Universitat Oberta de Catalunya
uoc.edu

Seguridad en Kubernetes.
Despliegue de contenedores seguros.

Pedro Pedraza Gutiérrez
Máster Universitario en Ciberseguridad y Privacidad
M1.887 - Seguridad empresarial

Nombre Tutor/a de TFM
Manuel Jesús Mendoza Flores

Nombre Profesor/a responsable de la asignatura
Víctor García Font

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Seguridad en Kubernetes, Despliegue de contenedores seguros.*
Nombre del autor:	*Pedro Pedraza Gutiérrez*
Nombre del consultor/a:	*Víctor García Font*
Nombre del PRA:	*Manuel Jesús Mendoza Flores*
Fecha de entrega (mm/aaaa):	*10/2023*
Titulación o programa:	*Máster Universitario en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*M1.887 - Seguridad empresarial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Contenedor, kubernetes, seguridad*

Resumen del Trabajo

El Trabajo de Fin de Máster tiene como fin la realización de despliegues seguros con Kubernetes, utilizando Docker para realizar la contenerización de una aplicación web.

Este trabajo se ha desarrollado en dos partes:

La primera en la que se ha desarrollado una aplicación web, siguiendo una metodología DevSecOps (Aplicamos seguridad en todas las fases del desarrollo DevOps). En la fase de análisis se ha utilizado MITRE ATTACK para documentar los puntos de ataques de un clúster de Kubernetes.

En segunda parte, se ha realizado la implementación de la aplicación creada en la primera parte, y posteriormente se ha aplicado seguridad el despliegue utilizando la información obtenida de MITRE ATTACK.

Abstract

The purpose of the Master's Thesis is to carry out secure deployments with Kubernetes, using Docker to carry out the containerization of a web application.

This work has been developed in two parts:

The first in which a web application has been developed, following a DevSecOps methodology (We apply security in all phases of DevOps development). In the analysis phase, MITRE ATTACK has been used to document the attack points of a Kubernetes cluster.

In the second part, the implementation of the application created in the first part has been carried out, and later the deployment security has been applied using the information obtained from MITRE ATTACK.

Índice

Lista de figuras

1. Introducción

Esta memoria tiene como motivación garantizar la seguridad y la privacidad de los datos que contiene un clúster de Kubernetes. Esta tecnología ofrece mucha flexibilidad en cuanto a su uso, pero tiene como inconveniente que muchas de sus funcionalidades de seguridad no se encuentren activadas, y han de ser los usuarios los que deben aplicar las medidas de seguridad que estimen oportunas. Un usuario poco formado o que desconozca la tecnología, puede dejar muchas puertas abiertas para que un atacante actúe libremente dentro del sistema.

1.1. Contexto y justificación del Trabajo

En los últimos años se está produciendo un cambio en la manera en la cual se diseña y realiza el desarrollo de software, basado en la creación de aplicaciones monolíticas. Estas aplicaciones son diseñadas en una arquitectura de capas, las cuales son utilizadas por un programa. Con el paso del tiempo aumenta el volumen y la complejidad de las aplicaciones, que las hacen poco manejables cuando se han de distribuir.

Como solución a los problemas de escalabilidad, desaprovechamiento de recursos, así como evitar que se produzcan errores humanos con relación al hardware y software, surge una tendencia en la que se intenta cambiar las arquitecturas tradicionales, encaminando los nuevos modelos hacia los sistemas distribuidos.

La aparición de las máquinas virtuales ha permitido grandes mejoras en la distribución y la seguridad de las aplicaciones, dado que disminuyen la superficie de ataque, mejoran la escalabilidad de las aplicaciones, reducen los costes de hardware y permite aislar el contenido entre una máquina virtual y otra.
El uso de máquinas virtuales presenta como punto negativo el tiempo de arranque que necesitan a la hora de escalar, y el desperdicio que se produce en el uso de recursos del sistema, ya que cada máquina virtual tiene que crear su kernel, bibliotecas propias y dependencias.

