fichero tiene la siguiente forma:

```
location / {
        auth_basic "Restricted Content";
        auth_basic_user_file /etc/nginx/.htpasswd;
        proxy_pass http://mscv-items:8001/;
}
```

Finalmente, el fichero para la creación de la imagen tiene la siguiente forma:

**Figura 15 Fichero para la creación de la imagen usada para el servicio del proxy inverso**

```
1 FROM nginx
2 COPY default.conf /etc/nginx/conf.d/
3 COPY .htpasswd /etc/nginx/
4 EXPOSE 85
5 CMD ["nginx", "-g", "daemon off;"]
```

Fuente: Elaboración propia

Toda esta configuración se puede visualizar en el apartado Proxy inverso del anexo.

## 7.8. Uso de pods con Podman

Tal y como se comentó en el apartado "3.3. Podman" **Podman es capaz de hacer uso de pods** los cuales mejorarían el aislamiento del sistema. Es por ello por lo que en este apartado **ejecutaremos el sistema basado en contenedores haciendo uso de un pod** que contendrá tanto el aplicativo API como el motor de base de datos MySQL (Bause. 2019; y Goyal, 2021).

Para poder crear nuestro pod con Podman ejecutaremos el siguiente comando "**podman run -d -p 8001:8001 --name msvc-items --network netitems --pod new:pod-sistema msvc-items"** el cual lo que hace es crear un pod que contiene en su interior el servicio de "msvc-items".

**Figura 16 Creación del pod y el servicio API**


Fuente: Elaboración propia

Finalmente se agrega al mismo pod el servicio correspondiente a la base de datos con el siguiente comando "**podman run -d --name mysql --network netitems --pod pod-sistema -e MYSQL_ROOT_PASSWORD=sasa -e MYSQL_DATABASE=msvc_items Docker.io/library/mysql**".

**Figura 17 Agregación del servicio de base de datos en el pod**


Fuente: Elaboración propia

Todos los detalles del procedimiento para crear el pod se puede consultar en el apartado Uso de pods con Podman del anexo.

Gracias al uso de pods agregamos una nueva capa da aislamiento al sistema de información por lo que aumentaría la seguridad de este.

# 8. Resultados

Tal y como se ha ido observando durante el desarrollo del trabajo Fin de Máster, el uso de un sistema basado en contenedores a pesar de ser más seguro a priori no está exento de problemas de seguridad.

Esto quiere decir que al desarrollar un sistema basado en contenedores se debe tener presente una gran cantidad de amenazas que pueden provocar serios riesgos para la seguridad del aplicativo.

Durante el desarrollo de la prueba de concepto se han aplicado medidas de seguridad para mitigar dichas amenazas y de esta forma aumentar la seguridad del sistema de información.

Tras realizar la prueba de concepto y analizar los resultados se puede observar que a pesar de que un sistema basado en contenedores es algo más seguro que uno tradicional se deben aplicar medidas de seguridad para mitigar o eliminar las amenazas a las que está expuesto.

**Tanto Docker como Podman son dos buenas herramientas** que con una buena configuración y aplicando las medidas de seguridad necesarias pueden llegar a ser muy robustas.

No obstante, **Podman ofrece mayor facilidad** a la hora de realizar ciertas **configuraciones de seguridad** ya que carece de demonio y es capaz de ejecutar pods.

A continuación, se muestra un resumen de las medidas tomadas durante la realización de la prueba de concepto, así como los resultados obtenidos.

## 8.1. Política de mínimos privilegios

Al aplicar esta medida se consigue que el usuario que ejecuta el sistema basado en contenedores tenga los mínimos privilegios posibles.

Esto se logra primeramente haciendo que el **demonio de Docker** se ejecute como **rootless** y posteriormente modificando la creación de las imágenes para que de esta forma al ejecutar el contenedor éste se ejecute con un **usuario que no tenga privilegios de root**.

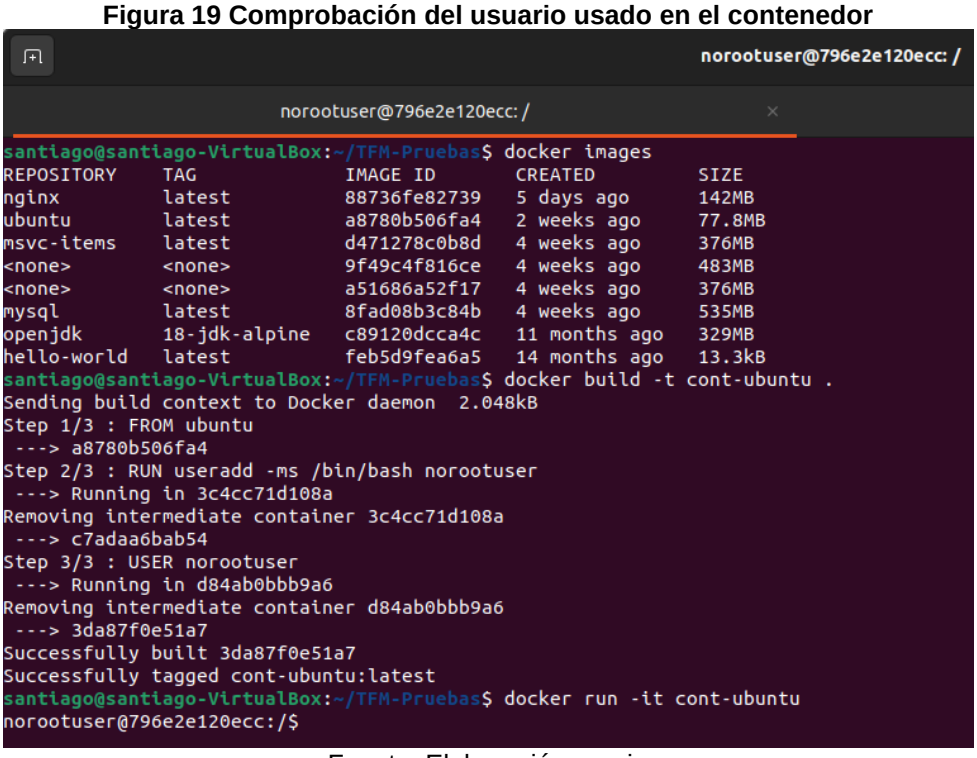
Como resultado final si visualizamos los permisos del demonio de Docker se puede observar que tiene activado el perfil de “**dockerd-rootless.sh**”.

**Figura 18 Comprobación de los permisos del demonio de Docker**


Fuente: Elaboración propia

Por otro lado, para comprobar que el contenedor se está ejecutando con **permisos de noroot** nos conectaremos a él con el comando "**docker run -it**", y tal y como se puede observar en la siguiente ilustración nos conectamos al contenedor con el usuario "**norootuser**".

**Figura 19 Comprobación del usuario usado en el contenedor**




Fuente: Elaboración propia

## 8.2. Aislamiento de contenedores

Como se ha visto, el hecho de hacer uso de un contenedor no quiere decir que éste se encuentre totalmente aislado del sistema host.

Es por eso por lo que durante la prueba de concepto se ha optado por usar **AppArmor** y **Seccomp,** para de esta forma mejorar considerablemente el aislamiento y los recursos a los que tiene acceso el servicio que se encuentra ejecutándose dentro del contenedor.

Como resultado obtenemos un servicio **mejor aislado**, el cual solo tiene acceso a ciertos **recursos,** además pueden ser **monitoreados** tal y como se observa en la siguiente ilustración.

**Figura 20 Monitoreo de los recursos a los que accede el servicio**


Fuente: Elaboración propia

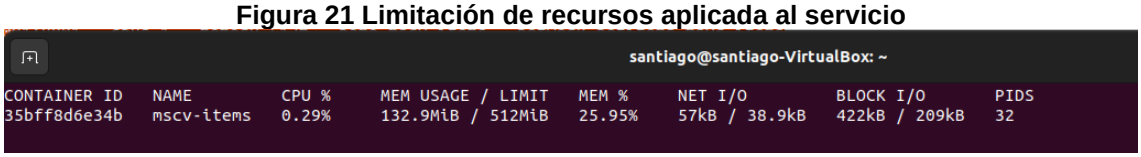
## 8.3. Limitación de recursos

Otra posible amenaza a la que se enfrenta el sistema es el agotamiento de los recursos del sistema host por parte de un contenedor mal configurado o malintencionado.

Para mitigar esta amenaza se optó por **limitar** tanto el **uso de CPU** como de **RAM** del servicio para evitar que pueda consumir más recursos de los que teóricamente necesita.

Tras realizar las medidas de mitigación correspondientes se puede observar como el límite máximo tanto de CPU como de RAM se ha visto reducido.

**Figura 21 Limitación de recursos aplicada al servicio**




Fuente: Elaboración propia

## 8.4. Búsqueda de vulnerabilidades

Ningún software está exento de contener vulnerabilidades por eso es muy buena práctica **analizar las imágenes** que se usarán para montar los servicios.

En este caso la herramienta usada fue **Trivy** y podemos comprobar el resultado obtenido al realizar un análisis de la imagen en la siguiente figura.

**Figura 22 Análisis de la imagen haciendo uso de la herramienta Trivy**

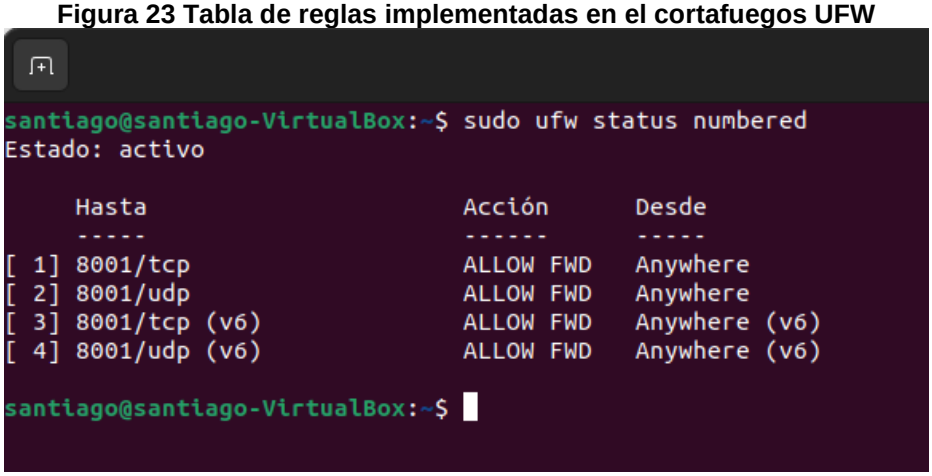

Fuente: Elaboración propia

## 8.5. Implementación de cortafuegos

A pesar de que se puede elegir que puerto del contenedor exponer esto no es suficiente para aislar adecuadamente la red, es por ello por lo que durante la prueba de concepto se hizo uso de un **cortafuegos (UFW)** mejorando de esta forma la seguridad a nivel de red del sistema.

Tras la configuración pertinente el resultado de las reglas usadas se puede observar en la siguiente ilustración.

**Figura 23 Tabla de reglas implementadas en el cortafuegos UFW**




Fuente: Elaboración propia

## 8.6. Proxy inverso

Como ya se comentó en la prueba de concepto, el uso de un proxy inverso (con autenticación en este caso) **mejora la seguridad del servicio** que se expone, ya que el usuario final que accede no interactúa directamente con el contenedor sino con el proxy inverso el cual puede filtrar ciertos tipos de paquetes malintencionados. Para realizar este proxy inverso se optó por hacer uso de **Nginx**, el cual fue montado también en un contenedor.

Como resultado obtenemos un **sistema simple de login** a través del proxy inverso para acceder al servicio.

**Figura 24 Sistema de login mediante el proxy inverso para poder acceder al servicio**


Fuente: Elaboración propia

## 8.7. Uso de Pods con Podman

Una característica que **tiene Podman** y de la cual **carece Docker** es la creación de pods en los cuales se pueden ejecutar uno o varios contenedores.

Gracias a esto se **aumenta el nivel de aislamiento** de dichos contenedores al agregar una nueva capa de aislamiento proporcionada por el pod.

El resultado obtenido es un pod el cual contiene todo el sistema de información, es decir, tanto el servicio API como la base de datos.

**Figura 25 Pod que contiene todo el sistema de información**

```
santiago@santiago-VirtualBox: $ podman run -d --name mysql --network netitems --pod pod-sistema -e MYSQL_ROOT_PASSWORD=sasa -e MYSQL_DATABASE=msvc_items docker.io/library/mysql
fc2ad36721cb3d820d5acffbe261240bfe92c219637c24640bd7e3063df7312f
santiago@santiago-VirtualBox: $ podman start mscv-items
mscv-items
santiago@santiago-VirtualBox: $ podman ps -a --pod
CONTAINER ID  IMAGE                           COMMAND               CREATED         STATUS                     PORTS                     NAMES               POD ID        PODNAME
36444f664120  localhost/proxy:latest          nginx -g daemon o...  2 days ago      Exited (0) 2 days ago      0.0.0.0:9090->80/tcp      proxy
b2f88bfdbb4e  k8s.gcr.io/pause:3.5                                  5 minutes ago   Up 5 minutes ago           0.0.0.0:8001->8001/tcp    ac7316acf421-infra  ac7316acf421  pod-sistema
a318c67f8a31  localhost/msvc-items:latest     java -jar ./msvc-...  5 minutes ago   Up 14 seconds ago          0.0.0.0:8001->8001/tcp    mscv-items          ac7316acf421  pod-sistema
fc2ad36721cb  docker.io/library/mysql:latest  mysqld                32 seconds ago  Up 33 seconds ago                                    mysql               ac7316acf421  pod-sistema
santiago@santiago-VirtualBox: $
```

Fuente: Elaboración propia

# 9. Conclusiones y trabajos futuros

## 9.1. Conclusión

Los sistemas basados en contenedores se han ido volviendo muy populares a medida que los sistemas más tradicionales han menguado, es por eso por lo que actualmente **una gran cantidad de los sistemas** que se encuentran en explotación **están basados en contenedores**.

Algunos de los motivos por los cuales muchas empresas hacen uso de contenedores para almacenar sus servicios es por la **agilidad**, **facilidad de uso**, **portabilidad**, pero también por su **seguridad**. No obstante, tal y como se ha podido comprobar durante el desarrollo del presente trabajo Fin de Máster el hecho de hacer uso de este tipo de tecnología no tiene porqué aumentar la seguridad del sistema.

Tal y como se ha visualizado en los apartados anteriores **un sistema basado en contenedores tiene un número de amenazas considerable**, las cuales **debemos eliminar o mitigar** en la medida de lo posible. De no ser así se correría el riesgo de que un cibercriminal aprovechase la brecha de seguridad.

Se ha comprobado que independientemente de **las herramientas que se usen para la gestión de los contenedores** se debe tener muy presente la seguridad del sistema, debiendo estar **correctamente configurada** para evitar problemas de seguridad.

Durante la realización del proyecto se han ido aplicando medidas de mitigación para solventar los distintos problemas de seguridad, no obstante, esto no quiere decir que el sistema se encuentre 100% seguro.

Esto se debe a que pueden aparecer nuevas vulnerabilidades o nuevos tipos de ataques que pueden comprometer la seguridad de todos el aplicativo. Debido a esto la seguridad debe ser integrada en todo el ciclo de vida del proyecto (**DevSecOps**).

Como ya se avanzó en el punto 1.3, este trabajo colabora con la sostenibilidad ya que el uso de contenedores permite hacer un **uso más eficiente de los recursos disponibles**. Además, se asegura que la productividad económica de las empresas que hagan uso de los contenedores pueda crecer económicamente aumentando su productividad. También cabe resaltar que se produce un **ahorro energético** que trae ventajas a toda la sociedad.

## 9.2. Planificación del proyecto

La planificación realizada durante todo el desarrollo del proyecto ha sido la adecuada, **pudiendo alcanzar todos los objetivos marcados** en el planteamiento inicial.

No obstante, se debe recalcar el **aumento considerable de la carga de trabajo a la hora de realizar el apartado de Prueba de conceto**, debido principalmente a los problemas encontrados a la hora de configurar adecuadamente el sistema y a la incorporación de las medidas de seguridad.

Por otro lado, es cierto que hay aspectos que se han pasado por encima y no se ha profundizado lo suficiente debido a que actualmente **se escapa del alcance del proyecto** como es el uso de **orquestadores** o el uso de **servicios cloud** para implementar el sistema de información.

## 9.3. Problemas encontrados durante el desarrollo del proyecto

A continuación, se detallan algunos de los problemas encontrados durante la implementación del presente proyecto.

### *9.3.1. Configuración del microservicio API*

Durante la configuración inicial del servicio **se intentó realizar una interfaz de usuario** haciendo uso de algún framework de front, pero debido a la complejidad que suponía realizarlo **se optó finalmente por la implementación de una API** careciendo de esta forma de una interfaz de usuario.

### *9.3.2. Configuración del sistema operativo host*

Algunas de las **medidas de seguridad** implementadas en el proyecto son **correspondientes al sistema operativo Linux** lo cual unido a la **poca experiencia trabajando con sistemas Linux** provocó problemas a la hora de implementar ciertas medidas de seguridad como AppArmor, Seccomp o UFW.

### *9.3.3. Implementación del proxy inverso*

Durante la implementación del proxy inverso aparecieron diversos **problemas a la hora de configurar el servidor de Nginx** ya que no era capaz de redirigir las peticiones adecuadamente al servicio de API.
Finalmente se puedo realizar dicha implementación agregando además un sistema de login para acceder al microservicio.

## 9.4. Evaluación de los objetivos alcanzados

En el presente apartado se realizará una valoración de los objetivos que se marcaron al inicio del proyecto.

### *9.4.1. Analizar las amenazas en sistemas basados en contenedores*

El objetivo puede darse como cumplido ya que durante el desarrollo del proyecto se ha realizado una **investigación de las principales amenazas** que afectan a un sistema basado en microservicios y contenedores.

Además, también se ha analizado las **amenazas referentes** a **sistemas de información tradicionales** y al uso de **herramientas de orquestación** como es **Kubernetes**.

### *9.4.2. Diseñar un modelo seguro de implementación de un sistema basado en contenedores.*

Para la realización de este objetivo se ha optado por la **implementación de un sistema basado en contenedores y orientado a los microservicios**, consiguiendo separar de esta manera el aplicativo del sistema host de una manera más eficaz.

Por otro lado, también **se han separado los dos servicios** que necesita el sistema de información como es la base de datos y el servicio correspondiente a la API **ejecutando ambos en contenedores separados**.

### *9.4.3. Validar el diseño seguro desarrollado mediante una prueba de concepto.*

Para alcanzar este objetivo de forma satisfactoria **se ha realizado una prueba de concepto** en la cual se ha implementado un pequeño sistema de información sobre el que se han ido aplicando las distintas medidas de seguridad.

El sistema de información desarrollado es **totalmente funcional** y puede realizar las **operaciones de CRUD** contra al base de datos haciendo uso de la API que ofrece.

## 9.5. Trabajos futuros

Existen una gran cantidad de posibles líneas de trabajo en las que poder ampliar el contenido del presente proyecto.

A continuación, se detallan algunas de las líneas consideradas más relevantes en las que poder trabajar.

- **Orquestadores:** En el presente trabajo los orquestadores han quedado en un segundo plano para de esta forma centrar todos los esfuerzos en herramientas como Docker y Podman. De esta forma una línea de trabajo futuro puede ser la **implementación de un sistema de**

**información** haciendo uso de una plataforma de orquestación como puede ser **Kubernetes** o **Docker Swarm**.

- **Servicios Cloud:** Actualmente muchas empresas hacen uso de servicios cloud ya que logran reducir costes y delegan en terceros ciertas tareas de mantenimiento y securización. Estos servicios han sido comentados en el trabajo Fin de Máster, aunque no se ha realizado ninguna implementación con ninguno de ellos.
  Otra posible línea de trabajo sería realizar una **implementación de un sistema de información** haciendo uso de algunos de los **servicios cloud** comentados y llevar todo el sistema a la nube.

# 10. Glosario

**Amenaza:** Una amenaza es un hecho potencialmente peligroso para el sistema que puede ocurrir, y de ser así provocaría daños sobre el mismo (Ministerio de Hacienda y Administraciones Públicas, 2012).

**API:** El api (Interfaz de Programación de Aplicaciones) es un mecanismo que permite la comunicación entre dos componentes software mediante un conjunto de definiciones y protocolos (Amazon, 2022).