Frente a los inconvenientes que nos encontramos en las máquinas virtuales, surge un nuevo movimiento que intenta solventar los problemas mencionados mediante el uso de contenedores de software. La contenerización consiste en realizar un paquete con todo lo necesario para que una aplicación se ejecute (librerías, binarios, archivos de configuración y dependencias.) por lo que se consigue que sean menos pesados que las máquinas virtuales. El principal beneficio que se obtiene de realizar despliegues de contenedore es que podemos abstraernos del hardware o software instalado en los hosts en los que se hace el despliegue, dado que contenedor tiene todo lo necesario para funcionar.
Los contenedores no incluyen el sistema operativo en los paquetes lo que les hace más rápidos y fáciles de desplegar, aunque esto nos origina el problema de que un contenedor ha de ser instalado en un host con sistema operativo compatible con el contenedor, dado que el para su funcionamiento utiliza el

sistema operativo del host hospedador. Es una pequeña diferencia con respecto a las máquinas virtuales que, si permitían varios sistemas operativos, al poseer cada una su kernel.

Esta tecnología de contenedores permite la creación de clústeres de contendores.
La tendencia de esta tecnología va encaminada a que aplicaciones complejas se dividan en funcionalidades más sencillas y cada una de ellas pueda instalarse en un contenedor, con lo que se consigue que las actualizaciones sean más sencillas y se puedan realizar mejoras parciales sin tener que desplegarla toda la aplicación. Otro beneficio es permite escalar las funcionalidades más concurridas en lugar de toda la aplicación.

Al igual que las máquinas virtuales son entornos aislados por lo que nos permiten restringir el acceso desde otros contenedores o del exterior.

Este trabajo de fin de grado está motivado, en primer lugar, por conocer una tecnología que actualmente se encuentra en auge con la aparición de los contenedores y más en concreto como establecer la seguridad en kubernetes dado que al ser software libre se encuentra por defecto sin ninguna configuración de seguridad.

Dada la gran cantidad de herramientas y posibilidades que actualmente nos ofrece el mercado, nos sería imposible abarcar todo el ciclo de vida en esta memoria, por lo que nos centraremos en la seguridad que afecta al despliegue de aplicaciones con "**kubernetes",** herramienta con la que realizaremos la gestión de nuestro clúster de contenedores**.**

Kubernetes es un proyecto de software libre que sale a la luz en el año 2014, surge a partir de una herramienta de Google que utilizaba para la gestión de los contenedores llamada Borg [5][1].

Kubernetes (k8s o kube) es un orquestador de contenedores (Docker, CRI-O, ...) intenta resolver el problema de las operaciones manuales con los contendores. Principalmente se encarga de resolver problemas de planificación y automatización de los despliegues, así como realizar la gestión y el escalado de aplicaciones en los contenedores.

1.2. Objetivos del trabajo

Los **objetivos** de este Trabajo de Fin de Máster son los siguientes:

- Conocer la tecnología de contenedores.
- Profundizar en el conocimiento de herramientas de seguridad aplicadas al desarrollo de aplicaciones web.
- Conseguir desplegar un clúster de kubernetes.
- Aplicar las medidas de seguridad necesarias para garantizar la seguridad y privacidad de un clúster de kubernetes.
- Posibilidad de conocer la utilización de kubernetes en la nube.

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

Realizando un pequeño análisis del impacto generado por el uso de la tecnología en este trabajo, claramente observamos que el uso de la virtualización y más concretamente el uso del cloud computing, tienen que ver con la sostenibilidad. El uso de esta tecnología genera los siguientes efectos:

- Reducción de costos. El uso de la virtualización reduce el uso de máquinas y la necesidad de infraestructura hace que las empresas tengan una menor inversión en la compra de equipos, esto también influye en costos de actualizaciones y compra de software.
- Ahorro energético. Asociado a la disminución de equipos, existe un menor consumo de energía.
- Disminución de la cantidad de residuos generados con la enajenación de equipos.
- Disminución del uso de materias primas para la fabricación de equipos.
- Los nuevos centros de datos donde se alojan las plataformas cloud son más eficientes en términos energéticos.

Como principal *factor social* podemos relacionarlo con la privacidad y seguridad de los datos que se almacenan, el uso que se da a la información personal de los usuarios, datos financieros o incluso confidenciales. Debido a lo anterior se deben tomar medidas que eviten el acceso a esos datos. Asociado a adopción de políticas de seguridad podría generar cultura de empresa para el mantenimiento de los empleados, así como la adopción del uso de nuevas tecnologías podría ser un aliciente para la atracción y mantenimiento de talento.

Éticamente podríamos relacionarlo con la cantidad de datos que se almacenan de los usuarios. Los datos que almacenamos deberían limitarse a su mínima expresión o dar opciones al usuario para que elija los datos que desea ceder.

En cuanto a la diversidad, podría asociarse a la posibilidad de trabajo en remoto dando la posibilidad a personas de todo el mundo a su entrada en el entorno empresarial.

1.3.1. Objetivos de desarrollo sostenible (ODS)

En este proyecto nos encontramos con dos características que podríamos aplicar, el uso de la tecnología de contenedores y la ciberseguridad.

El uso de kubernetes, y en general TIC (tecnologías de la información y la comunicación) pueden tener un gran impacto en la consecución de los ODS (Objetivos de desarrollo sostenible) de la agenda 2030 aunque directamente no estén relacionados con los ODS.

ODS_01	Fin de la pobreza	Mediante el uso de tecnología financiera que de acceso a créditos. Podríamos realizar

		despliegues para facilitar el acceso a esas tecnologías.
ODS_02	Hambre cero	Kubernetes mediante su distribución K0s facilita el uso de clúster para recoger información de dispositivos IOT que faciliten la agricultura de precisión. Uso remoto de válvulas de riego.
ODS_03	Salud y bienestar	Uso de tecnologías de kubernetes en biosensores e IOT que ayuden a prevenir las muertes infantiles o la transmisión de enfermedades.
ODS_04	Educación calidad	• Mediante el despliegue de contenedores con plataformas que den servicio de calidad. • Uso de contendores para el despliegue de la red 5G que permita un acceso a la red de mayor calidad y velocidad.
ODS_06	Agua y saneamiento sostenibles	Uso de sensores instalados en clúster de Kubernetes.
ODS_07	Energía limpia y asequible	Mediante el uso de tecnologías de contenedores que son instalados en molinos de viento. Con máquinas barebone y en las cuales se realizan las operaciones necesarias.
ODS_08	Trabajo decente y crecimiento económico	Mediante el uso de tecnología financiera, desplegada en cloud.
ODS_09	Industria, innovación e infraestructura	• Despliegue de 5G mediante el uso de Kubernetes • Uso de tecnología IOT en contenedores
ODS_11	Ciudades y comunidades sostenibles	Mediante el uso de sensores con tecnología de contenedores.
ODS_12	Producción y consumo responsable.	El uso ve virtualización evita que tengan que fabricarse equipos informáticos, produciendo un ahorro en el uso de los recursos naturales
ODS_13	Acción por el clima	La virtualización ha conseguido que haya un numero de máquinas en funcionamiento y por tanto un menor consumo de energía.
ODS_17	Alianzas para lograr los objetivos	El uso de la ciberseguridad da fiabilidad a los usuarios para que todas las políticas puedan usarse. Dado que los delincuentes son de diferentes nacionales obligan a que exista cooperación entre los diferentes gobiernos.

Tabla 1: Objetivos de desarrollo sostenible relacionados con el proyecto.

1.4. Enfoque y método seguido

El presente trabajo contara de dos partes para su desarrollo :

Parte 1:
Se realizará una aproximación muy ligera de la metodología **DevSecOps,** en las fases que ocupan la parte de desarrollo de la metodología (codificación – release), para lo cual se creará una aplicación web contenerizada , la cual, mediante un proceso de integración continua se almacenará en un registro de contenedores seguro.

Parte 2:
Se realizará despliegue en un clúster de kubernetes haciendo un gran énfasis en la aplicación de medidas de seguridad al clúster sobre el que se ha realizado el despliegue.
Las medidas de seguridad implementadas al clúster serán extraídas del análisis a la matriz de ataques MITRE.

1.5. Planificación del Trabajo

Adjuntamos la planificación inicial para realizar las entregas de este trabajo obtenidas de la plataforma como una estimación inicial para realizar las entregas.

Imagen 1. Diagrama de Gantt de entregas

La definición de las tareas a bajo nivel podría desglosarse en las siguientes:

1.5.1. Estudio de preliminar de Kubernetes dado el desconocimiento de la tecnología, y por lo tanto de los recursos necesarios para realizar el despliegue.