**Cloud:** El Cloud o Cloud Computing es un modelo que permite el acceso bajo demanda y a través de Internet a un conjunto de recursos informáticos. Estos recursos informáticos son gestionados por un proveedor y son ofrecidos al usuario (Suppi, 2022).

**Contenedor:** Son instancias en ejecución de una imagen, gracias a los contenedores podemos ejecutar imágenes y exponerlas en algún puerto (Garzas, 2015).

**Daemon:** Es un proceso que se ejecuta continuamente en segundo plano para que otro proceso funcione correctamente (IBM, 2021).

**Daemonless:** Se dice que un programa es daemonless cuando carece de un demonio para su correcto funcionamiento (IBM, 2021).

**Docker:** Docker es una plataforma para automatizar la implementación de aplicaciones como contenedores portátiles (Oracle, 2022).

**Host:** Se denomina host al sistema primario (sistema operativo y hardware) sobre el cual ejecutaremos los contenedores (Theastrologypage, 2022).

**Imagen:** Es una representación estática del servicio y de la configuración/dependencias del aplicativo a ejecutar (Garzas, 2015).

**Json:** Json es una estructura de datos en forma de texto que permite el intercambio de información entre distintos servicios (de Souze, 2019).

**Kernel:** El Kernel o núcleo es una parte fundamental del sistema operativo que se encarga de conceder acceso al hardware de forma segura por parte del software que lo solicita (Soto, 2020).

**Kubernetes:** Es una plataforma de código abierto la cual nos permite administrar, automatizar y escalar aplicaciones basadas en contenedores, es decir, es un orquestador de contenedores (Kubernetes, 2022).

**Petición GET:** Método usado en la API para la obtención de datos de lectura, la respuesta ofrecida por este método siempre serán datos (Redondo, 2018).

**Petición POST:** Método usado en la API para la creación de nuevos objetos (Redondo, 2018).

**Podman:** Podman es una herramienta que nos ayuda en la tarea de administrar y trabajar con contenedores (Podman, 2019).

**Pods:** Los Pods son grupos de contenedores que se ejecutan juntos y son capaces de compartir los mismos recursos de una forma muy similar a como lo hace Kubernetes (Bause, 2019).

**Postman:** Es una plataforma que facilita la tarea de realizar peticiones REST a una API (Postman, 2022).

**Proxy Inverso:** Un proxy inverso es un servidor el cual se coloca delante de los servidores que ofrecen un servicio y su función es reenviar las solicitudes que realizan los clientes al servidor API (Cloudflare, 2022).

**Root:** El usuario root es aquel que tiene permisos de administración sobre el sistema, es decir, puede acceder a cualquier archivo y ejecutar cualquier comando. Por norma general un usuario carece de dichos privilegios (Pérez, 2020).

**Sistema de información:** Es un conjunto de recursos informáticos que juntos son capaces de administrar datos (Editorial Etecé, 2021).

**Vulnerabilidad:** Un defecto en un programa o en un sistema de información que puede provocar un problema de seguridad es denominado vulnerabilidad (Ministerio de Hacienda y Administraciones Públicas, 2012).

# 11. Bibliografía


Aladrén, J. (2020). *Análisis estático de vulnerabilidades en Kubernetes para entornos de integración continua*. Universitat Oberta de Catalunya [Trabajo Fin de Grado]. Recuperado día 08 de octubre de 2022, de: https://openaccess.uoc.edu/handle/10609/107606

Amazon. (2022). *¿Qué es una API?* Recuperado el día 11 de diciembre de 2022 de: https://aws.amazon.com/es/what-is/api/#:~:text=API%20significa%20%E2%80%9Cinterfaz%20de%20programaci%C3%B3n,de%20servicio%20entre%20dos%20aplicaciones.

Ameijeiras, R. (2022). *¿Qué es Podman? ¿El sucesor de Docker?* Recuperado el día 16 de noviembre de 2022 de: https://pandorafms.com/blog/es/que-es-podman/

Avi. 2022. *8 Kubernetes Scanner to find Security Vulnerability and Misconfiguration*. Recuperado el día 30 de diciembre de 2022, de: https://geekflare.com/kubernetes-security-scanner/

AWS (Amazon Web Services). (2022). *Amazon detective*. Recuperado el día 13 de noviembre de 2022 de: https://aws.amazon.com/es/detective/

AWS (Amazon Web Services). (2022). *Amazon Elastic Container Service (Amazon ECS)*. Recuperado el día 12 de noviembre de 2022 de: https://aws.amazon.com/es/ecs/

AWS (Amazon Web Services). (2022). *Amazon Elastic Kubernetes Service (EKS)*. Recuperado el día 13 de noviembre de 2022 de: https://aws.amazon.com/es/eks/

AWS (Amazon Web Services). (2022). *Amazon GuardDuty*. Recuperado el día 13 de noviembre de 2022 de: https://aws.amazon.com/es/guardduty/

AWS (Amazon Web Services). (2022). *AWS WAF*. Recuperado el día 13 de noviembre de 2022 de: https://aws.amazon.com/es/waf/

Azure. (2022). *Application Gateway*. Recuperado el día 13 de noviembre de 2022 de: https://azure.microsoft.com/es-es/products/application-gateway/#features

Azure. (2022). *Azure Container Apps*. Recuperado el día 13 de noviembre de 2022 de: https://azure.microsoft.com/es-es/products/container-apps/#overview

Azure. (2022). *Azure Kubernetes Service (AKS)*. Recuperado el día 13 de noviembre de 2022 de: https://azure.microsoft.com/es-es/products/kubernetes-service/#overview

Azure. (2022). *Azure Monitor*. Recuperado el día 13 de noviembre de 2022 de: https://azure.microsoft.com/en-us/products/monitor/

Baeldung. (2022). *Add a User in Alpine Docker Image*. Recuperado el día 20 de noviembre de 2022 de: https://www.baeldung.com/linux/docker-alpine-add-user

Bailón, P. (2021). *¿Afectará el ataque informático al SEPE en las prestaciones? Las diferentes versiones de ministerio y sindicatos*. Antena 3 Noticias. Recuperado el día 15 de noviembre de 2022, de: https://www.antena3.com/noticias/economia/afectara-el-ataque-informatico-al-sepe-en-las-prestaciones-las-diferentes-versiones-de-ministerio-y-sindicatos_20210323605a08d1404114000187067c.html

Baretto, A. (2017). *Best Practices for Securing Containers*. Recuperado el día 16 de noviembre de 2022 de: https://medium.com/@axbaretto/best-practices-for-securing-containers-8bf8ae0d9952

Bause, B. (2019). *Podman: Managing pods and containers in a local container runtime*. Recuperado el día 27 de noviembre de 2022 de: https://developers.redhat.com/blog/2019/01/15/podman-managing-containers-pods#

Caparrós, J. [Joan]; Cubero, L. [Lorenzo]; Fernández, J.C. [Juan Carlos]; y, Guijarro, J. [Jordi]. (2022). *Introducción a la seguridad en arquitecturas de microservicios basadas en contenedores* [recurso de aprendizaje]. Recuperado día 23 de octubre de 2022 de: https://materials.campus.uoc.edu/daisy/Materials/PID_00287480/pdf/PID_00287480.pdf

CCII (Consejo General de Colegios Profesionales Ingeniería Informática). (2022). *Código ético y deontológico de la Ingeniería Informática*. Recuperado día 08 de octubre de 2022 de: Código Ético y Deontológico de la Ingeniería Informática (ccii.es)

Cloudflare. (2022). *¿Qué es un proxy inverso?* | *Servidores proxy explicados*. Recuperado el día 11 de diciembre de 2022 de: https://www.cloudflare.com/es-es/learning/cdn/glossary/reverse-proxy/

Dawson. (2022). *Podman vs Docker: All You Need To Know!* Recuperado el día 28 de octubre de 2022, de: https://www.lambdatest.com/blog/podman-vs-docker/#:~:text=How%20is%20Podman%20different%20from,pods%20and%20images%20are%20smaller.

de Souza, I. (2019). *Archivo JSON: ¿qué es y para qué sirve en las páginas web?* Recuperado el día 11 de diciembre de 2022 de: https://rockcontent.com/es/blog/archivo-json/

Deberdt, C. (2022). *What is Apparmor and how to add a security layer with it in Docker?* Recuperado el día 22 de noviembre de 2022 de: https://www.padok.fr/en/blog/security-docker-apparmor

Debian. (2022). *El manual del Administrador de Debian*. Recuperado el día 22 de noviembre de 2022 de: https://debian-handbook.info/browse/es-ES/stable/sect.apparmor.html

Docker. (2022). *AppArmor security profiles for Docker*. Recuperado el día 22 de noviembre de 2022 de: https://docs.docker.com/engine/security/apparmor/

Docker. (2022). *Docker docs*. Recuperado día 28 de octubre de 2022, de: https://docs.docker.com/

Docker. (2022). *docker stats*. Recuperado el día 22 de noviembre de 2022 de: https://docs.docker.com/engine/reference/commandline/stats/

Docker. (2022). *Run the Docker daemon as a non-root user (Rootless mode)*. Recuperado el día 20 de noviembre de 2022 de: https://docs.docker.com/engine/security/rootless/

Docker. (2022). *Runtime options with Memory, CPUs, and GPUs*. Recuperado el día 22 de noviembre de 2022 de: https://docs.docker.com/config/containers/resource_constraints/

Docker. (2022). *Seccomp security profiles for Docker*. Recuperado el día 30 de noviembre de 2022 de: https://docs.docker.com/engine/security/seccomp/

Docker. (2022). *Use containers to Build, Share and Run your applications*. Recuperado día 23 de octubre de 2022 de: https://www.docker.com/resources/what-container/

DockerHub. (2022). Nginx, Official build of Nginx. Recuperado el día 26 de noviembre de 2022 de: https://hub.docker.com/_/nginx

Donohue, T. (2022). *Containers: one single process, or multiple processes?* Recuperado el día 16 de noviembre de 2022 de: https://www.tutorialworks.com/containers-single-or-multiple-processes/

Editorial Etecé. (2021). *Sistema de información*. Recuperado el día 11 de diciembre de 2022 de: https://concepto.de/sistema-de-informacion/

Ellingwood, J. (2015). *How To Set Up Password Authentication with Nginx on Ubuntu 14.04*. Recuperado el día 26 de noviembre de 2022 de: https://www.digitalocean.com/community/tutorials/how-to-set-up-password-authentication-with-nginx-on-ubuntu-14-04

Figueiredo, R y Gamela, A. (2021). *Podman vs Docker: What are the differences?* Recuperado el día 28 de octubre de 2022, de: https://www.imaginarycloud.com/blog/podman-vs-docker/

Gamboa, J. (2018). *Seguridad en Docker*. Universitat Oberta de Catalunya [Trabajo Fin de Grado]. Recuperado día 09 de octubre de 2022 de: https://openaccess.uoc.edu/handle/10609/89325

Garzas, J. (2015). *Entendiendo Docker. Conceptos básicos: Imágenes, Contenedores, Links….* Recuperado el día 28 de octubre de 2022, de: https://www.javiergarzas.com/2015/07/entendiendo-docker.html

Github. (2022). *Podman, Troubleshooting*. Recuperado el día 22 de noviembre de 2022 de: https://github.com/containers/podman/blob/main/troubleshooting.md

González, B. (2018). *Seguridad en Docker*. Universitat Oberta de Catalunya [Trabajo Fin de Grado]. Recuperado día 09 de octubre de 2022 de: https://openaccess.uoc.edu/handle/10609/81405

Google Cloud. (2022). *Cloud Key Management*. Recuperado el día 13 de noviembre de 2022 de: https://cloud.google.com/security-key-management

Google Cloud. (2022). *Cloud Monitoring*. Recuperado el día 13 de noviembre de 2022 de: https://cloud.google.com/monitoring?hl=es

Google Cloud. (2022). *Cloud Run*. Recuperado el día 13 de noviembre de 2022 de: https://cloud.google.com/run?hl=es

Google Cloud. (2022). *Google Kubernetes Engine*. Recuperado el día 13 de noviembre de 2022 de: https://cloud.google.com/kubernetes-engine

Google Cloud. (2022). *Overview of Web Security Scanner*. Recuperado el día 13 de noviembre de 2022 de: https://cloud.google.com/security-command-center/docs/concepts-web-security-scanner-overview?hl=en

Goyal, M. (2021). *Spinning up and Managing Pods with multiple containers with Podman*. Recuperado el día 29 de noviembre de 2022 de: https://mohitgoyal.co/2021/04/23/spinning-up-and-managing-pods-with-multiple-containers-with-podman/

Heddings, A. (2022). *How To Use Docker with a UFW Firewall*. Recuperado el día 25 de noviembre de 2022 de: https://www.howtogeek.com/devops/how-to-use-docker-with-a-ufw-firewall/

Heidi, E. (2019). *Cómo configurar un firewall con UFW en Ubuntu 18.04*. Recuperado el día 25 de noviembre de 2022 de: https://www.digitalocean.com/community/tutorials/como-configurar-un-firewall-con-ufw-en-ubuntu-18-04-es

IBM. (2021). *Daemons TCP/IP*. Recuperado el día 11 de diciembre de 2022 de: https://www.ibm.com/docs/es/aix/7.2?topic=protocol-tcpip-daemons

Isaac. 2021. *Docker vs Kubernetes: ventajas y desventajas*. Recuperado el día 29 de diciembre de 2022, de: https://blog.desdelinux.net/docker-vs-kubernetes/

Kubernetes. (2022). *Orquestación de contenedores para producción*. Recuperado el día 11 de diciembre de 2022 de: https://kubernetes.io/es/

Kubernetes. 2022. *Security Checklist*. Recuperado el día 30 de diciembre de 2022, de: https://kubernetes.io/docs/concepts/security/security-checklist/

Ministerio de Hacienda y Administraciones Públicas. (2012). *MAGERIT versión 3: Metodología de Análisis y Gestión de Riesgos de los Sistemas de Información*. Portal de Administración Electrónica. Recuperado día 20 de octubre de 2022, de: https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Magerit.html

Naciones Unidas (2022). *Objetivos de Desarrollo Sostenible*. Recuperado día 09 de octubre de 2022 de: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/

NetApp. (2022). ¿Qué son los contenedores?. *NetApp*. Recuperado día 23 de octubre de 2022, de: https://www.netapp.com/es/devops-solutions/what-are-containers/#:~:text=Docker%20es%20un%20popular%20entorno,las%20pruebas%20y%20la%20producci%C3%B3n.

Nginx. (2022). *Module ngx_http_core_module*. Recuperado el día 26 de noviembre de 2022 de: https://nginx.org/en/docs/http/ngx_http_core_module.html#location

Ngnix. (2022). *Restricting Access with HTTP Basic Authentication*. Recuperado el día 26 de noviembre de 2022 de: https://docs.nginx.com/nginx/admin-guide/security-controls/configuring-http-basic-authentication/

Oosterhof, N. (2022). *How to create new users in a Docker container?* Recuperado el día 20 de noviembre de 2022 de: https://net2.com/how-to-create-new-users-in-docker-container/

Oracle. (2022). *¿Qué es Docker?* Recuperado el día 11 de diciembre de 2022 de: https://www.oracle.com/es/cloud/cloud-native/container-registry/what-is-docker/

OWASP (2022). *OWASP Docker Top 10. OWASP*. Recuperado día 27 de octubre de 2022, de: https://owasp.org/www-project-docker-top-10/#

OWASP. (2022). OWASP Top Ten. *OWASP*. Recuperado día 20 de octubre de 2022, de: https://owasp.org/www-project-top-ten/

Pérez, M. (2020). *5 cosas que deber saber sobre el usuario root en Ubuntu*. Recuperado el día 11 de diciembre de 2022 de: https://geekytheory.com/usuario-root-ubuntu-linux/#:~:text=En%20Linux%20el%20usuario%20root,los%20que%20nunca%20deber%C3%ADas%20ejecutar.

Podman. (2019). *What is Podman?* Recuperado el día 28 de octubre de 2022, de: https://docs.podman.io/en/latest/index.html

Podman. (2022). *Podman run*. Recuperado el día 22 de noviembre de 2022 de: https://docs.podman.io/en/latest/markdown/podman-run.1.html
Portada - Desarrollo Sostenible (un.org)

Postman. (2022). *Build APIs together*. Recuperado el día 11 de diciembre de 2022 de: https://www.postman.com/

Ravoof, S. (2022). *¿Cómo Configurar un Proxy Inverso? (Paso a Paso para Nginx y Apache)*. Recuperado el día 26 de noviembre de 2022 de: https://kinsta.com/es/blog/proxy-inverso/#:~:text=Un%20proxy%20inverso%20se%20ubica,el%20usuario%20o%20el%20cliente

RedHat. 2022. *Cómo abordar la seguridad de Kubernetes*. Recuperado el día 30 de diciembre de 2022, de: https://www.redhat.com/es/topics/containers/kubernetes-security

Redondo, M. (2018). *Anatomía de una petición a una API REST.* Recuperado el día 11 de diciembre de 2022 de: https://errequeerre.es/anatomia-de-una-peticion-a-una-api-rest/#:~:text=GET%3A%20nos%20sirve%20para%20solicitar,se%20ha%20encontrado%20el%20recurso).

Salcedo, R.M. (2022). *Un ataque informático a un proveedor de Orange deja al descubierto datos "sensibles" de clientes*. Antena 3 Noticias. Recuperado el día 15 de noviembre de 2022, de: https://www.antena3.com/noticias/tecnologia/ataque-informatico-proveedor-orange-deja-descubierto-datos-sensibles-clientes_2022110763693a75ac67a20001acab87.html

Simón, M. (2021). *Análisis de seguridad de arquitecturas basadas en Kubernetes*. Universitat Oberta de Catalunya [Trabajo Fin de Grado]. Recuperado día 09 de octubre de 2022 de: https://openaccess.uoc.edu/handle/10609/132656

Singh, G. (2022). *Basics of Seccomp for Docker*. Recuperado el día 30 de noviembre de 2022 de: https://tbhaxor.com/basics-of-seccomp-for-dockers/

Soto, J.A. (2020). *¿Qué es el Kernel y para qué sirve?* Recuperado el día 11 de diciembre de 2022 de: https://www.geeknetic.es/Kernel/que-es-y-para-que-sirve

Stackscale. 2021. *Contenedores, sistemas de contenerización y orquestadores*. Recuperado el día 29 de diciembre de 2022, de: https://www.stackscale.com/es/blog/contenerizacion-orquestacion-contenedor/#Que_es_la_orquestacion_de_contenedores

Suppi, R. [Remo]. (2022). *Fundamentos y plataformas de cloud computing* [recurso de aprendizaje]. Recuperado día 12 de noviembre de 2022 de: https://materials.campus.uoc.edu/daisy/Materials/PID_00287480/pdf/PID_00287480.pdf

Tara Seals (2022). *Deep Dive: Protecting Against Container Threats in the Cloud. ThreatPost*. Recuperado el día 27 de octubre de 2022, de: https://threatpost.com/container_threats_cloud_defend/179452/

The kernel development community. (2022). *The Linux Kernel. Seccomp BPF (SECure COMPuting with filters)*. Recuperado el día 30 de noviembre de 2022 de: https://www.kernel.org/doc/html/v4.18/userspace-api/seccomp_filter.html

Theastrologypage. (2022). *¿Qué es el sistema operativo host? - definición de techopedia*. Recuperado el día 11 de diciembre de 2022 de: https://es.theastrologypage.com/host-operating-system#menu-1

Trivy. (2022). *Aqua Trivy*. Recuperado el día 24 de noviembre de 2022 de: https://aquasecurity.github.io/trivy/v0.34/

Wilson, B (2021). *Podman Tutorial For Beginners: Step by Step Guides*. Recuperado el día 28 de octubre de 2022, de: https://devopscube.com/podman-tutorial-beginners/

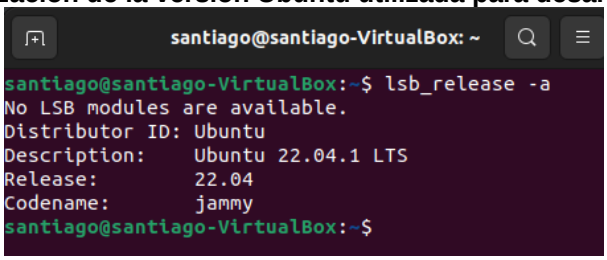
# Anexos

## Anexo I. Configuración del sistema de información

### Configuración del entorno de trabajo

Para la implementación del sistema de información se ha optado por el uso de un sistema operativo Linux, más concretamente **Ubuntu en su versión 22.04.1 LTS.** Este sistema operativo se ejecuta sobre **Oracle VM Virtual Box** cuyo sistema host es Windows 10.

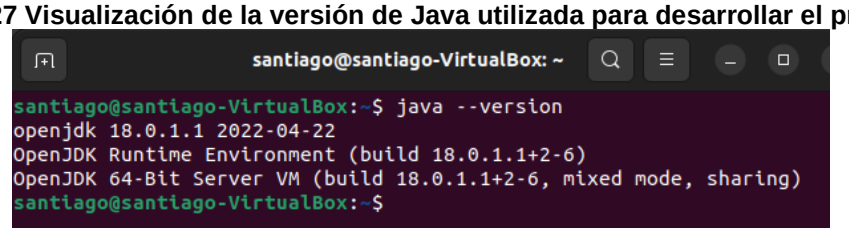
**Figura 26 Visualización de la versión Ubuntu utilizada para desarrollar el proyecto**




Fuente: Elaboración propia

Para el desarrollo del aplicativo (API) necesitaremos hacer uso de **Java** en su **versión 18** o superior, aunque también es posible realizarlo con versiones más antiguas. Para evitar problemas de licencias se ha optado por hacer uso de **openJDK**.

**Figura 27 Visualización de la versión de Java utilizada para desarrollar el proyecto**




Fuente: Elaboración propia

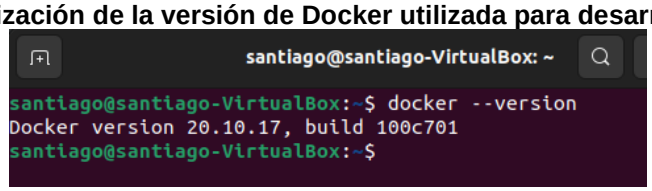
Como IDE para realizar la implementación se ha optado por usar **IntelliJ IDEA** en su versión **Community**, pero se puede usar otro como **Eclipse**.

Para realizar las peticiones a nuestra API se puede usar cualquier tipo de herramienta y/o plataforma, pero durante el desarrollo del presente documento se usará **Postman** debido a que es gratuito y su manejo es sencillo.

Por último, necesitaremos una herramienta para ayudarnos a crear los contenedores, en este caso hemos elegido **Docker y Podman** debido a su popularidad y fácil manejo.
Para realizar la instalación seguiremos la guía que se nos proporciona en la página web de Docker (https://docs.docker.com/engine/install/ubuntu/).

Una vez instalado comprobaremos su versión la cual para este trabajo Fin de Máster será la **20.10.17**.

**Figura 28 Visualización de la versión de Docker utilizada para desarrollar el proyecto**

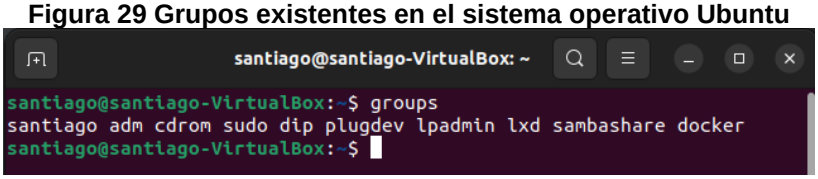



Fuente: Elaboración propia

A continuación, debemos realizar un paso más para completar correctamente la instalación de Docker ya que por defecto para la ejecución de cualquier contenedor nos solicita escalar privilegios de root, es decir, nos obliga a usar el comando “sudo” para la ejecución de contenedores (Docker, 2022).

Para evitar esto debemos agregar nuestro usuario al grupo de docker que nos crea automáticamente al finalizar la instalación.

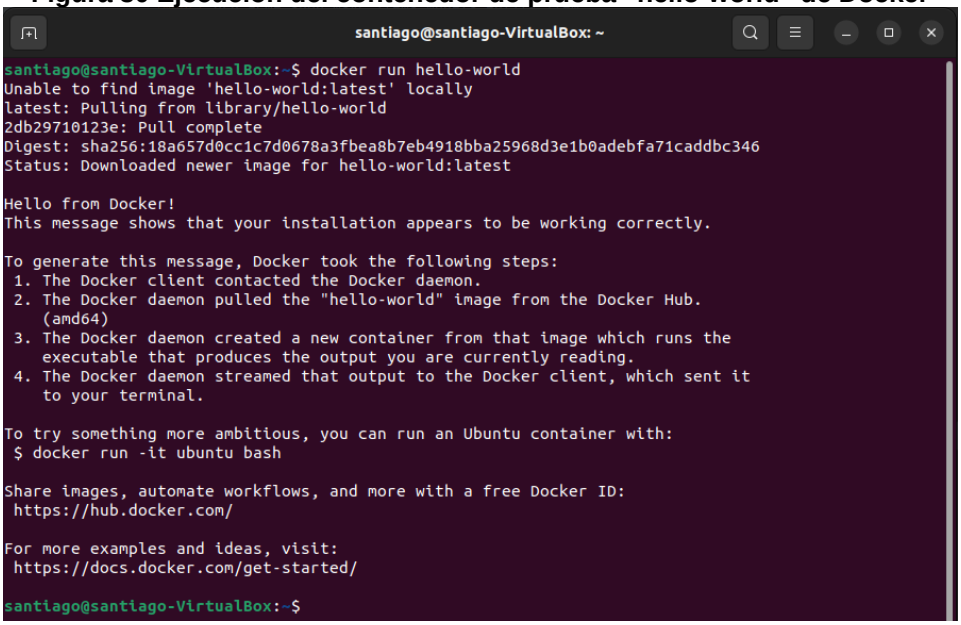
**Figura 29 Grupos existentes en el sistema operativo Ubuntu**




Fuente: Elaboración propia

Para realizar esta acción ejecutaremos el comando “**sudo usermod -aG Docker $USER”.** Una vez ejecutado el comando podemos lanzar contenedores sin la necesidad de usar el comando “sudo” tal y como se puede ver en la siguiente ilustración.

**Figura 30 Ejecución del contenedor de prueba “hello-world” de Docker**



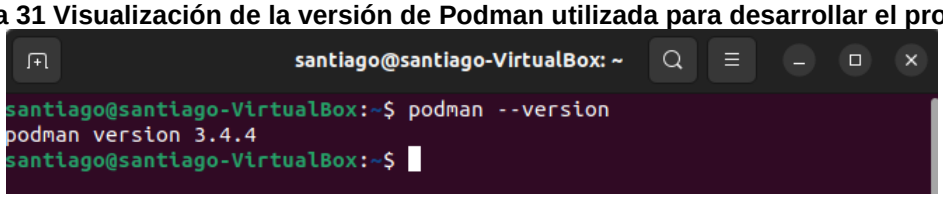

Fuente: Elaboración propia

Además de Docker se usará **Podman** por lo que a continuación se detalla su instalación (Podman, 2019).

Para realizar la instalación en Ubuntu seguiremos los pasos que encontraremos en la documentación oficial accesible desde el siguiente enlace https://podman.io/getting-started/installation

Una vez instalado ejecutaremos el siguiente comando para visualizar la versión instalada.

**Figura 31 Visualización de la versión de Podman utilizada para desarrollar el proyecto**




Fuente: Elaboración propia

**Podman** no necesita agregar nuestro usuario a ningún grupo ya que no necesita permisos de root para poder ejecutarse (**rootless**).

En la siguiente tabla se muestra un resumen de las herramientas que se han usado para la implementación del sistema de información.

**Tabla 6 Configuración necesaria para ejecutar el sistema de información**

| Herramienta/S.O./Plataforma | Versión | Descripción | URL |
|---|---|---|---|
| Oracle VM VirtualBox | 6.1 | Software de virtualización | https://www.virtualbox.org/ |
| Ubuntu | 22.04 | Sistema operativo | https://ubuntu.com/download |
| Java (openJDK) | 18.0.1.1 | Lenguaje de desarrollo | https://openjdk.org/ |
| Spring Boot | 2.7.5 | Framework de desarrollo | https://spring.io/projects/spring-boot |
| IntelliJ Idea Community | 222.4345 | IDE de desarrollo | https://www.jetbrains.com/idea/ |
| Postman | 9.31.0 | Plataforma para consumir APIs | https://www.postman.com/ |
| Docker | 20.10.17 | Proyecto para desplegar aplicaciones en contenedores | https://www.docker.com/ |
| Podman | 3.4.4 | Proyecto para desplegar aplicaciones en contenedores | https://podman.io/ |

Fuente: Elaboración propia

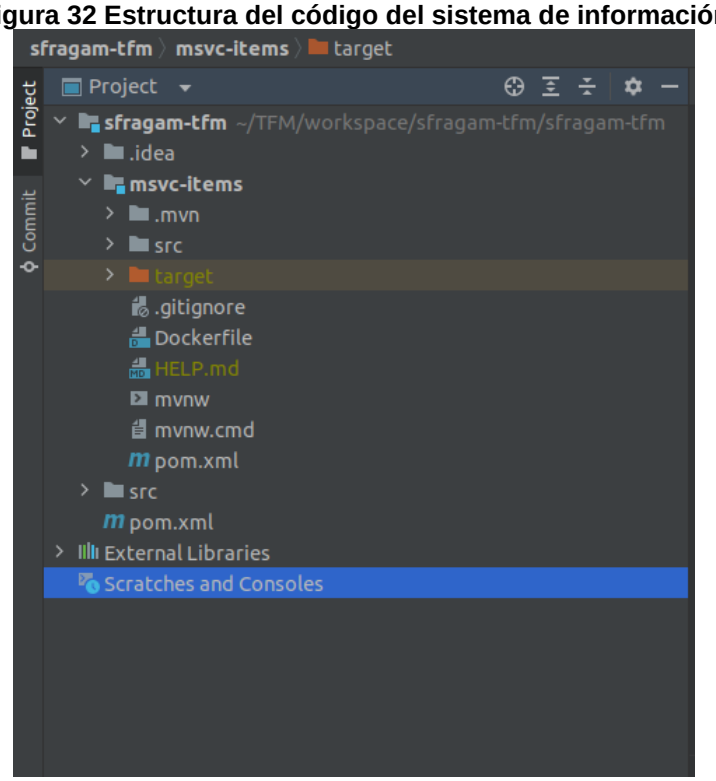
### Estructura usada en el sistema de información

La estructura que se ha optado por usar es la orientada a microservicios ya que actualmente es la arquitectura recomendada en el desarrollo de sistemas de información tanto simples como complejos.

Todo el código del proyecto en el cual nos basaremos se encuentra alojado en un repositorio público de Github el cual se encuentra accesible desde la siguiente dirección: https://github.com/sanframar/sfragam-TFM

Una vez descargado el código en nuestra máquina local nos encontraremos con la siguiente estructura:

**Figura 32 Estructura del código del sistema de información**




Fuente: Elaboración propia

Tal y como se puede observar en la figura superior, el sistema de información consta de un proyecto padre denominado "**sfragam-tfm**" y un proyecto hijo (microservicio) llamado "**msvc-items**".

Dentro del proyecto "msvc-items" tenemos todo el código necesario para que nuestra API funcione correctamente. No se detallará como se encuentra implementado dicho microservicio ya que se sale del alcance del trabajo Fin de Máster.

En la raíz del proyecto "msvc-items" podemos observar el fichero **Dockerfile** el cual usaremos para crear nuestra imagen y posteriormente **crear un contenedor en base a esta**.

### Sistema de información con Docker

A continuación, se detallarán los pasos que se han seguido para montar el sistema de información haciendo uso de Docker.

Lo primero que debemos hacer es crear la imagen para el servicio API, para ello usaremos un fichero llamado Dockerfile el cual contiene la configuración estática y servicios que tendrá nuestra imagen.

El fichero **Dockerfile** que se ha usado para la creación de la imagen tiene la siguiente forma:

**Figura 33 Dockerfile para crear la imagen del microservicio msvc-items**


Fuente: Elaboración propia

El fichero Dockerfile está compuesto de la siguiente forma:

- **Estructura:** Se sigue una estructura denominada **multi-stage builds,** esto quiere decir que la imagen se construye en diferentes etapas para de esta forma ahorrar espacio en el disco duro.
- **ARG MSVC_NAME:** Con este comando lo que hacemos es crear un una variable con el el valor “msvc-items” para poder reutilizar dicho valor en todo el fichero.
- **FROM:** Con este comando elegimos la imagen base de la que queremos partir, en este caso se trata de una imagen de openjdk.
- **as:** Con esta sentancia le damos un valor a esta primera etapa de compilación del proyecto
- **WORKDIR:** Con este comando elegimos el espacio de trabajo dentro de la imagen.
- **COPY:** Con la sentencia copy copiamos los ficheros necesarios desde nuestro host a la imagen.
- **RUN:** Con este comando somos capaces de ejecutar sentencias dentro de la imagen. Con el valor **“./mvnw clean package -Dmaven.test.skip -Dmaven.main.skip -Dspring-boot.repackage.skip && rm -r ./target/**” lo que realizamos es una descarga de las dependencias del proyecto mientras que con el valor “**./mvnw clean package -DskipTests**”

compilamos el proyecto y generamos el fichero .jar que usaremos en la segunda etapa para desplegar el servicio API.

- **EXPOSE:** Exponemos el puerto del contenedor, aunque verdaderamente lo que expone el puerto es el comando "-p <port>:<port>" a la hora de ejecutar el contenedor.
- **CMD:** Definimos los valores que se usarán al ejecutar nuestro contenedor, en este caso lo que hará será ejecutar el .jar que se ha construido en la fase anterior llamado "**msvc-items-0.0.1-SNAPSHOT.jar**"

La configuración que se ha usado para la creación de la imagen **no es la más segura** y los comandos usados son los **mínimos para ejectuar correctamente el sistema**. En el apartado de prueba de concepto aplicaremos ciertos comandos para mejorar la seguridad.

Para ejecutar el fichero **Dockerfile con Docker** nos situaremos en la raiz del proyecto padre y ejecutaremos el siguiente comando "**docker build -t msvc-items -f ./msvc-items/Dockerfile .".** El comando usa las siguientes banderas:

- **-t:** Indicamos el nombre que tendrá nuestra imagen.
- **-f:** Con este comando indicamos la ruta donde se encuentra nuestro fichero Dockerfile.

Tras ejecutar dicho comando se creará una imagen llamada "**msvc-items**" que usaremos para ejecutar el contenedor.

**Figura 34 Ejecución del Dockerfile para crear la imagen**

```
santiago@santiago-VirtualBox:~/TFM/workspace/sfragam-tfm/sfragam-tfm$ docker build -t msvc-items -f ./msvc-items/Dockerfile .
Sending build context to Docker daemon  139.3kB
Step 1/18 : ARG MSVC_NAME=msvc-items
Step 2/18 : FROM openjdk:18-jdk-alpine as mavenBuilder
 ---> c89120dcca4c
Step 3/18 : ARG MSVC_NAME
 ---> Using cache
 ---> ac00f2affded
Step 4/18 : WORKDIR /app/$MSVC_NAME
 ---> Using cache
 ---> 2a6da4e20d2b
Step 5/18 : COPY ./pom.xml /app
 ---> Using cache
 ---> 54bf89a80c8d
Step 6/18 : COPY ./$MSVC_NAME/.mvn ./.mvn
 ---> Using cache
 ---> f0e4bf9a244c
Step 7/18 : COPY ./$MSVC_NAME/mvnw .
 ---> Using cache
 ---> d327397f8492
Step 8/18 : COPY ./$MSVC_NAME/pom.xml .
 ---> Using cache
 ---> 321f8148ee40
Step 9/18 : RUN ./mvnw clean package -Dmaven.test.skip -Dmaven.main.skip -Dspring-boot.repackage.skip && rm -r ./target/
 ---> Using cache
 ---> 9ec659482be1
Step 10/18 : COPY ./$MSVC_NAME/src ./src
 ---> Using cache
```

Fuente: Elaboración propia

**Figura 35 Listado de imágenes que tenemos en el sistema Host**


Fuente: Elaboración propia

Antes de seguir con la creación de contenedores debemos crear una red en la cual se encontraran los dos contenedores, el de nuestra aplicación API y el correspondiente a la base de datos con MySQL.

Para crear nuestra red ejecutaremos el comando "**docker network create netitems**" y tras su creación podemos listar las redes que tenemos creadas para visualizar si se ha creado correctamente.

**Figura 36 Listado de networks que tenemos en el sistema Host**


Fuente: Elaboración propia

Tras crear la imagen y la red correctamente ahora ejecutaremos los dos contenedores que necesitamos para implementar el sistema de información, en primero lugar ejecutaremos el contenedor de MySQL.

Para ejecutar MySQL nos iremos primero a Docker Hub (https://hub.docker.com/) para buscar una imagen oficial de MySQL. Una vez localizada (https://hub.docker.com/_/mysql) seguiremos los pasos que se nos indica para ejecutar el contenedor.

El comando que se ha usado para crear el contenedor correspondiente a MySQL ha sido el siguiente: "**docker run -d -p 3307:3306 --name mysql --network netitems -e MYSQL_ROOT_PASSWORD=sasa -e MYSQL_DATABASE=msvc_items mysql**". A continuación se muestran con detalle las distintas variables que se han usado para la creación del contenedor:

- **-d:** Ejecutamos el contenedor en modo daemon, es decir, el contenedor se ejecutará en segundo plano.

- **-p:** Indicamos los puertos que van a ser expuesto, el formato es el siguiente -p <puerto externo>:<puerto interno>.
- **--name:** Modificamos el nombre del contenedor.
- **--network:** Seleccionamos la red a la que se conectará, es importante que los contenedores que requieran una comunicación entre ellos se encuentren en la misma red y tengan algún puerto expuesto.
- **-e:** Es un comando usado para dar valor a las variables que se encuentran dentro del contenedor, en este caso usamo la varible "**MYSQL_ROOT_PASSWORD**" para indicar las credenciales del usuario root y "**MYSQL_DATABASE**" para que el contenedor nos cree una base de datos con el nombre que le indicamos.

En este caso debemos tener en cuenta que los datos que se almacenen en la base de datos se borrarán cuando eliminemos el contenedor ya que no se están usando volumenes.

**Figura 37 Ejecución del contenedor que contiene el servicio de MySQL**


Fuente: Elaboración propia

Con dicho comando se nos creará un contenedor que ejecutará MySQL sobre el puerto 3306 (**puerto interno usado entre los contenedores**).

A continuación ejecutaremos el contenedor de nuestro aplicativo API con el comando "**docker run -d -p 8001:8001 --name msvc-items --network netitems msvc-items**".

**Figura 38 Ejecución del contenedor con el microservicio msvc-items**

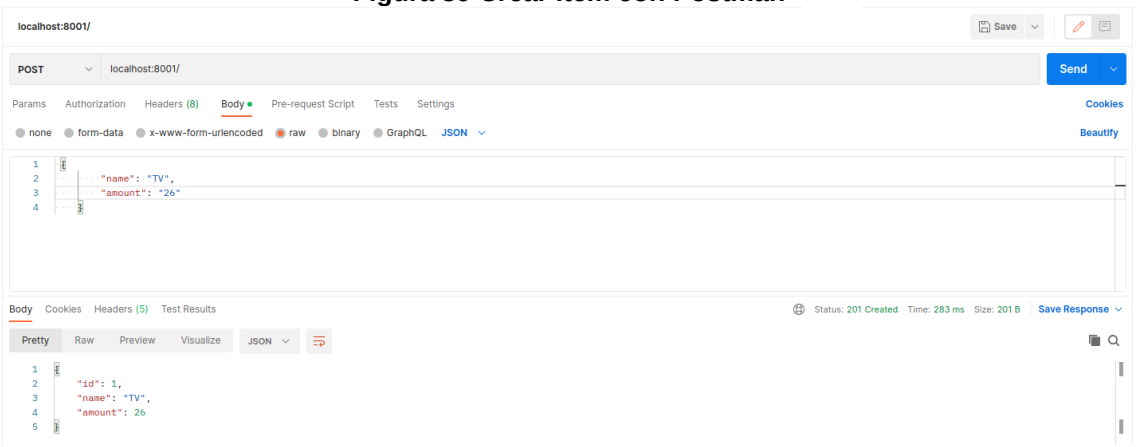

Fuente: Elaboración propia

A la hora de ejecutar los contenedores se ha utilizado los **comandos mínimos e indispensables** para ejecutar correctamente el sistema, ya que hasta ahora **no se ha aplicado ninguna acción para mejorar la seguridad** de los contenedores.