1.5.2. Estudio de la matriz de MITRE ATT$CK.

1.5.3. Desarrollo de aplicación a desplegar, creación de los contenedores y almacenamiento de la imagen del contenedor en un repositorio.

1.5.4. Realizar el despliegue en un clúster de kubernetes.

1.5.5. Securización del clúster.

1.5.6. Test para comprobar que las medidas han sido efectivas.

1.5.7. Existirá una tarea transversal que se extenderá a lo largo de las entregas y que consistirá en cumplimentar esta memoria con todos los pasos realizados.

1.6. Breve sumario de productos obtenidos

Lo productos que obtendremos al final serán los siguientes:

- Documentación plasmada en esta memoria.
- Una presentación de la información recopilada.
- Generación de una imagen no entregable, que contendrá nuestro clúster de kubernetes y las pruebas realizadas en él.

1.7. Breve descripción de los otros capítulos de la memoria

Capítulo 2. En este capítulo ser recoge la metodología a utilizar un análisis previo del riesgo y costes que comporta la realización del proyecto.

*Capítulo 3***.** Análisis de las herramientas utilizadas

*Capítulo 4***.** Explicación del marco de trabajo MITRE ATTACK centrándonos en su matriz para Kubernetes

*Capítulo 5***.** Mediante una metodología de desarrollo Devops se realiza un deployment completo en un clúster de kubernetes.

*Capítulo 6***.** Introducción a kubernetes y ejemplo de realización de un deployment.

*Capítulo 7***.** Aplicación de información extraída de la matriz de ataques de MITRE para conseguir un clúster seguro de Kubernetes.

1.8. Revisión de estado del arte

Una vez realizada una pequeña investigación para ver cuál es el estado de las tecnologías que pueden utilizadas en este trabajo nos encontramos kubernetes.

La tecnología de los contendores nos permite la ejecución de aplicaciones tanto en ordenadores físicos como nubes públicas y privadas. Se considera una tecnología sólida y en expansión. Como ya hemos comentado en puntos anteriores su funcionamiento consiste en empaquetar todo lo necesario para que la aplicación funcione. El contenedor utiliza el sistema operativo del host donde es instalado.

Kubernetes es el orquestador de contenedores más utilizado dado que es un proyecto de código abierto, gratuito y sobre el que se basan el resto de los proyectos de orquestadores.

Como distribuciones de pago de kubernetes nos encontramos con Openshift del fabricante Red Hat (tiene políticas más restrictivas, y algunas funcionalidades más avanzadas que kubernetes.)
En cuanto a servicios gestionados en la nube nos encontramos con Amazon web services (EKS), Azure (AKS) y Google cloud (GKE), estos son los más conocidos , pero existen muchos más.

2. Materiales y métodos

Este punto recoge la metodología que se aplicará en el proyecto, un análisis de dos factores que pueden alterar el desarrollo del proyecto, y los medios materiales necesarios y sus costes.

2.1. Metodología DevSecOps

La metodología que utilizaremos para la realización este proyecto consistirá en el uso de DevSecOps (Desarrollo + Seguridad + Operaciones, traducido del inglés Development, Security and Operations).

Esta metodología se caracteriza por la integración de la seguridad en todas las fases del desarrollo del software (desde el diseño, hasta la entrega final de código), incorpora puntos de control y pruebas de seguridad. Actuando de este modo lo que se intenta es abordar los problemas cuando surgen, sin importar en que etapa del desarrollo en la que nos encontremos.