Una vez llegados a este punto lo único que queda es probar con Postman que la API funciona correctamente. Para ello accederemos a la herramienta y escribiremos la URL del aplicativo e intentaremos crear un objeto item.

Para crear un nuevo item enviaremos una petición **POST** con un **Json** con la estructura indicada en la siguiente ilustración a la url "**localhost:8001**". Si todo funciona correctamente nos devolverá el código 201 junto con el objeto creado.

**Figura 39 Crear item con Postman**




Fuente: Elaboración propia

A continuación, ejecutaremos el **GET** para obtener el item que acabamos de crear a la url "**localhost:8001**".

**Figura 40 Consultar items con Postman**


Fuente: Elaboración propia

Si todo ha funcionado adecuadamente nos devolverá un listado con los ítems creados, que en este caso en concreto es solo uno.

### Sistema de información con Podman

Para poder crear el sistema de información con **Podman** se deben seguir los mismos pasos que hemos realizado con Docker, además, los **comandos que**

**usa Podman son iguales a los comandos que se usan en Docker** por lo que la migración de un sistema a otro es muy sencilla.

Primero comenzaremos creando la imagen de nuestra API, para ello usaremos el Dockerfile creado para Docker con las siguientes modificaciones:

- A la hora de usar imágenes públicas de Docker se deben definir con la url "**docker.io/library/**" delante del nombre de la imagen. Esto se puede observar en los comandos de "**FROM**" presentes en las líneas 2 y 18.

**Figura 41 Dockerfile usado para crear la imagen con Podman**

Dockerfile

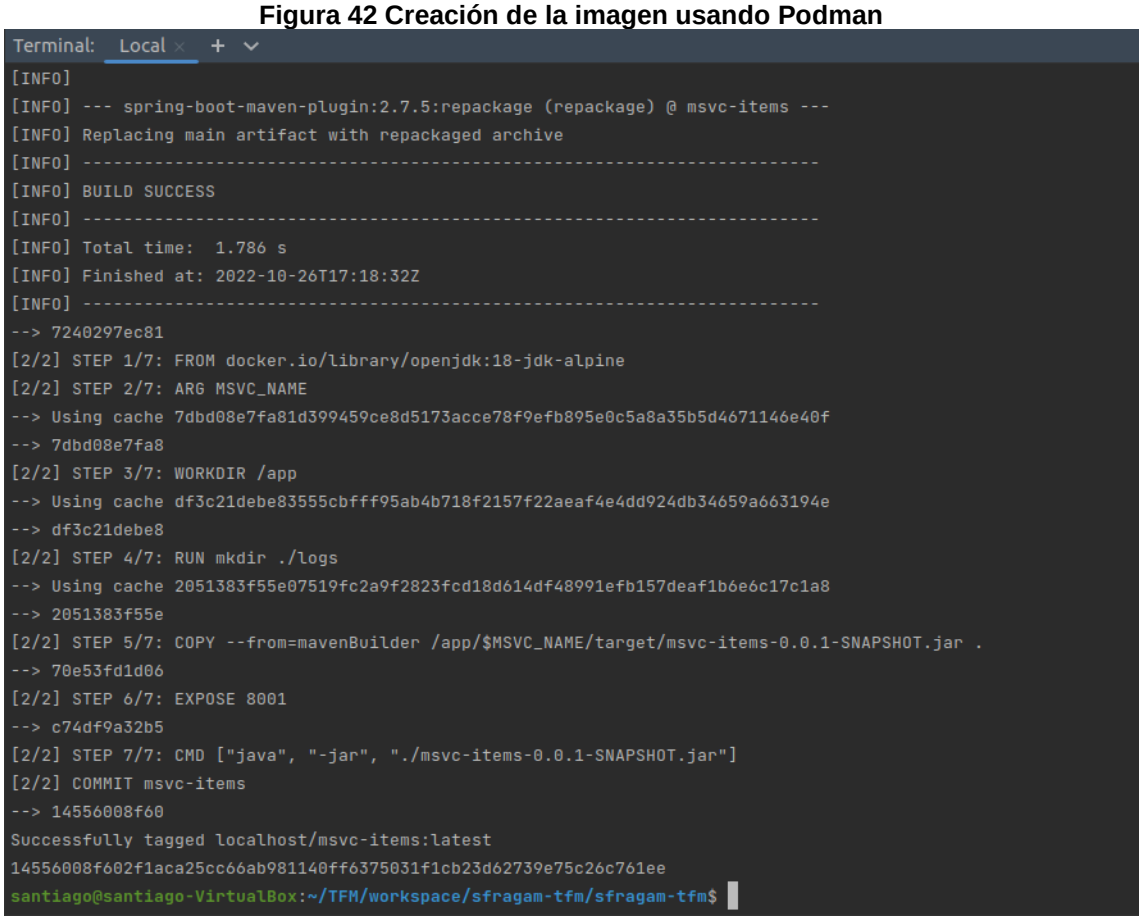
```
1  ARG MSVC_NAME=msvc-items
2  FROM docker.io/library/openjdk:18-jdk-alpine as mavenBuilder
3  ARG MSVC_NAME
4
5  WORKDIR /app/$MSVC_NAME
6
7  COPY pom.xml /app/
8  COPY ./$MSVC_NAME/.mvn ./.mvn
9  COPY ./$MSVC_NAME/mvnw .
10 COPY ./$MSVC_NAME/pom.xml .
11
12 RUN ./mvnw clean package -Dmaven.test.skip -Dmaven.main.skip -Dspring-boot.repackage.skip && rm -r ./target/
13
14 COPY ./$MSVC_NAME/src ./src
15
16 RUN ./mvnw clean package -DskipTests
17
18 FROM docker.io/library/openjdk:18-jdk-alpine
19 ARG MSVC_NAME
20 WORKDIR /app
21 RUN mkdir ./logs
22 COPY --from=mavenBuilder /app/$MSVC_NAME/target/msvc-items-0.0.1-SNAPSHOT.jar .
23
24 EXPOSE 8001
25 CMD ["java", "-jar", "./msvc-items-0.0.1-SNAPSHOT.jar"]
```

Fuente: Elaboración propia

Una vez realizado el cambio comentado, ya tendríamos nuestro fichero Dockerfile completado. Para la creación de la imagen ejecutamos el siguiente comando "**podman build -t msvc-items -f ./msvc-items/Dockerfile .**"

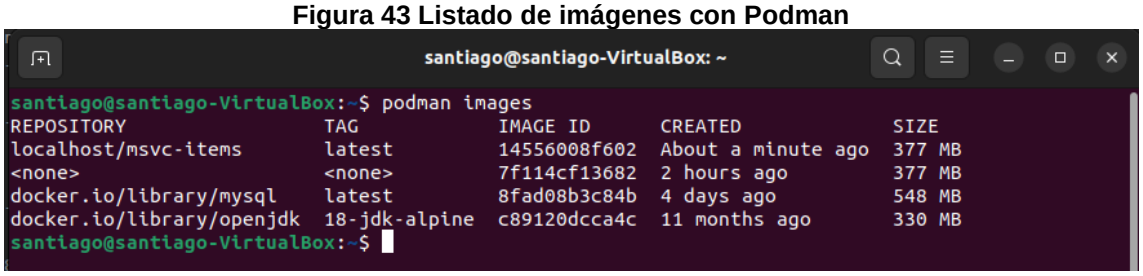
La configuración que se ha usado para la creación de la imagen **no es la más segura** y los comandos usados son los **mínimos para ejectuar correctamente el sistema**. En el apartado de prueba de concepto aplicaremos ciertos comandos para mejorar la seguridad.

**Figura 42 Creación de la imagen usando Podman**




Fuente: Elaboración propia

**Figura 43 Listado de imágenes con Podman**




Fuente: Elaboración propia

Tal y como podemos observar en las ilustraciones superiores la imagen se crea correctamente y la podemos visualizar con "**podman images**".

Antes de seguir con la creación de contenedores debemos crear una red en la cual se encontraran los dos contenedores, el de nuestra aplicación API y el correspondiente a la base de datos con MySQL.

Para crear nuestra red ejecutaremos el comando "**podman network create netitems**" y tras su creación podemos listar las redes que tenemos creadas para visualizar si se ha creado correctamente.

**Figura 44 Listado de Networks existentes en el sistema Host para Podman**


Fuente: Elaboración propia

Ahora comenzaremos ejecutando el contenedor que tendrá la base de datos MySQL. El comando usado es muy parecido al de Docker "**podman run -d -p 3307:3306 --name mysql --network netitems -e MYSQL_ROOT_PASSWORD=sasa -e MYSQL_DATABASE=msvc_items docker.io/library/mysql**"

**Figura 45 Ejecución del contenedor con el servicio de MySQL con Podman**


Fuente: Elaboración propia

Como podemos observar en la figura se ha creado correctamente el contenedor con MySQL y se está ejecutando con Podman.

Llegados a este punto crearemos el contenedor que ejecutará el servicio API lanzando el siguiente comando "**podman run -d -p 8001:8001 --name msvc-items --network netitems msvc-items".**

**Figura 46 Ejecución del contenedor con el servicio API con Podman**


Fuente: Elaboración propia

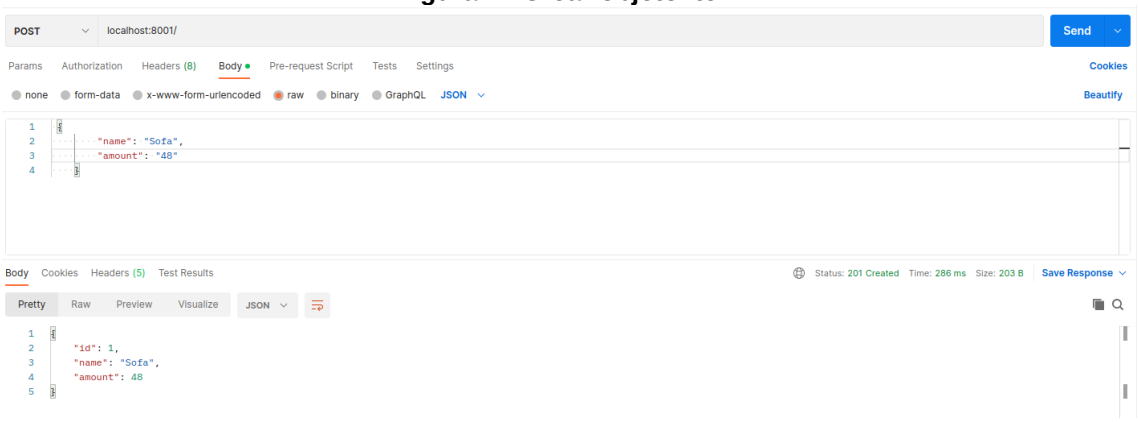
Tal y como se ha comentado en el apartado superior, **los comandos** que se han usado para la ejecución del contendor son los **mínimos e indispensables** para ejecutar correctamente el sistema de información por lo que **no se han aplicado ninguna medida de seguridad**.

Tras la ejecución del comando podemos observar que tanto el contenedor de MySQL como el de la API (msvc-items) están funcionando correctamente.

Ahora con ayuda de **Postman** realizaremos varias peticiones a nuestro servicio para crear un item y posteriormente mostrar un listado de los ítems almacenados en la base de datos.

Para crear un nuevo item enviaremos una petición **POST** con un **Json** con la estructura indicada en la siguiente ilustración a la url "**localhost:8001**". Si todo funciona correctamente nos devolverá el código 201 junto con el objeto creado.

**Figura 47 Crear objeto item**

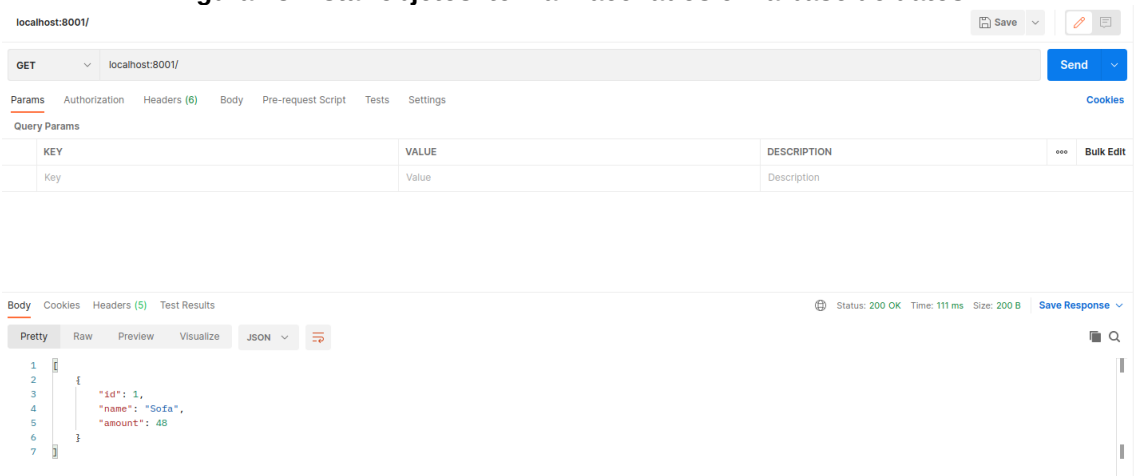



Fuente: Elaboración propia

Tal y como se puede observar el sistema nos devuelve el objeto item que ha creado acompañado del código **201 Created**.

A continuación, usaremos una petición **GET** para obtener el listado de elementos que hay en la base de datos de MySQL.

**Figura 48 Listar objetos item almacenados en la base de datos**




Fuente: Elaboración propia

La API nos devuelve el objeto que acabamos de crear acompañado del código **200 OK** por lo que el sistema de información está funcionando correctamente.

# Anexo II. Prueba de concepto

## *Política de mínimos privilegios*

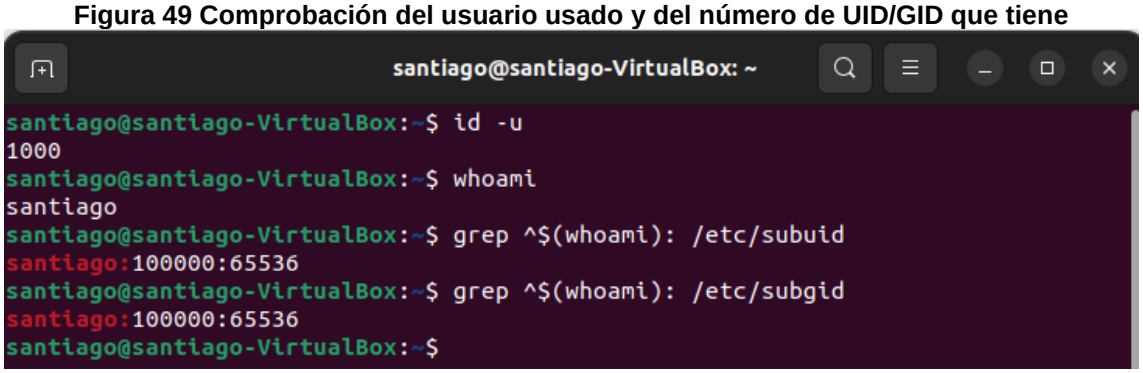
### Demonio de Docker

Lo primero que debemos hacer en el caso de **Docker** es lograr configurar el **demonio de Docker** para que se ejecute sin permiso de root, esta configuración en **Podman** no es necesaria ya que no utiliza ningún demonio (Docker, 2022).

Para configurar correctamente el demonio de Docker se ha seguido la guía propia que ofrece Docker y que se puede encontrar en el siguiente enlace https://docs.docker.com/engine/security/rootless/.

Antes de comenzar a configurar el demonio necesitamos una serie de prerrequisitos como son las librerías **newuidmap** y **newgidmap** además de que tanto **/etc/subuid** como **/etc/subgid** debe contener al menos **65.536 UID/GID** subordinados para el usuario.

**Figura 49 Comprobación del usuario usado y del número de UID/GID que tiene**




Fuente: Elaboración propia

Otro elemento que debemos tener en cuenta es la versión de Docker, ya que si tenemos una versión superior a la 20.10 ya deberíamos tener todos los paquetes necesarios como es el **dockerd-rootless-setuptool.sh.**

Antes de comenzar con la configuración necesaria consultaremos los permisos que tiene Docker actualmente en el sistema con el comando **ps auxw | grep docker.**

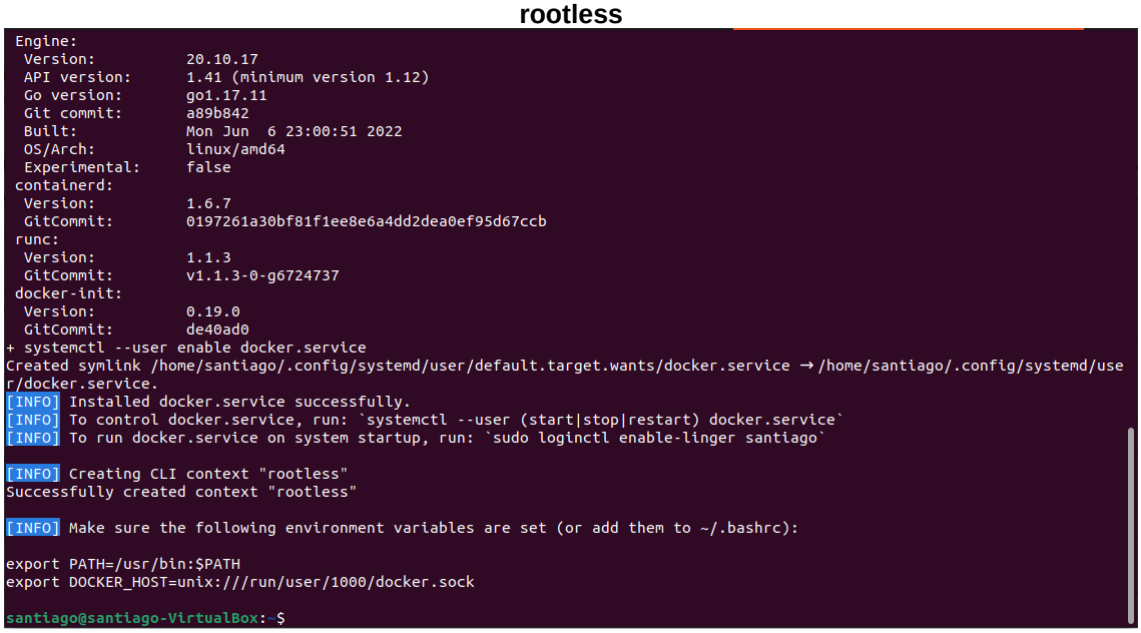
**Figura 50 Permisos que tiene Docker antes de configurar el demonio como rootless**


Fuente: Elaboración propia

Tal y como se puede observar en la primera posición nos encontramos con el permiso de root y posteriormente tenemos al usuario de santiago.

Llegados a este punto ejecutaremos el comando **dockerd-rootless-setuptool.sh install** y tras su ejecución obtendremos una salida como la que se muestra en la figura siguiente.

**Figura 51 Creación del contexto necesario para ejecutar el demonio de Docker como rootless**




Fuente: Elaboración propia

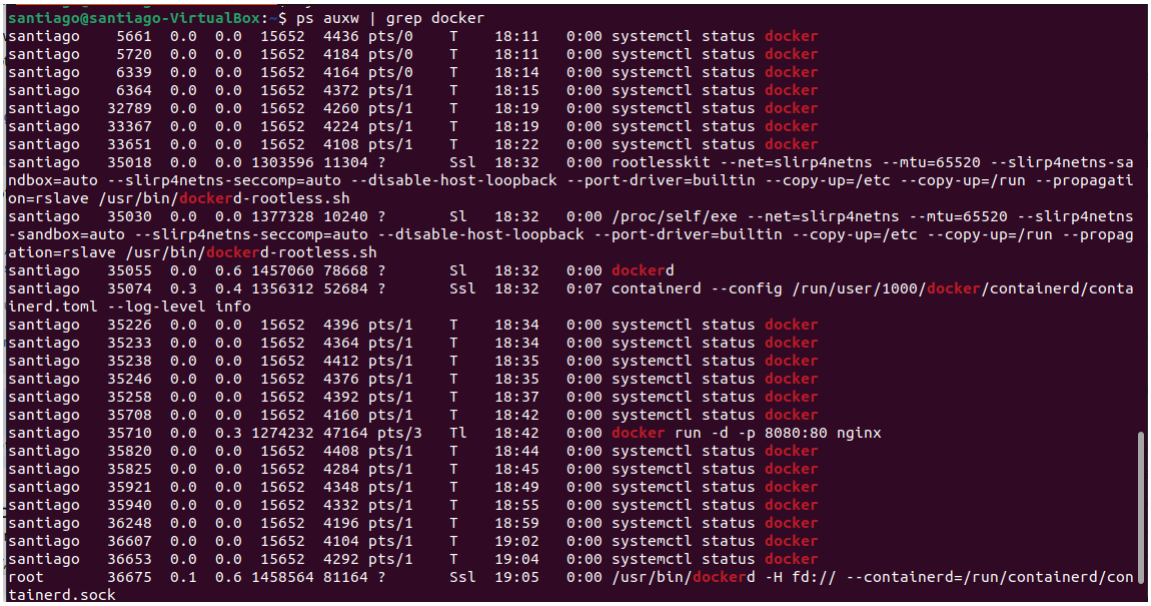
Tal y como podemos observar, nos dice que se ha ejecutado correctamente y que ahora podemos usar el comando "**--user**" para inicializar el servicio de Docker, por otro lado, también se nos confirma que se ha creado satisfactoriamente el **contexto de rootless**.

A continuación, volveremos a ejecutar el servicio de Docker y volveremos a comprobar los permisos que tenemos. Además, ejecutaremos un pequeño contenedor de nginx para ver cuáles son los permisos que Docker le asigna.

Para ejecutar el contenedor de **Nginx** lanzaremos le siguiente comando Docker **run -d -p 8080:80 nginx.**

Para ver los permisos que tenemos volveremos a usar el comando **ps auxw | grep docker.**

**Figura 52 Mostrar los permisos de Docker después de configurar el Daemon como rootless**




Fuente: Elaboración propia

Tal y como podemos observar, esta vez tenemos muchos más permisos referentes al usuario de “santiago” y nos aparece “**usr/bin/dockered-rootless.sh**”.

Además, podemos visualizar que el contenedor de nginx que se ha creado se está ejecutando sin permisos de **root**, en su lugar lo está haciendo con los permisos de “santiago”.

Toda esta configuración del demonio de **Docker no es necesaria realizarla para Podman**, ya que afortunadamente Podman no requiere de un demonio para su funcionamiento. Gracias a esta característica de Podman, dicha herramienta logra ser algo **más segura con la configuración que trae por defecto**.

### Permisos dentro del contenedor

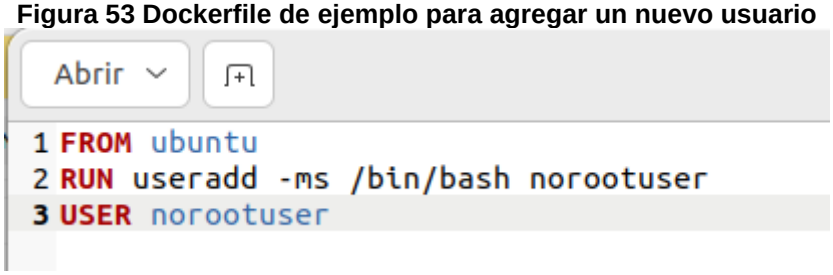
Hay varias formas de poder realizar dicho paso, pero la escogida para esta prueba de concepto ha sido la de introducir un **nuevo comando en el fichero Dockerfile** para que cuando se ejecute el contendor use un usuario sin permisos de root (Oosterhof, 2022).

El comando que debemos usar en el fichero Dockerfile para agregar un usuario es el siguiente “**RUN useradd -ms /bin/bash nortootuser**”.

Posteriormente, para usar dicho usuario debemos lanzar la sentencia “**USER myuser**”. Dicha sentencia lo que provoca es que a partir de esta, todas las acciones que se hagan en el fichero serán realizadas por el usuario que acabamos de crear.

Un ejemplo muy sencillo de cómo funciona lo podemos ver en el siguiente fichero creado para probar esta configuración.

**Figura 53 Dockerfile de ejemplo para agregar un nuevo usuario**

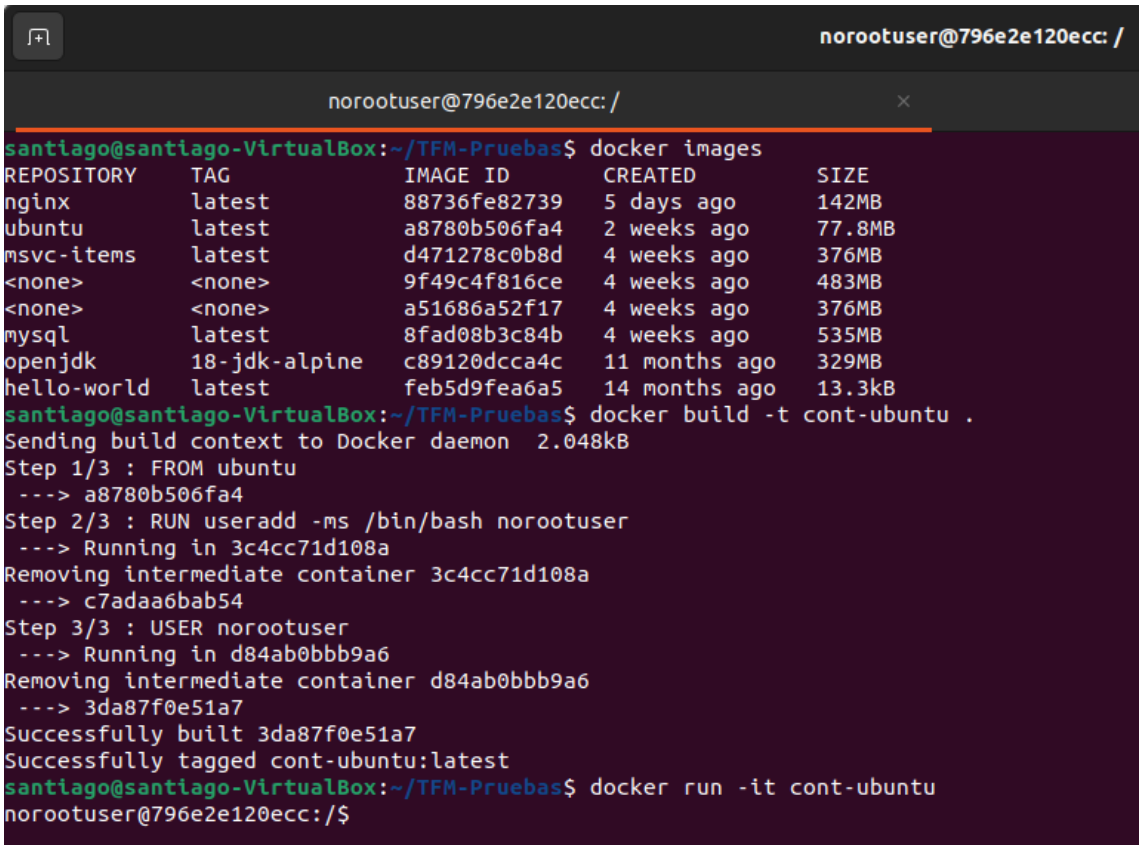



Fuente: Elaboración propia

En la figura de la parte superior podemos observar cómo es la creación de una simple imagen basada en Ubuntu en la cual creamos un usuario con el comando "**RUN useradd -ms /bin/bash nortootuser**" y luego hacemos uso de él con el comando "**USER norootuser**".

Para comprobar que funciona correctamente ejecutaremos un contenedor de forma interactiva ejecutando el comando "**docker run -it cont-ubuntu**". En la siguiente ilustración podemos ver todo el proceso, incluido el de creación de la imagen.

**Figura 54 Ejecución del contenedor para comprobar que se ha agregado correctamente el nuevo usuario**




Fuente: Elaboración propia

Tal y como se puede observar, al conectarnos de forma interactiva al contenedor estamos haciendo uso del usuario que hemos creado, el cual carece de permisos de root, es decir, es rootless.

Para pasar esta configuración a nuestro fichero Dockerfile con el cual estamos trabajando deberemos usar el siguiente comando "**RUN adduser -D norootuser**", el comando es algo distinto ya que en este caso se está haciendo uso de una imagen alpine que es ligeramente distinta a Ubuntu (Baeldung, 2022).

El fichero completo quedaría de la siguiente forma:

**Figura 55 Fichero modificado para agregar un usuario no root en el último paso**

```
1 ARG MSVC_NAME=msvc-items
2 FROM openjdk:18-jdk-alpine as mavenBuilder
3 ARG MSVC_NAME
4
5 WORKDIR /app/$MSVC_NAME
6
7 COPY ./pom.xml /app
8 COPY ./$MSVC_NAME/.mvn ./.mvn
9 COPY ./$MSVC_NAME/mvnw .
10 COPY ./$MSVC_NAME/pom.xml .
11
12 RUN ./mvnw clean package -Dmaven.test.skip -Dmaven.main.skip -Dspring-boot.repackage.skip && rm -r ./target/
13
14 COPY ./$MSVC_NAME/src ./src
15
16 RUN ./mvnw clean package -DskipTests
17
18 FROM openjdk:18-jdk-alpine
19 ARG MSVC_NAME
20 WORKDIR /app
21 RUN mkdir ./logs
22 COPY --from=mavenBuilder /app/$MSVC_NAME/target/msvc-items-0.0.1-SNAPSHOT.jar .
23
24 RUN adduser -D norootuser
25 USER norootuser
26
27 EXPOSE 8001
28 CMD ["java", "-jar", "./msvc-items-0.0.1-SNAPSHOT.jar"]
```

Fuente: Elaboración propia

Una vez realizado dicho cambio los pasos para la creación de la imagen y del contenedor son exactamente iguales, pero esta vez se ejecutará con el usuario que acabamos de crear llamado "**norootuser**".

## *Aislamiento de contenedores*

### AppArmor

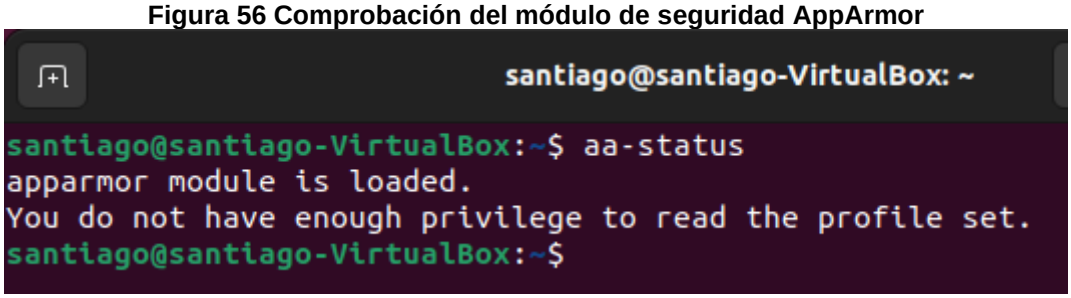
Para realizar esta tarea se hará uso del módulo de seguridad **AppArmor**, el cual implementa un sistema de control obligatorio de acceso. Al usar este módulo el núcleo de Linux pregunta a AppArmor antes de realizar una llamada al sistema para ver si el proceso está autorizado a hacerla (Debian, 2022; Docker, 2022 y Deberdt, 2022).

Gracias a esta herramienta podemos conseguir un aislamiento mucho más robusto de los contenedores.

A continuación, se detallan los pasos necesarios para aislar correctamente nuestros contenedores.

Lo primero que se debe hacer es **comprobar que AppArmor se encuentra habilitado** en la máquina, para ello ejecutaremos el comando "**aa-status**".

**Figura 56 Comprobación del módulo de seguridad AppArmor**




Fuente: Elaboración propia

Tal y como se puede observar en la figura superior, AppArmor se encuentra activado en el sistema actual.

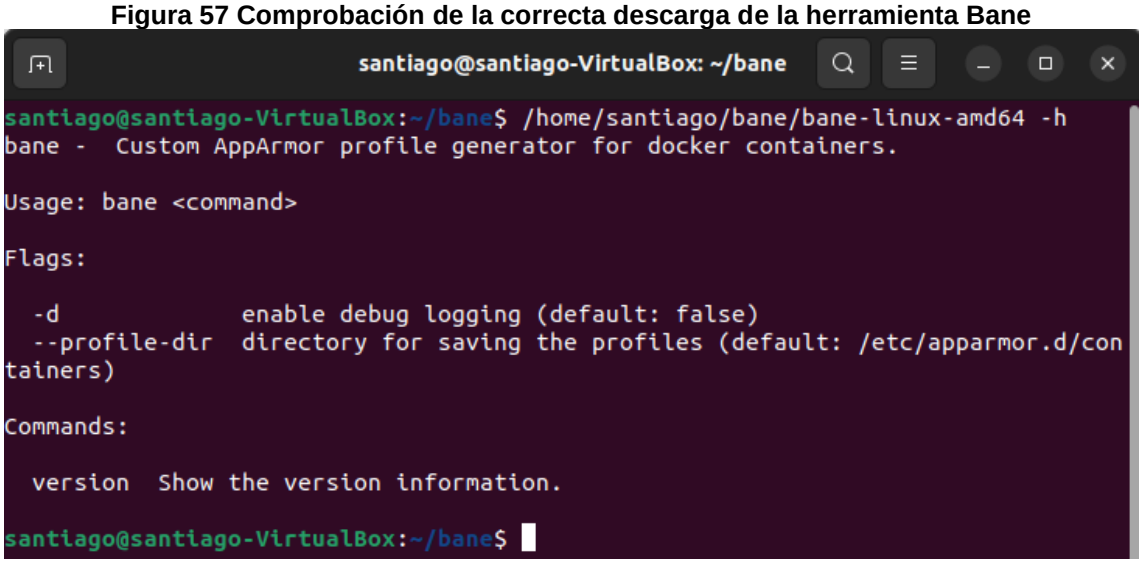
Como se ha comentado anteriormente, **AppArmor usa perfiles** para comprobar si un proceso está autorizado a realizar una acción determinada. Esto quiere decir que debemos crear **un perfil por cada contenedor** para aplicar las reglas.

Para crear un contenedor usaremos una herramienta para facilitarnos esta tarea, la herramienta en cuestión se llama **bane** y es capaz de crear esqueletos de perfiles para contenedores.

Para usar la herramienta bane nos iremos al repositorio Github https://github.com/genuinetools/bane y nos descargaremos la versión que corresponda al sistema operativo del host.

Una vez descargada ejecutaremos bane con el flag -h, en este caso en concreto el comando usado ha sido "**/home/santiago/bane/bane-linux-amd64 -h**".

**Figura 57 Comprobación de la correcta descarga de la herramienta Bane**




Fuente: Elaboración propia

A continuación, a través de una plantilla se creará el perfil que usaremos. La plantilla que se usará es la siguiente:

**Figura 58 Plantilla para crear el perfil que se usará en AppArmor**

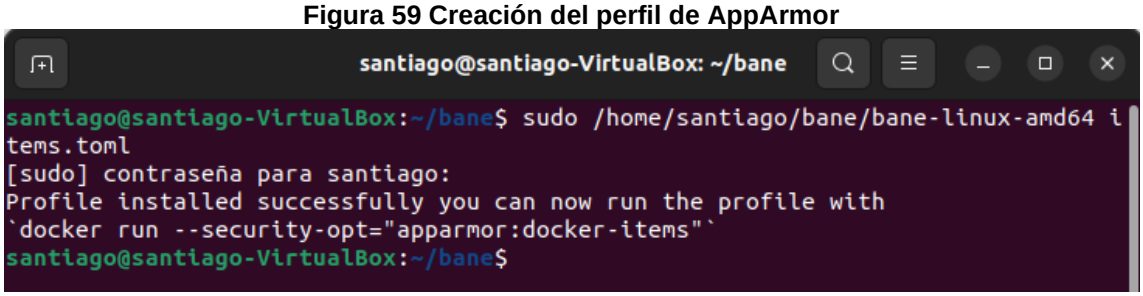
Abrir — items.toml ~/bane

```
Name = "items"

[Filesystem]
# read only paths for the container
ReadOnlyPaths = [
        "/bin/**",
        "/boot/**",
        "/dev/**",
        "/etc/**",
        "/home/**",
        "/lib/**",
        "/lib64/**",
        "/media/**",
        "/mnt/**",
        "/opt/**",
        "/proc/**",
        "/root/**",
        "/sbin/**",
        "/srv/**",
        "/tmp/**",
        "/sys/**",
        "/usr/**",
]

[Network]
# deny raw sockets
Raw = false
# deny packet sockets which allow to manage raw packets
# at the device driver level
Packet = false
Protocols = [
        "tcp",
]
```

Fuente: Elaboración propia

A continuación ejecutaremos la plantilla con el comando “**bane ítems.toml**”.

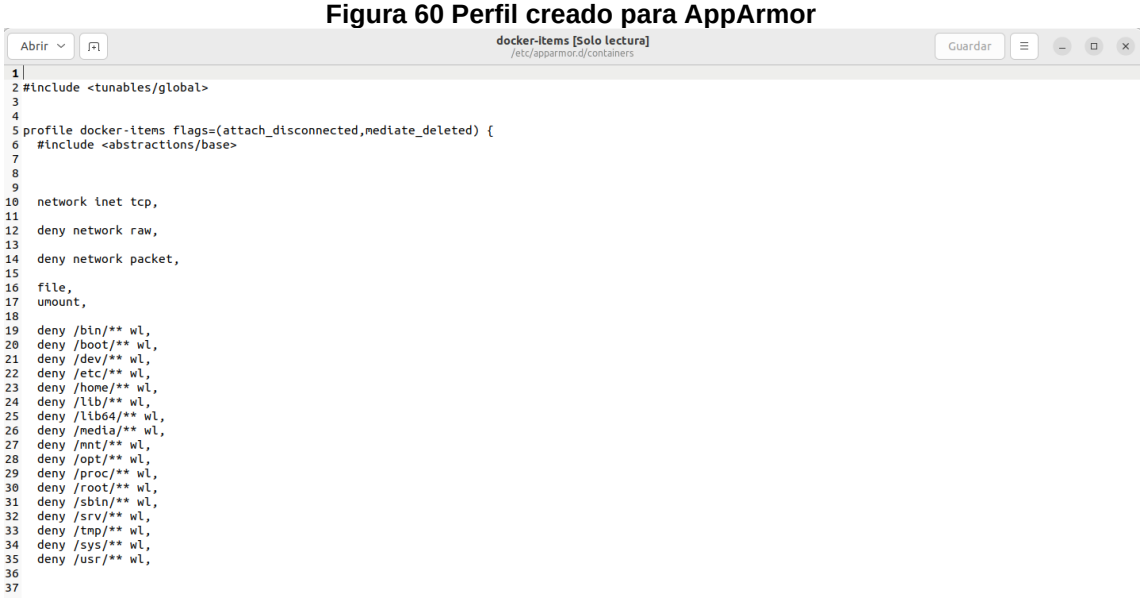
**Figura 59 Creación del perfil de AppArmor**




Fuente: Elaboración propia

El perfil que se ha creado tiene la siguiente forma:

**Figura 60 Perfil creado para AppArmor**

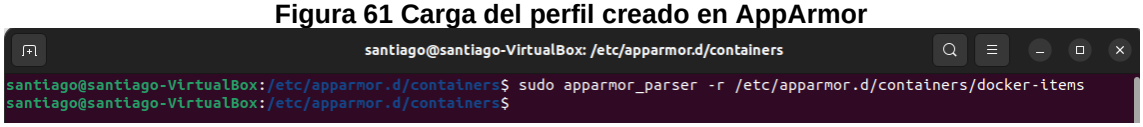



Fuente: Elaboración propia

Antes de continuar usaremos el modo auditoría de AppArmor para que de esta forma nos proporcione más información cuando estemos aplicando el perfil. Para hacer esto agregaremos el comando "**audit,complain**" en la cabecera del perfil (línea 5).

Por último, cargaremos el perfil que se acaba de crear con el comando "**sudo apparmor_parser -r /etc/apparmor.d/containers/docker-items**".