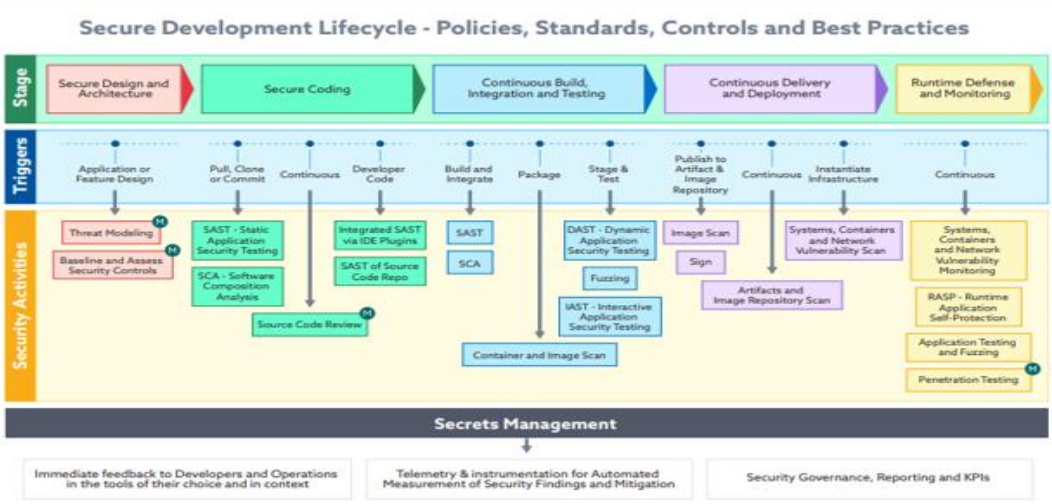

Imagen 2. Diagrama ciclo de vida desarrollo seguro

En DevSecOps nos encontramos con las siguientes etapas en su ciclo de vida.

3.1. Planificación (Análisis + Diseño):

En la etapa de planificación se han de incluir los requisitos de seguridad necesarios para cumplir con las políticas y normativas de seguridad establecidas (ejemplo: protección de datos GPRD).
Se validan los requisitos de seguridad en el análisis funcional.

3.2. Codificación :

Se debe fomentar las prácticas de programación segura y las buenas prácticas.
Incorporar herramientas para pruebas de seguridad e inclusión de auditorías de código y auditorias de tipo estático (SAST)

Una auditoria *SAST*(Static Application Security Testing) consiste en la validar los requisitos de seguridad de una aplicación a través de la revisión de su código y arquitectura. Esta auditoria nos facilitara información acerca de las vulnerabilidades y uso de malas prácticas de codificación o diseño inseguro .

Evaluación de dependencias de código.

3.3. Compilación (Build)**:**

Una vez que el sistema se encuentra en ejecución introduciremos el uso de herramientas de análisis dinámico se buscarán vulnerabilidades como inyección SQL, autenticación de usuarios.

3.4. Pruebas:

Se realizan nuevas pruebas de código debido a la realización desarrollos correctivos, auditorias de tipo dinámico.
Si las pruebas son satisfactorias obtendremos el producto candidato a instalar en producción.

3.5. Mantenimiento:

En esta fase se realizará la monitorización continua de la aplicación con el fin de garantizar que no existen vulnerabilidades en ella.

3.6. Implementación:

Con los datos obtenidos en las etapas anteriores, se intentará comprobar que no existen problemas como:

3.6.1. La imagen es insegura.
3.6.2. Problemas de privacidad o acceso a ella.
3.6.3. Accesos privilegiados indebidos.

Las implementaciones deben realizarse con limitaciones de recursos si estos no son necesarios para el uso.

3.7. Operaciones y monitorización:

En esta fase se realiza una monitorización continua en el entorno de producción de la aplicación con el fin de contrastar que las medidas de seguridad adoptadas en fases anteriores funcionan correctamente.
Se intentará automatizar la mayor parte de los procesos ya que la correcta gestión de las amenazas evitará mayores daños.