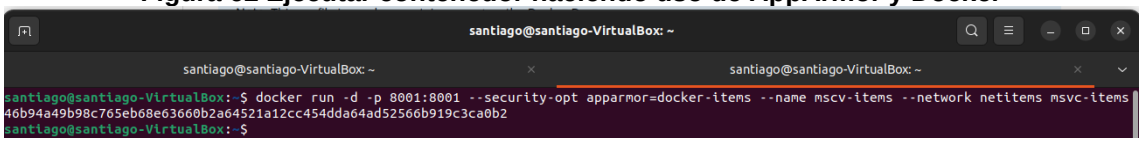
**Figura 61 Carga del perfil creado en AppArmor**




Fuente: Elaboración propia

Ahora ejecutaremos el contendor con Docker haciendo uso del perfil que se ha creado para AppArmor, el comando completo se puede visualizar en la siguiente figura.

**Figura 62 Ejecutar contenedor haciendo uso de AppArmor y Docker**




Fuente: Elaboración propia

El nuevo flag que se ha usado ha sido **“--security-opt apparmor=docker-items**”.

Tras lanzar el contendor podemos acceder al registro de logs de AppArmor para visualizar los permisos que se le está pidiendo y si están autorizados. Para ello podemos ejecutar el comando “**sudo tail -f /var/log/kern.log**”.

**Figura 63 Log del módulo de seguridad AppArmor al ejecutar un contenedor**


Fuente: Elaboración propia

Con esta simple configuración podemos hacer que el contenedor solo acceda a los recursos que necesite restringiendo el acceso a todos los demás.

A la hora de implementar en Podman la sentencia es exactamente igual, el comando usado es el siguiente “**podman run -d -p 8001:8001 --security-opt apparmor=Docker-items --name msvc-items --network netitems msvc-items**”.

Lamentablemente el uso de **AppArmor** en **Podman** suele presentar problemas y a veces es necesario el uso de la cláusula “**sudo**” para poder ejecutarlo correctamente.

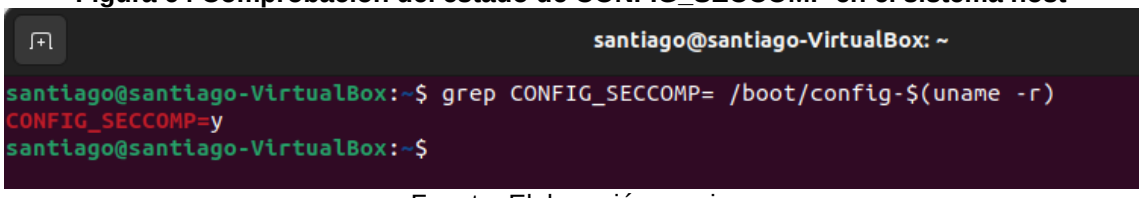
Gracias al uso de **AppArmor** los **contenedores** pueden estar **mejor aislados** tanto del sistema host como de otros servicios que se estén ejecutando en otros contenedores.

### Seccomp

Existe otro camino (más recomendado) para controlar las llamadas que hacen los contenedores al sistema host y es haciendo uso de las directivas **seccomp** que nos proporciona el kernel de Linux (Docker, 2022; Singh, 2022; y The kernel development community, 2022).

Antes de continuar debemos asegurarnos de que el kernel de Linux tiene habilitado seccomp para ello ejecutaremos el siguiente comando "**grep CONFIG_SECCOMP= /boot/config-$(uname -r)**".

**Figura 64 Comprobación del estado de CONFIG_SECCOMP en el sistema host**




Fuente: Elaboración propia

**Seccomp** funciona con perfiles los cuales son cargados a la hora de ejecutar el contendor. Estos perfiles contienen las reglas necesarias para evitar llamadas no deseadas al sistema host por parte del contenedor.

El perfil que se nos recomienda para lanzar contenedores se puede visualizar desde el siguiente enlace "**https://github.com/moby/moby/blob/master/profiles/seccomp/default.json"** y será el que usaremos para la ejecución de los contenedores.

El perfil viene en formato Json y tiene la siguiente forma:

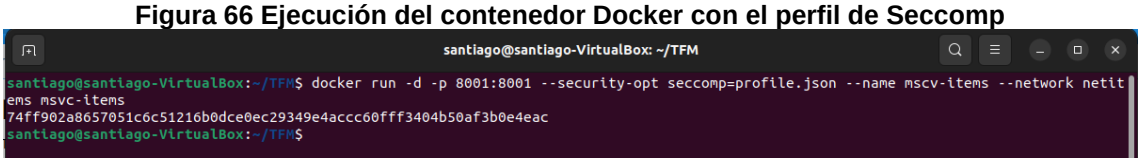
**Figura 65 Perfil Seccomp usado para ejecutar contenedores**

```
{
        "defaultAction": "SCMP_ACT_ERRNO",
        "defaultErrnoRet": 1,
        "archMap": [
                {
                        "architecture": "SCMP_ARCH_X86_64",
                        "subArchitectures": [
                                "SCMP_ARCH_X86",
                                "SCMP_ARCH_X32"
                        ]
                },
                {
                        "architecture": "SCMP_ARCH_AARCH64",
                        "subArchitectures": [
                                "SCMP_ARCH_ARM"
                        ]
                },
                {
                        "architecture": "SCMP_ARCH_MIPS64",
                        "subArchitectures": [
                                "SCMP_ARCH_MIPS",
                                "SCMP_ARCH_MIPS64N32"
                        ]
                },
                {
                        "architecture": "SCMP_ARCH_MIPS64N32",
                        "subArchitectures": [
                                "SCMP_ARCH_MIPS",
                                "SCMP_ARCH_MIPS64"
                        ]
                },
                {
                        "architecture": "SCMP_ARCH_MIPSEL64",
                        "subArchitectures": [
                                "SCMP_ARCH_MIPSEL",
                                "SCMP_ARCH_MIPSEL64N32"
                        ]
                },
                {
                        "architecture": "SCMP_ARCH_MIPSEL64N32",
                        "subArchitectures": [
```

Fuente: Elaboración propia

Al ejecutar el contenedor con el perfil mencionado obtenemos la siguiente salida:

**Figura 66 Ejecución del contenedor Docker con el perfil de Seccomp**

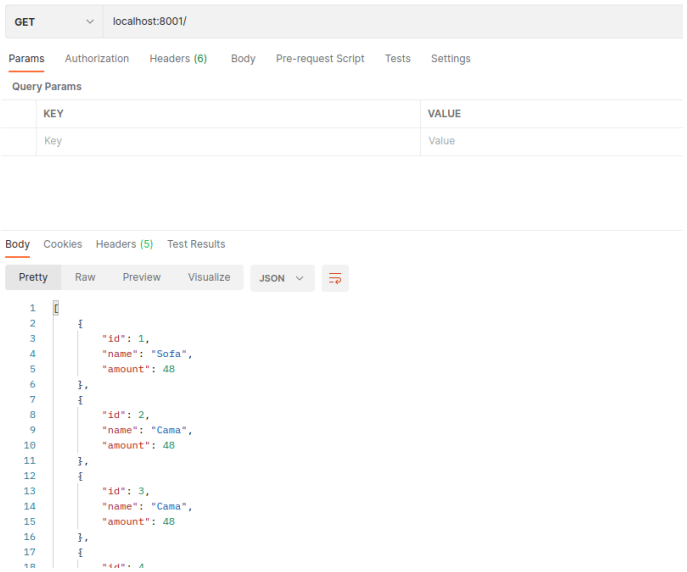



Fuente: Elaboración propia

Tal y como se puede observar en la figura superior hemos usado la cláusula **"--security-opt seccomp=profile.json**" para cargar el perfil. Dicho perfil se encuentra en la ruta base de la terminal.

Si accedemos a la dirección **localhost:8001/** podemos observar que el sistema sigue funcionando con normalidad.

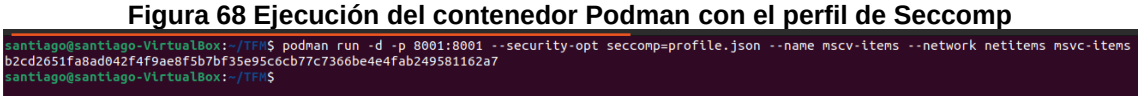
**Figura 67 Petición al sistema API para obtener un listado con los items de la base de datos**




Fuente: Elaboración propia

A la hora de implementar **seccomp** con **Podman** tanto la sentencia usada como el perfil de seccomp son los mismos que se han usado en Docker.

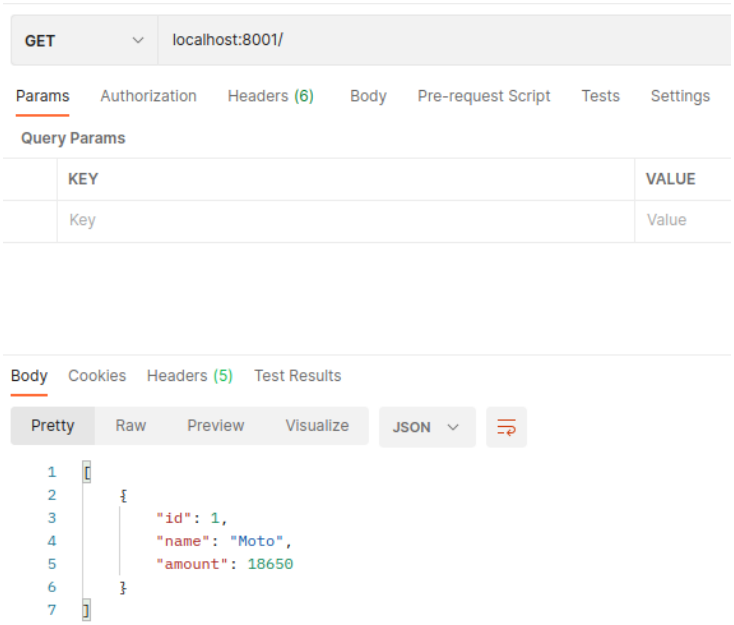
**Figura 68 Ejecución del contenedor Podman con el perfil de Seccomp**




Fuente: Elaboración propia

Una vez ejecutado el contendor haciendo uso del perfil de seccomp comprobamos que **todo funciona correctamente**.

**Figura 69 Petición al sistema API para obtener un listado con los items de la base de datos**



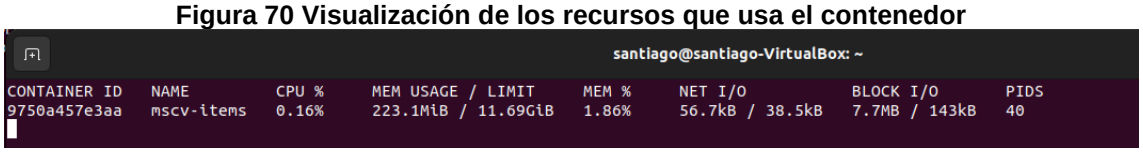

Fuente: Elaboración propia

*Limitación de recursos*

En este apartado vamos a restringir el uso de los recursos por parte del contenedor que contiene el microservicio de “msvc-items”.

Antes de limitar los recursos al contenedor debemos saber aproximadamente los recursos que necesita, para ello se ejecuta el comando “**docker stats msvc-items**” el cual nos dice los recursos que está consumiendo (Docker, 2022).

**Figura 70 Visualización de los recursos que usa el contenedor**




Fuente: Elaboración propia

Una vez se sabe aproximadamente los recursos que necesita los limitaremos.

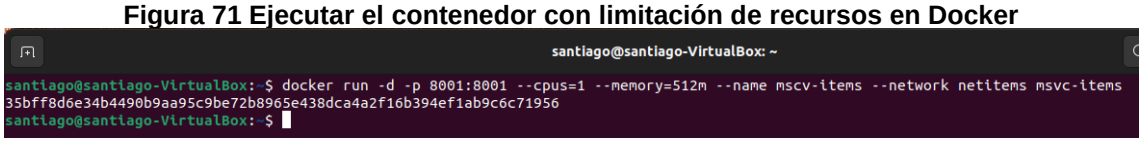
Primero **limitaremos** el uso de **CPU** con el comando “**--cpus=1**” lo que significa que el contenedor solo podrá hacer uso de una CPU del sistema host (Docker, 2022).

Por otro lado, para **limitar** la **memoria** que usa ejecutaremos el comando **“--memory 512m**” lo que quiere decir que el contenedor solo podrá hacer uso de un total de 512Mb de memoria RAM.

A continuación, agregamos los comandos comentados a la hora de crear nuestro contenedor de tal forma que la sentencia completa quedaría de la

siguiente forma “**docker run -d -p 8001:8001 --cpus=1 --memory=512m --name mscv-items --network netitems msvc-items**”.

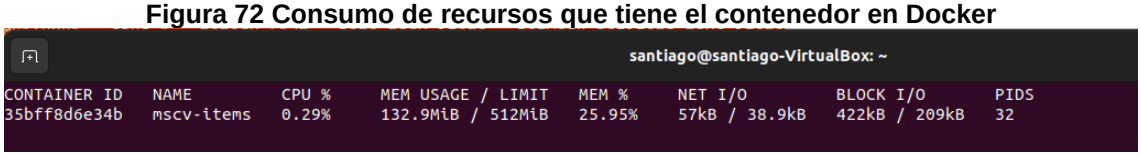
**Figura 71 Ejecutar el contenedor con limitación de recursos en Docker**




Fuente: Elaboración propia

Una vez ejecutado el contendor con los comandos mencionados volvemos a mirar los recursos que está usando y podemos ver como el límite de la RAM ha disminuido hasta 512Mb.

**Figura 72 Consumo de recursos que tiene el contenedor en Docker**



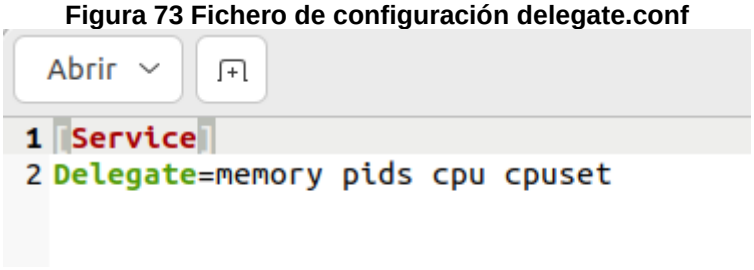

Fuente: Elaboración propia

Para limitar los recursos en Podman es exactamente igual, es decir, el comando que se debe usar es el siguiente “**podman run -d -p 8001:8001 --cpus=1 --memory=512m --name mscv-items --network netitems msvc-items**” (Podman, 2022).

En el caso de Podman para poder hacer uso del comando “--cpus” antes debemos activarlo en el sistema host. Ya que si no lo hacemos Podman arrojará un error “**Error: OCI runtime error: crun: the request cgroup controller ‘cpuset’ is not available**” (Github, 2022).

Para ello debemos crear un archivo en la ruta “/etc/systemd/system/user@.service.d/delegate.conf” que tenga el siguiente contenido.

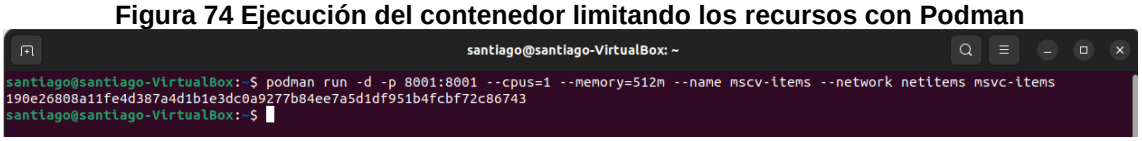
**Figura 73 Fichero de configuración delegate.conf**




Fuente: Elaboración propia

Una vez realizada dicha configuración Podman es capaz de limitar adecuadamente el uso de la CPU y podemos levantar nuestro contendor.

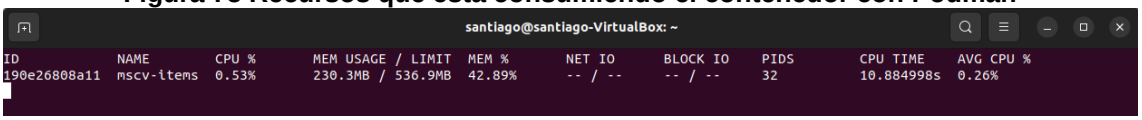
**Figura 74 Ejecución del contenedor limitando los recursos con Podman**




Fuente: Elaboración propia

Además, tal y como se hizo con Docker podemos ver los recursos que está consumiendo el contendor con el comando "**podman stats mscv-items**".

**Figura 75 Recursos que está consumiendo el contenedor con Podman**




Fuente: Elaboración propia

## *Búsqueda de vulnerabilidades*

La herramienta elegida es Trivy ya que es gratuita. Para su instalación en el sistema host usaremos el comando "**sudo apt-get install trivy**" y seguiremos la guía de instalación que se puede encontrar en la dirección https://aquasecurity.github.io/trivy/v0.34/getting-started/installation/

Una vez instalada la herramienta comprobaremos que se ha instalado correctamente comprobando la versión de esta (Trivy, 2022).

**Figura 76 Comprobación de la versión instalada de Trivy**


Fuente: Elaboración propia

Una vez en este punto, Trivy ya estaría listo para escanear imágenes, no obstantes a continuación se detallan algunos flag para personalizar dicho escaneo de imágenes (Trivy, 2022).

- **--severity**= Con este flag indicamos la severidad de las vulnerabilidades que queremos buscar.
- **--format**= Indicamos le formato de salida.

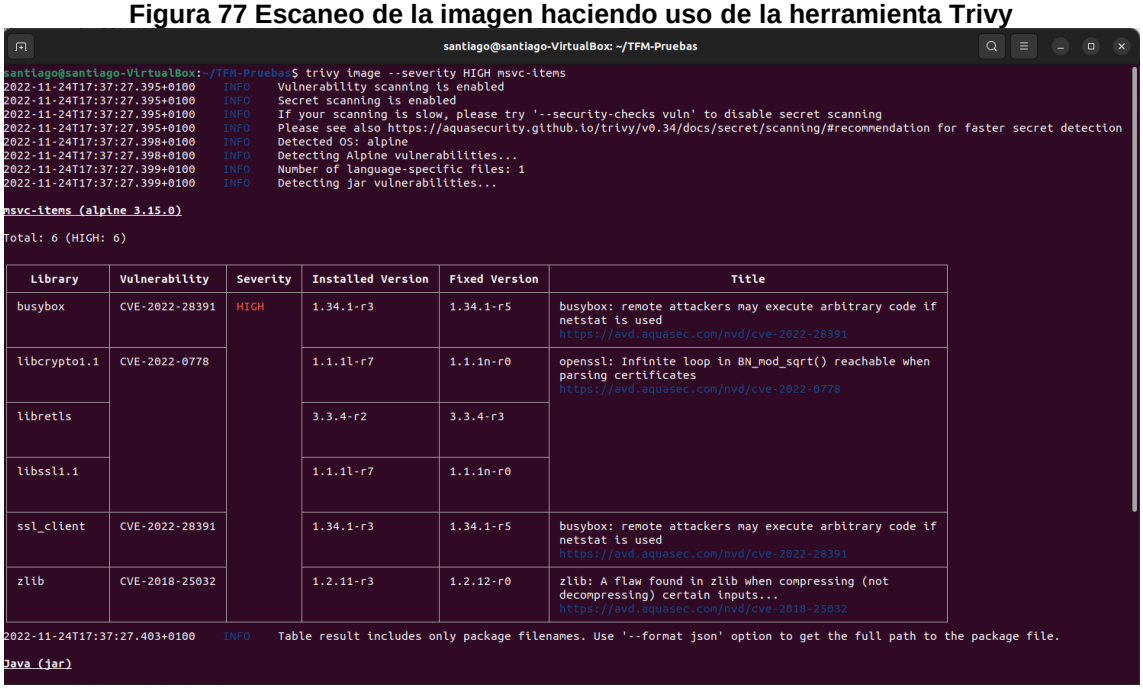
- **--output**= Le decimos el fichero de salida que queremos que nos genere.

A continuación, realizaremos un escaneo de la imagen llamada “msvc-items” que corresponde a nuestro servicio de API (Trivy, 2022).

Primero realizaremos un escaneo indicando solamente el nivel de severidad que queremos que busque.

El comando usado para realizar este escaneo es el siguiente “**trivy image --severity HIGH msvc-items**”.

**Figura 77 Escaneo de la imagen haciendo uso de la herramienta Trivy**



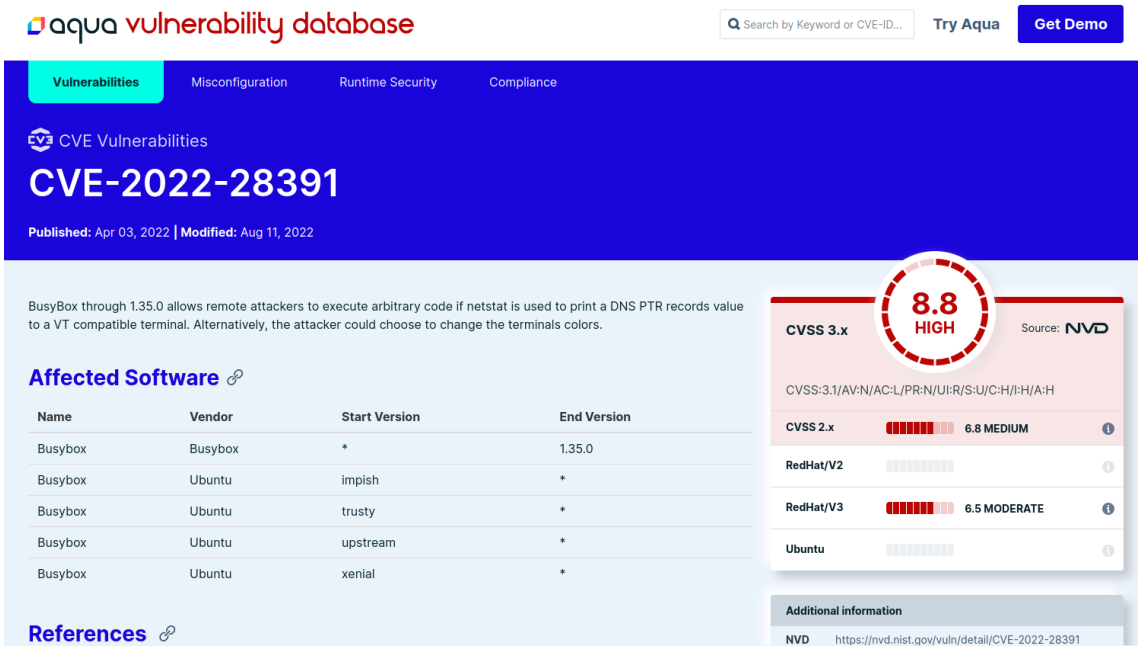

Fuente: Elaboración propia

Tal y como podemos visualizar en la figura superior, Trivy nos devuelve en la misma consola el resultado del escaneo organizado en tablas.

Podemos observar que tenemos varias vulnerabilidades potenciales del tipo “HIGH”, además nos da información sobre el código que identifica a la vulnerabilidad y en que versión ha sido corregida.

Por último, en la columna llamada “Title” nos dá más información sobre dicha vulnerabilidad y una URL en la cual tenemos más información al respecto.

**Figura 78 Información detallada de la vulnerabilidad**




Fuente: Elaboración propia

Ahora ejecutaremos de nuevo un escaner pero usando más opciones para de esta forma generar un fichero json, el comando usado para realizar dicho escaneo ha sido el siguiente "**trivy image --severity HIGH --format json --output scan.json msvc-items**" (Trivy, 2022).

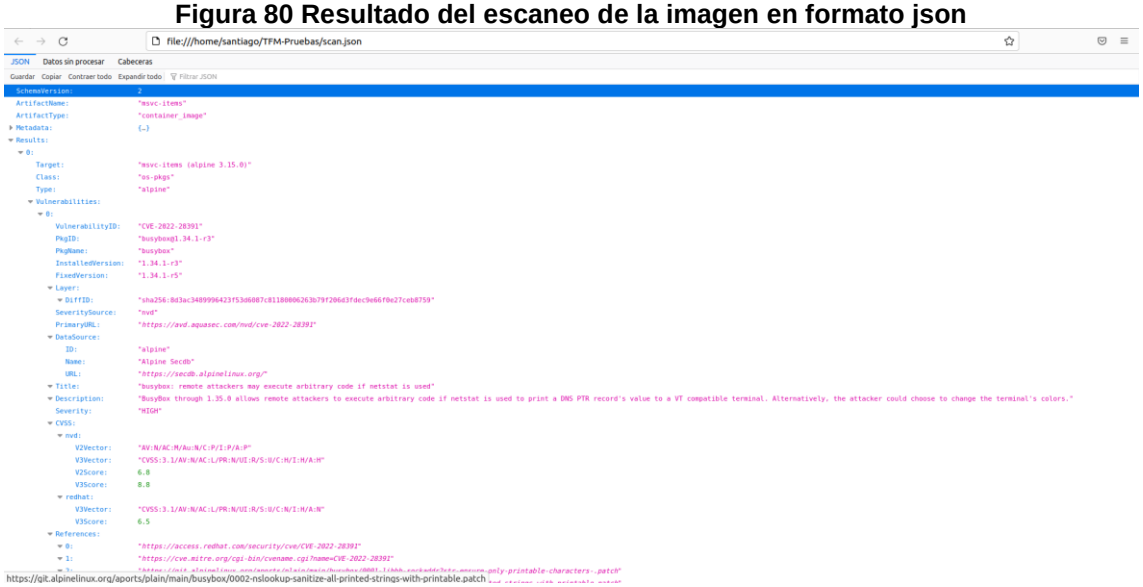
**Figura 79 Escaneo de la imagen msvc-items haciendo uso de la herramienta Trivy en formato json**


Fuente: Elaboración propia

Tras la ejecución del comando superior la herramienta crea un fichero llamado "scan.json" en formato json en el directorio en el que se encuentra la terminal con el resultado del escaneo.

Dicho fichero lo podemos abrir con un navegador o puede ser procesado por un servicio.

**Figura 80 Resultado del escaneo de la imagen en formato json**




Fuente: Elaboración propia

## *Implementación de cortafuegos*

Antes de comenzar debemos tener en cuenta que **Docker** por defecto **se salta la configuración de UFW** que tengamos en el sistema por lo que requiere de una configuración adicional, en cambio **Podman** si **es capaz de usar las reglas** que define el UFW.

Para configurar adecuadamente UFW tenemos que modificar el archivo "**/etc/ufw/after.rules**" e introducir las líneas que se pueden visualizar en la figura siguiente (Heddings, 2022 y Heidi, 2019).

**Figura 81 Configuración de Docker para que funcione correctamente con UFW**

```
31
32 # BEGIN UFW AND DOCKER
33 *filter
34 :ufw-user-forward - [0:0]
35 :ufw-docker-logging-deny - [0:0]
36 :DOCKER-USER - [0:0]
37 -A DOCKER-USER -j ufw-user-forward
38
39 -A DOCKER-USER -j RETURN -s 10.0.0.0/8
40 -A DOCKER-USER -j RETURN -s 172.16.0.0/12
41 -A DOCKER-USER -j RETURN -s 192.168.0.0/16
42
43 -A DOCKER-USER -p udp -m udp --sport 53 --dport 1024:65535 -j RETURN
44
45 -A DOCKER-USER -j ufw-docker-logging-deny -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -d
   192.168.0.0/16
46 -A DOCKER-USER -j ufw-docker-logging-deny -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -d
   10.0.0.0/8
47 -A DOCKER-USER -j ufw-docker-logging-deny -p tcp -m tcp --tcp-flags FIN,SYN,RST,ACK SYN -d
   172.16.0.0/12
48 -A DOCKER-USER -j ufw-docker-logging-deny -p udp -m udp --dport 0:32767 -d 192.168.0.0/16
49 -A DOCKER-USER -j ufw-docker-logging-deny -p udp -m udp --dport 0:32767 -d 10.0.0.0/8
50 -A DOCKER-USER -j ufw-docker-logging-deny -p udp -m udp --dport 0:32767 -d 172.16.0.0/12
51
52 -A DOCKER-USER -j RETURN
53
54 -A ufw-docker-logging-deny -m limit --limit 3/min --limit-burst 10 -j LOG --log-prefix "[UFW
   DOCKER BLOCK] "
55 -A ufw-docker-logging-deny -j DROP
56
57 COMMIT
58 # END UFW AND DOCKER
```

Fuente: Elaboración propia

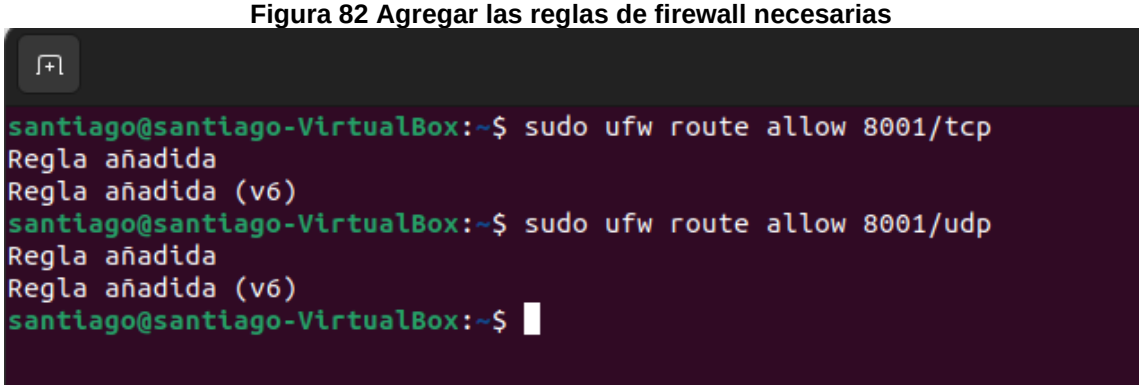
Gracias a esta modificación **Docker no se saltará las reglas** que incluyamos en el firewall (Heddings, 2022 y Heidi, 2019).

Por defecto **UFW deniega todas las conexiones**, eso quiere decir que una vez activado no permitirá ninguna conexión desde el exterior a nuestros servicios alojados en contenedores.

Para permitir solamente que los usuarios interactúen el servicio API incluiremos una regla que permita las conexiones al **puerto 8001** que es donde se encuentra el servicio mencionado.

Con los comandos "**ufw route allow 8001/tcp**" y "**ufw route allow 8001/udp**" conseguimos nuestro objetivo.

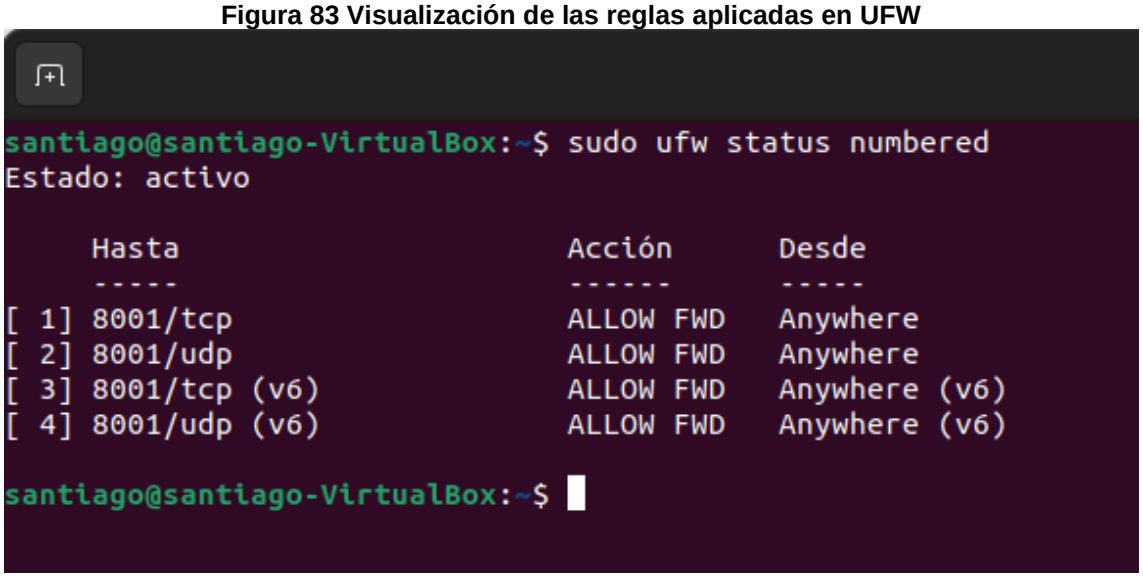
**Figura 82 Agregar las reglas de firewall necesarias**




Fuente: Elaboración propia

Una vez agregadas las dos reglas podemos visualizar un resumen de ufw con el comando "**ufw status numbered**" en el cual podemos visualizar las reglas que hemos agregado (Heddings, 2022 y Heidi, 2019).

**Figura 83 Visualización de las reglas aplicadas en UFW**




Fuente: Elaboración propia

Tal y como podemos visualizar en la figura superior se nos muestra las reglas que hemos creado las cuales permiten la conexión al puerto 8001 desde cualquier dirección origen.

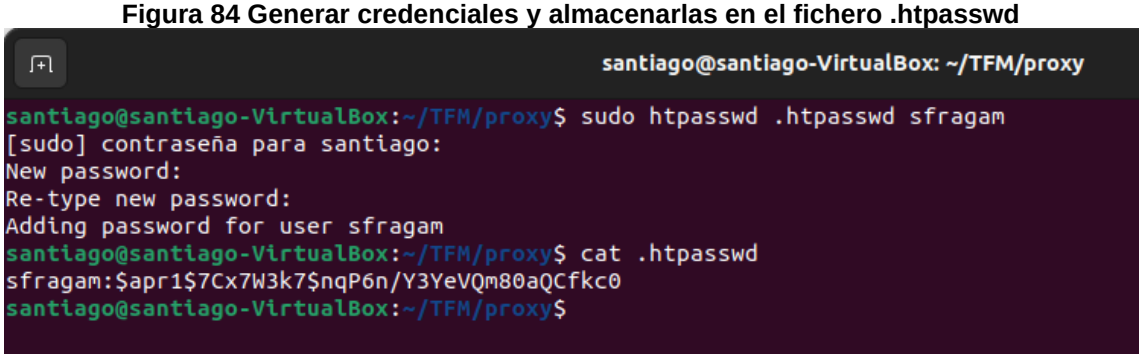
Con esta configuración el sistema de información estaría mejor protegido y solo expondría el puerto que los usuarios necesitan para hacer uso del aplicativo.

*Proxy inverso*

El primer paso es crear un fichero llamado "**.htpasswd**" en el cual almacenaremos las credenciales y posteriormente agregaremos al servidor de Nginx para que lo use.

Tras crear el fichero utilizaremos el comando "**sudo htpasswd .htpasswd sfragam**" para crear las credenciales.

**Figura 84 Generar credenciales y almacenarlas en el fichero .htpasswd**




Fuente: Elaboración propia

Una vez llegados a este punto debemos modificar el archivo llamado "**default.conf**" en la ruta de Nginx "**etc/nginx/conf.d**" para indicar que las peticiones que llegue a la ruta base del servidor proxy sean redirigidas al servicio API.

```
location / {
        auth_basic "Restricted Content";
        auth_basic_user_file /etc/nginx/.htpasswd;
        proxy_pass http://mscv-items:8001/;
}
```

- **auth_basic:** Con esta línea le estamos diciendo a Nginx que aplique la autenticación básica a la ruta "/".
- **auth_basic_user_file:** En esta otra línea indicamos dónde tendremos almacenado el fichero de credenciales dentro del contenedor Nginx.
- **proxy_pass:** Con esta última sentencia especificamos la ruta en la que se encuentra el servicio API.

A continuación, se creará el fichero para configurar la imagen de **Nginx**, el cual tendrá la siguiente estructura.

**Figura 85 Fichero necesario para la creación de la imagen**

```
1 FROM nginx
2 COPY default.conf /etc/nginx/conf.d/
3 COPY .htpasswd /etc/nginx/
4 EXPOSE 85
5 CMD ["nginx", "-g", "daemon off;"]
```

Fuente: Elaboración propia

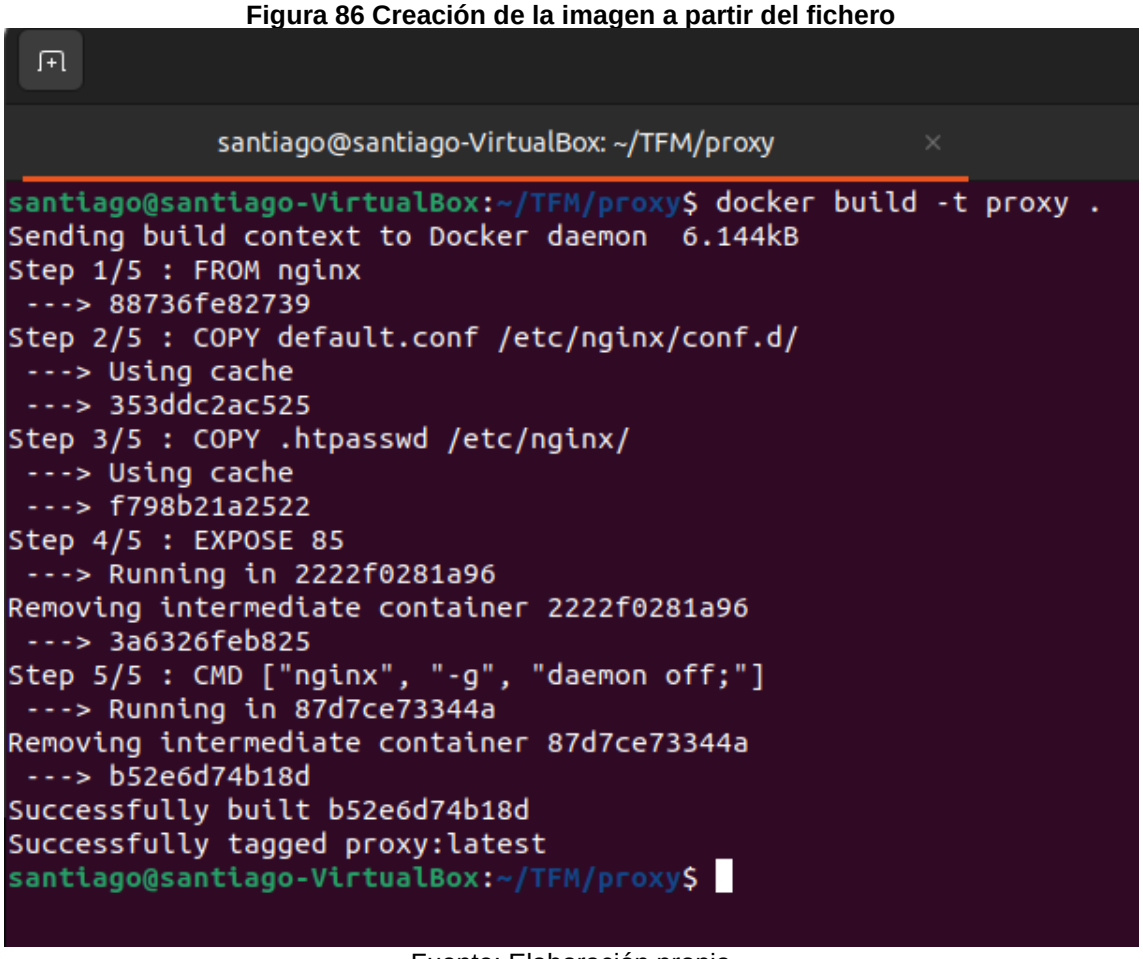
Podemos visualizar en la figura superior que el fichero es muy sencillo y se hace uso de la imagen oficial de Nginx para montar el proxy inverso.

Este fichero lo que hace es **copiar los archivos** correspondientes a la configuración del proxy "**default.conf**" y de las credenciales ".**htpasswd**" al directorio dentro del contenedor de Nginx.

Una vez realizada dichas copias se procede a ejecutar el servicio con "**CMD ["nginx", "-g", "Daemon off;"]**".