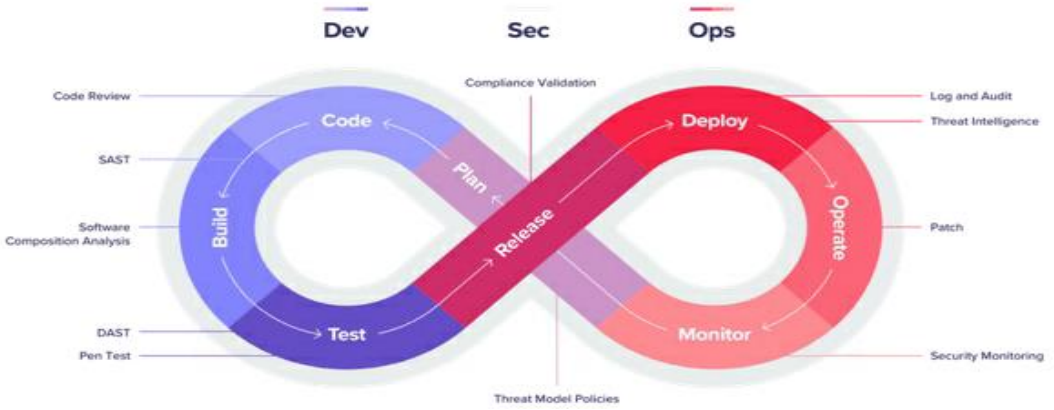

Imagen 3. Imagen fases DevSecOps

2.2. Recursos

Como componentes hardware, necesitaremos simplemente un monitor y un ordenador personal cumpla los requisitos mínimos para la instalación de las aplicaciones requeridas, dado la simplicidad de las aplicaciones que se van a desplegar en los laboratorios realizados.

La configuración del ordenador personal utilizado es la siguiente:

- Fabricante: HP
- Modelo: Pavilion Desktop - 510-p108ns
- Procesador: AMD A12-9800 RADEON R7@ 3.80GHz
- Memoria RAM: 16 GB
- Disco Duro: 1TB
- Disco SSD: 700 GB
- Sistema Operativo: Windows 10

2.3. Análisis de riesgos

El análisis de riesgos consiste en la evaluación de amenazas presentes o futuras que afectan negativamente en el desarrollo del proyecto. Una correcta identificación y evaluación de los riesgos que afectan a nuestro proyecto, nos permitirán tener un plan de acción para evitar que estos ocurran o que sus efectos puedan mitigarse en su mayoría. Estas actuaciones permiten que el proyecto avance de manera satisfactoria y no se produzcan retrasos.

Este proceso de gestión de riesgos se realiza en cuatro etapas:

2.3.1. Identificar. En esta tapa se intenta averiguar cuáles son los elementos que pueden afectar al correcto desarrollo del proyecto. Los riesgos identificados se priorizan a partir de la estimación de la probabilidad de que ocurran y su impacto. Nos ayudaremos de una matriz de riesgos para ayudarnos a realizar dicha estimación.

2.3.2. Análisis y calificación. En función de las posibilidades de que ocurra o de la gravedad de los efectos que ocurra nos encontraremos con riesgos:

2.3.2.1. Bajo. Pocas posibilidades de que ocurra
2.3.2.2. Moderado. Aumento en coste, tiempo o rendimiento del proyecto.
2.3.2.3. Alto. Efectos significativos en coste, tiempo o rendimiento. Se presta especial atención a estos riesgos.

2.3.2.4. Evaluación. En esta etapa se intenta conseguir un factor de riesgo evaluando la probabilidad y el impacto correspondientes a la calificación realizada en la etapa anterior

2.3.2.5. Tratamiento y mitigación. Se busca una respuesta al riesgo identificado.

Impacto / Probabilidad	Bajo	Medio	Alto
Baja	Bajo	Bajo	Moderado
Media	Bajo	Moderado	Alto
Alta	Moderado	Alto	Alto

A continuación, procedemos a la identificación de los riesgos identificados en nuestro proyecto.

Riesgo: 01	Mala definición en el alcance del proyecto.
Descripción	Se han introducido demasiadas tareas, y no es viable
Impacto:	Medio
Probabilidad:	Media
Tratamiento:	Seguimiento del plazo de entrega de las tareas, para ver que van en tiempo.
Respuesta:	Evaluación del proyecto nuevamente, eliminar tareas. En nuestro caso se aumentará el número de horas estimadas.

Riesgo: 02	Formación insuficiente en las tecnologías.
Descripción	El desconocimiento de la tecnología puede influir negativamente en el desarrollo de las tareas, y provocar retrasos.
Impacto:	Media
Probabilidad:	Alto
Tratamiento:	Realización de cursos, lectura de documentos técnicos sobre las tecnologías a utilizar.
Respuesta:	Aumento de tiempo en formación.

Riesgo: 03	Problemas con los recursos utilizados.
Descripción	Fallos en el funcionamiento del hardware utilizado
Impacto:	Medio
Probabilidad:	Baja
Tratamiento:	Tener servicio de mantenimiento durante el periodo de realización del proyecto.
Respuesta:	Sustitución o reparación del equipo.

Riesgo: 04	Problemas con la compatibilidad del software utilizado.
Descripción	Existen problemas de compatibilidad entre las diferentes versiones de las aplicaciones utilizadas que evitan que funcionen correctamente
Impacto:	Alto
Probabilidad:	Media
Tratamiento:	Revisión de la documentación.
Respuesta:	Reinstalación de componentes compatibles o software sustitutivo compatible

Riesgo: 05	Problemas con permisos en plataformas de terceros.
Descripción	Existen restricciones de uso con las versiones educativas del software.
Impacto:	Alto
Probabilidad:	Bajo
Tratamiento:	Revisión de las políticas de terceros.
Respuesta:	Cambio a una versión de pago con toda la funcionalidad.

Riesgo: 06	Perdida de información en documentación.
Descripción	No existe copia de seguridad, y se corrompe el fichero que la contiene
Impacto:	Alto
Probabilidad:	Bajo
Tratamiento:	Se deberá rehacer nuevamente la documentación
Respuesta:	Realización de copias de seguridad en la maquina y en la nube.

2.4. Seguimiento de riesgos.

Una vez que hemos identificado los riesgos, según nuestra matriz de riesgos debemos realizar un especial seguimiento a los que tienen un coeficiente alto.

Riesgo 02.
Una vez que se empiece la implementación del proyecto, se irán identificando las nuevas tecnologías, y se han de realizar búsquedas en manuales o en internet que nos ayuden a una correcta implementación de estas.
Se identificarán aquí en que recursos se ha necesitado más formación.

Riesgo 04.
Se ha de prestar especial atención a la hora de realizar las instalaciones de las versiones de software utilizado.

2.5. Análisis de costes inicial

Dado que la utilización de software ya se ha comentado en puntos anteriores al utilizar versiones educativas o software libre, el análisis de costes vendría derivado de salario del Analista-programador, al no necesitar el proyecto personal de sistemas, y al costo de adquisición de la maquina utilizada para trabajar.

Realizando una búsqueda por internet para obtener el salario hemos encontrado los siguiente en la web https://es.talent.com/salary?job=DevOps

Imagen 6. Imagen salario Ingeniero DevOps

En dicha web se nos hacilita el coste por hora, el cual corresponde a 23,08 €/h.

El otro elemento a valorar el la maquina y el monitor utilizado con un valor estimado de 650€.

Por lo que la estimacion incial de costes es 23,08€/hora * 75 horas estimadas + 650.

Coste inicial estimado = 2.381€

3. Herramientas

La flexibilidad de Kubernetes, permite que pueda ser usado en diferentes sistemas operativos y, por lo tanto, el despliegue de aplicaciones en cualquier lenguaje, siempre que las aplicaciones se puedan desplegarse utilizando el *sistema operativo del host* donde se instala Kubernetes (un host anfitrión con sistema operativo Windows, solo podrá alojar contenedores de Windows.

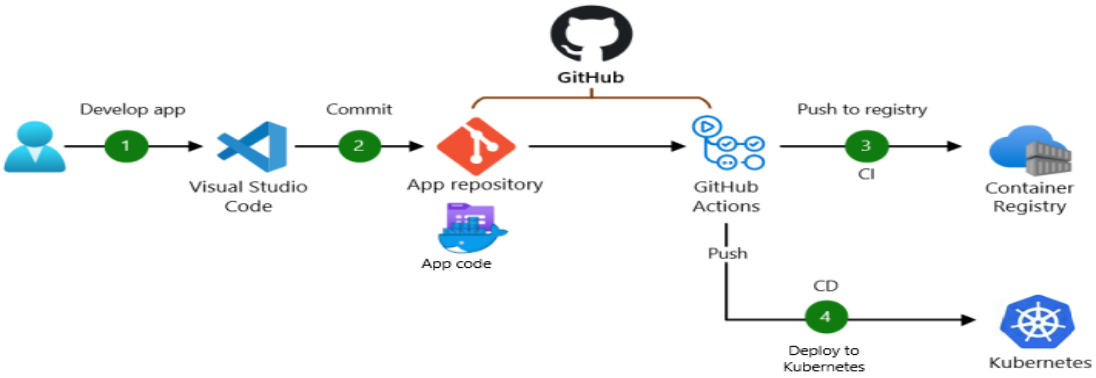

Imagen 5. Diagrama de herramientas

Para el desarrollo de este **Trabajo** se utilizarán herramientas de software libre o bajo licencia educativa.