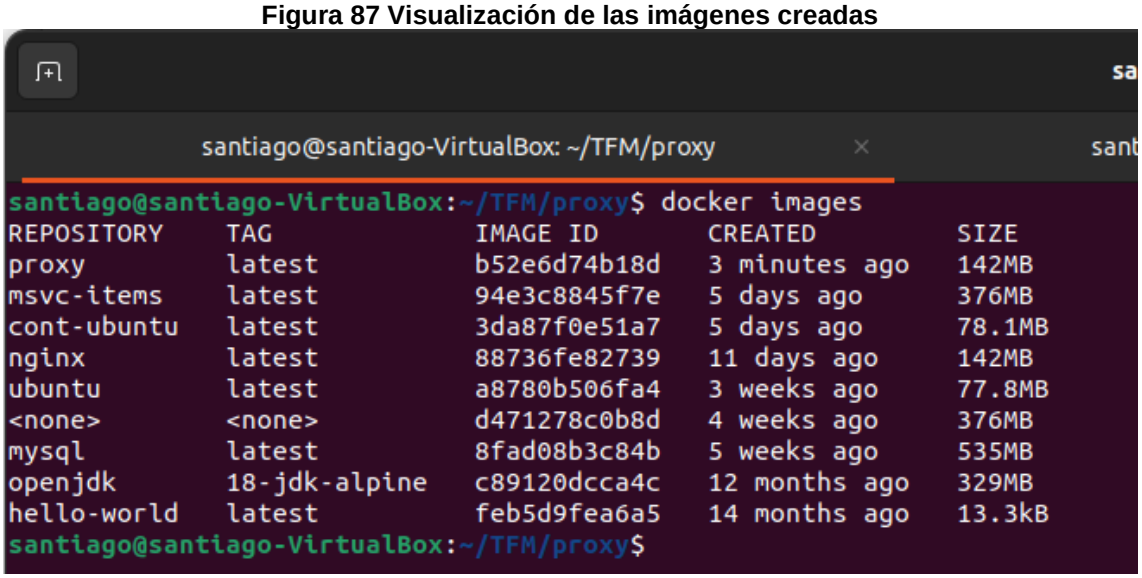
Llegado a este punto el primer paso es construir la imagen partiendo del fichero mostrado anteriormente con el comando "**docker build -t proxy .**".

**Figura 86 Creación de la imagen a partir del fichero**




Fuente: Elaboración propia

**Figura 87 Visualización de las imágenes creadas**




Fuente: Elaboración propia

Posteriormente ejecutaremos un contenedor exponiendo el puerto 85 y haciendo uso de la imagen que se acaba de crear llamada proxy. Para ello ejecutaremos el siguiente comando "**docker run -d -p 85:80 --name proxy --network netitems proxy**".

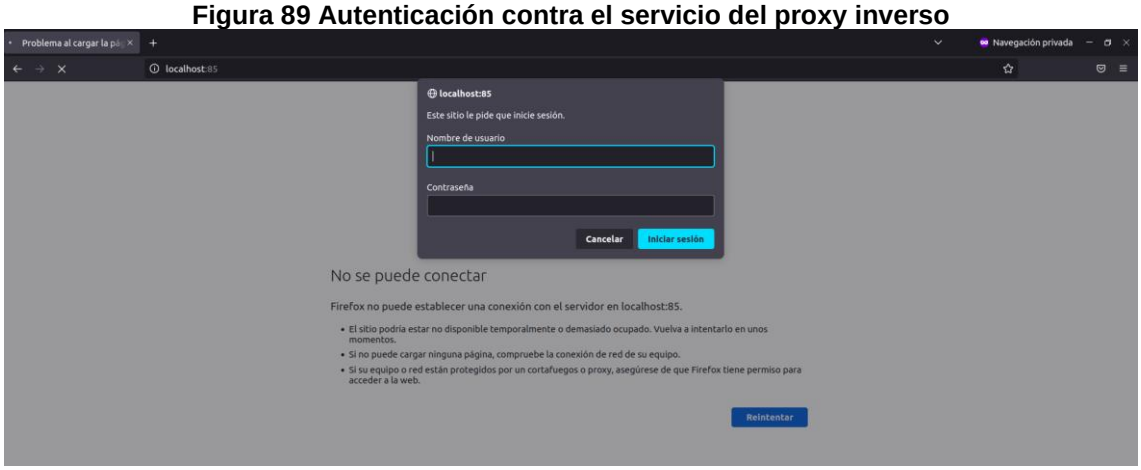
**Figura 88 Ejecución del contenedor con el servicio de proxy inverso Nginx**


Fuente: Elaboración propia

Una vez en ejecución el servicio de proxy inverso, nos dirigiremos al navegador y escribiremos la dirección URL localhost:85 que es donde se encuentra escuchando el proxy.

Tras acceder a dicha URL nos **solicitará las credenciales** que se crearon anteriormente que en este caso en concreto pertenecen al usuario **sfragam.**

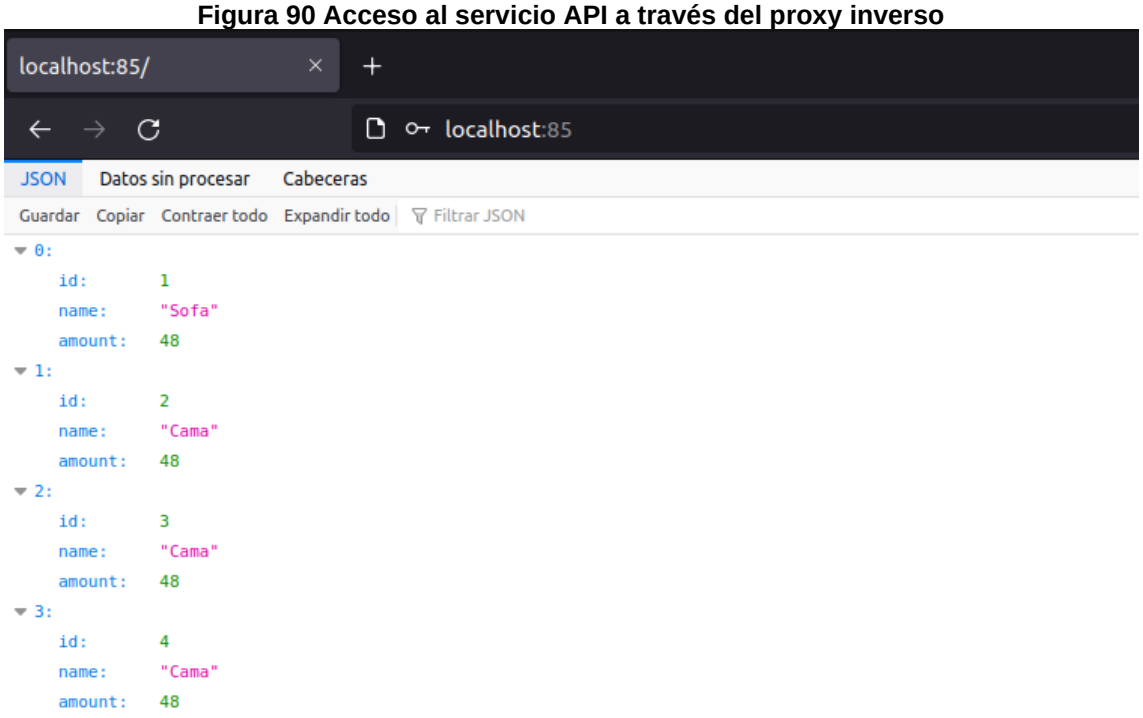
**Figura 89 Autenticación contra el servicio del proxy inverso**




Fuente: Elaboración propia

Tras introducir las credenciales correctas el sistema nos redirige a la dirección en la cual se encuentra el servicio API (**localhost:8001**) y se nos muestra un listado con los ítems que tenemos almacenados en la base de datos.

**Figura 90 Acceso al servicio API a través del proxy inverso**




Fuente: Elaboración propia

Podemos observar que a pesar de tener el servicio API desplegado en el puerto 8001, **la dirección URL** que se nos muestra en el navegador **es la correspondiente al proxy inverso** que se encuentra situado en localhost:85.

Para realizar el proxy inverso con **Podman** es prácticamente igual pero el fichero para generar el servicio cambia un poco.

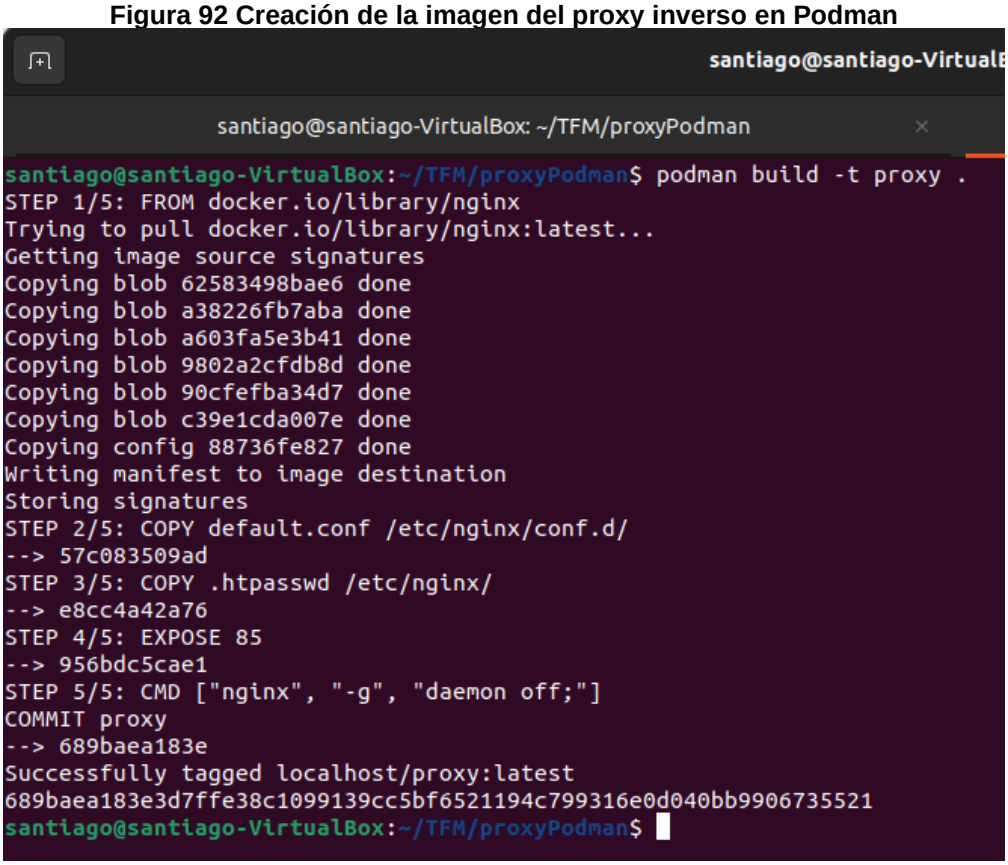
**Figura 91 Fichero para la creación de la imagen en Podman**

```
1 FROM docker.io/library/nginx
2 COPY default.conf /etc/nginx/conf.d/
3 COPY .htpasswd /etc/nginx/
4 EXPOSE 85
5 CMD ["nginx", "-g", "daemon off;"]
```

Fuente: Elaboración propia

El único cambio que debemos agregar es introducir el comando "**docker.io/library/**" delante de la imagen **nginx**.

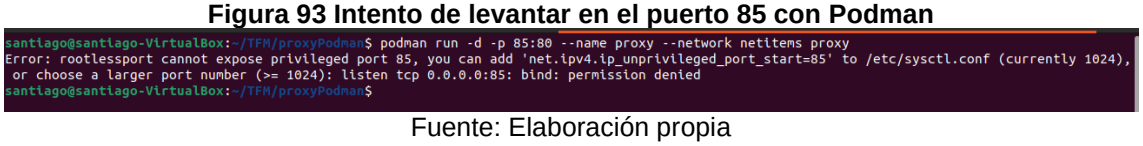
**Figura 92 Creación de la imagen del proxy inverso en Podman**




Fuente: Elaboración propia

A la hora de levantar el contenedor en el caso de Podman nos da un **error** si intentamos **usar el puerto 85** ya que estamos levantando el contenedor con rootless.

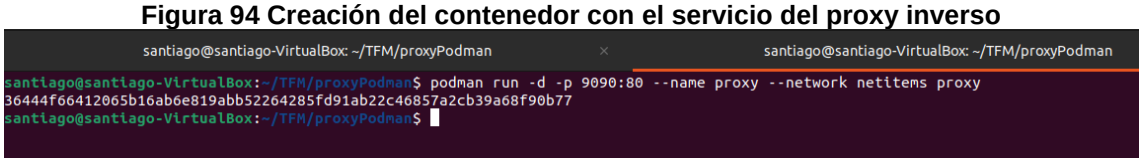
**Figura 93 Intento de levantar en el puerto 85 con Podman**




Fuente: Elaboración propia

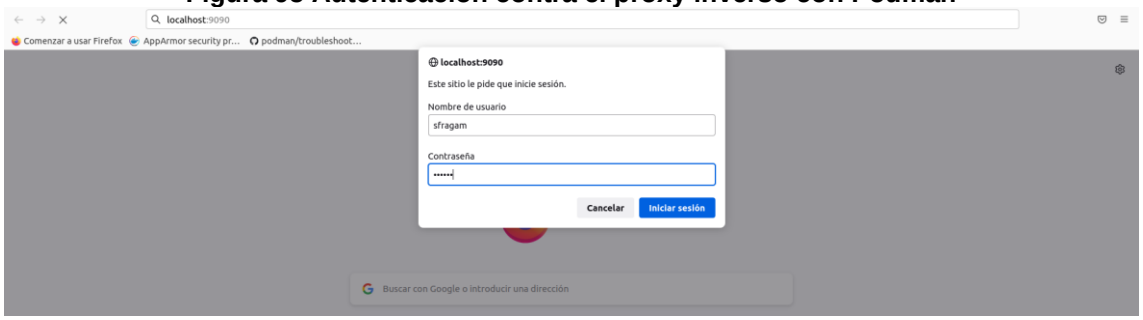
Tal y como podemos observar nos dice que usemos un puerto superior al 1024 por lo que se opta por usar el puerto 9090.

**Figura 94 Creación del contenedor con el servicio del proxy inverso**



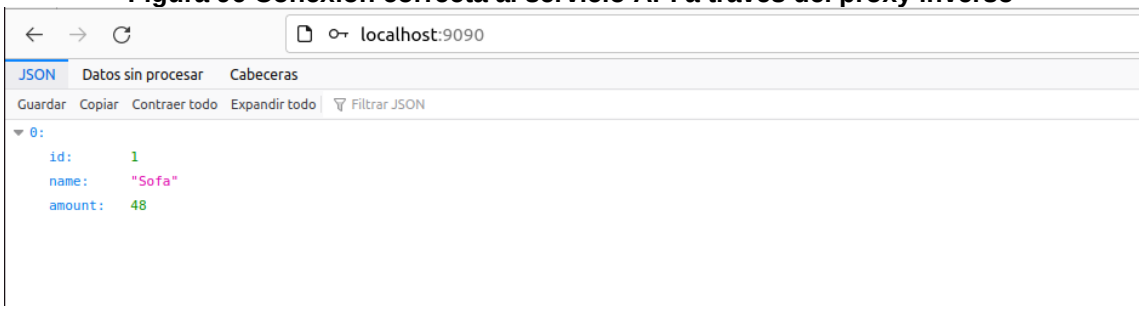

Fuente: Elaboración propia

**Figura 95 Autenticación contra el proxy inverso con Podman**




Fuente: Elaboración propia

**Figura 96 Conexión correcta al servicio API a través del proxy inverso**




Fuente: Elaboración propia

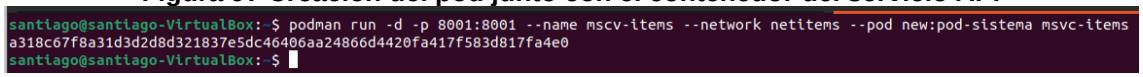
Tal y como podemos observar en las figuras superiores el funcionamiento es exactamente **igual en Podman que en Docker**.

Esto proporciona una **capa extra de seguridad** ya que el usuario nunca accederá al servicio directamente y siempre lo hará a través del proxy inverso.

*Uso de pods con Podman*

Primero empezaremos creando un pod, el cual solo contendrá el contenedor denominado como “Infra” y el contenedor correspondiente al servicio API. Para ello ejecutaremos el siguiente comando “**podman run -d -p 8001:8001 --name msvc-items --network netitems --pod new:pod-sistema msvc-items”.**

**Figura 97 Creación del pod junto con el contenedor del servicio API**




Fuente: Elaboración propia

Con el comando anterior y gracias a la sentencia "**--pod new:pod-sistema**" creamos el pod con el nombre **pod-sistema** y a la vez agregamos el contenedor de **msvc-items** al pod (Bause. 2019; y Goyal, 2021).

A continuación, crearemos el contenedor de mysql dentro del pod, para ello ejecutaremos la siguiente sentencia "**podman run -d --name mysql --network netitems --pod pod-sistema -e MYSQL_ROOT_PASSWORD=sasa -e MYSQL_DATABASE=msvc_items Docker.io/library/mysql**".

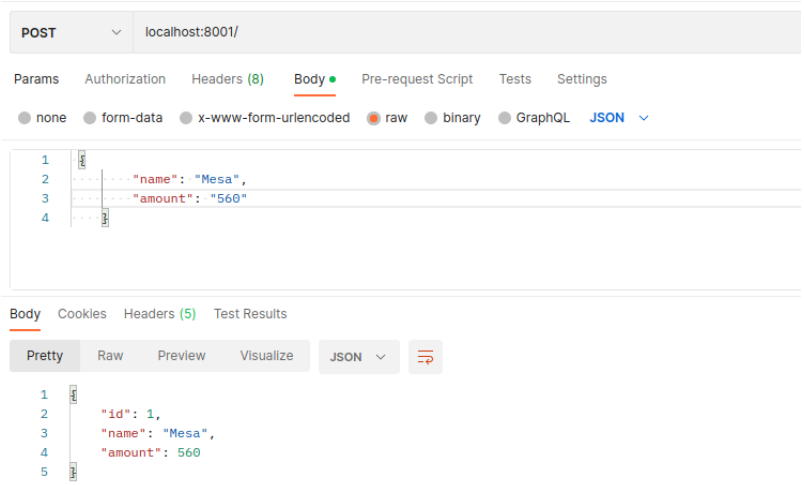
**Figura 98 Inserción del contenedor con MySQL en el pod de Podman**


Fuente: Elaboración propia

Tal y como podemos observar en la figura superior tras agregar el contenedor de MySQL en el pod tenemos 3 contenedores ejecutándose.

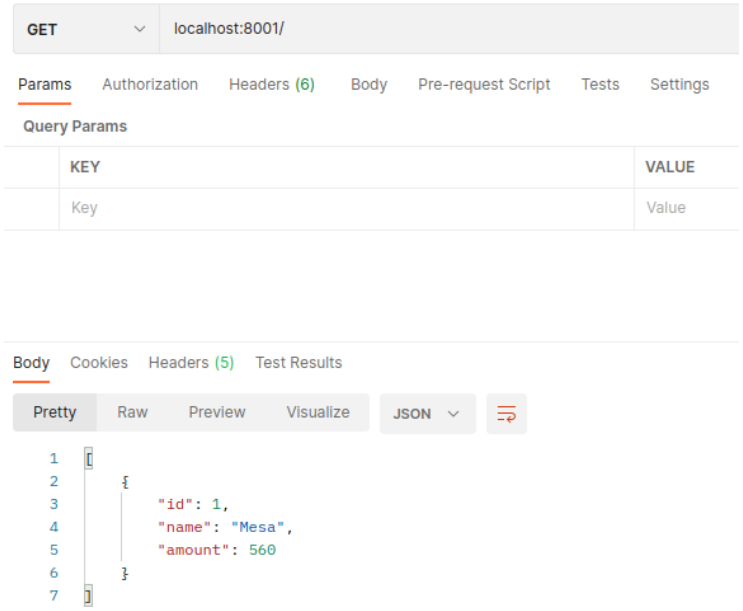
Para ver que todo funciona correctamente nos iremos a Postman y crearemos un item llamado Mesa y luego lo listaremos para ver que se ha creado correctamente.

**Figura 99 Creación del item haciendo uso del servicio API**




Fuente: Elaboración propia

**Figura 100 Listado de los items de base de datos usando el servicio API**




Fuente: Elaboración propia

Tal y como podemos observar el sistema funciona con total normalidad, pero esta vez se está ejecutando dentro de un pod de Podman.

Gracias al uso del pod, el sistema de información basado en contenedores está mejor aislado de los demás elementos que puede contener el sistema host en el que se está ejecutando (Bause, 2019).

Esto se consigue gracias a que los contenedores de un pod están asociados al mismo cgroups y conserva el espacio de nombres de los usuarios.

Por último, se debe destacar que **Podman** usa el mismo concepto de pod que **Kubernetes** lo cual facilita la migración a dicha plataforma de orquestación de contenedores.