***Añadimos una excepción, debido a que, durante el desarrollo por requerimientos de una mejora para una política de seguridad, se ha debido realizar la utilización de una plataforma de almacenamiento en Cloud (Google Cloud Storage) de pago. Para el uso de esta plataforma se ha utilizado un "Free Trial" que ofrece un crédito para iniciación y formación en dicha plataforma de 281,39€ de los cuales no se ha utilizado nada en la realización de la prueba.**

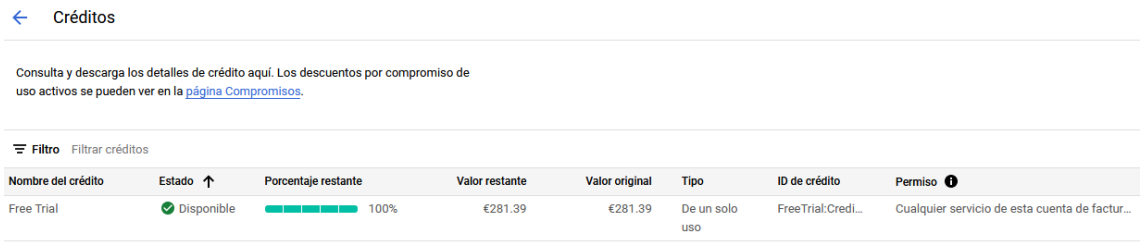

Imagen 6. Consumo servicios Cloud Store

Las herramientas utilizadas serán las siguientes:

3.1. *VMware Workstation*

VMware Workstation es un hipervisor de tipo 2 o alojado, funciona creando una capa software encima del sistema operativo o una aplicación. Dentro de la capa creada permite que instalemos otro sistema operativo virtualizado que aprovecha las características de hardware y del sistema operativo del host hospedador.

Utilizaremos el entorno de virtualización para instalar un entorno de trabajo en Windows 10 donde instalaremos el resto de las herramientas para crear un entorno en el que poder desarrollar el entorno de trabajo.

3.2. *Visual Studio Code.*

Visual Studio Code. Editor ligero de código fuente que puede ejecutarse en diferentes sistemas operativos.

3.3. *GitHub y Git*

Git es un sistema de control de versiones de código abierto. En nuestro proyecto lo utilizaremos como repositorio de código y control de cambios/versiones.

Github es un servicio alojado en la nube que entre muchas cosas nos permite la gestión de los proyectos almacenados en Git.

Aparte de como repositorio, también utilizaremos **Github Container Registry** para almacenar de forma segura las imágenes de Docker.

3.4. *Docker*

Docker. Proyecto de código abierto, que nos permite empaquetar aplicaciones y sus dependencias dentro de contenedores ligeros. Una vez que se ejecuta la aplicación, el contenedor utiliza los recursos del sistema anfitrión.

Docker aparte de almacenar imágenes en la nube en **DockerHub** nos permite el despliegue y utilización en el entorno local de los contenedores.

3.5. *Snyk*

Snyk. Es una herramienta de análisis estático de aplicaciones (SAST, Static Applicacion Security Testing), que nos ayudan mediante análisis de código fuente o sobre el código compilado, nos indican errores y vulnerabilidades comunes conocidas.

Snyk se encuentre instalado en Docker por lo que podremos realizar análisis de vulnerabilidades antes de que nuestras imágenes suban al registro de contenedores.

3.6. *Owasp Zap*

Owasp Zap. Es una herramienta de análisis dinámico de aplicaciones. (DAST, Dynamic Application Security Testing). Este tipo de análisis consiste en ejecutar una aplicación y analizar el comportamiento de esta en funcionamiento.

3.7. *Minikube*

Minikube. Es una distribución minimizada de Kubernetes y que nos permitirá crear un clúster completamente funcional en el entorno local.

3.8. Velero

Velero. Es una herramienta opensource que entre sus funciones incluye lo siguiente:

- Copias totales o parciales (a través del espacio de nombres o selectores de Etiquetas) del clúster.
- Programar copias de seguridad
- Dispone de unos conectores que nos permiten realizar operaciones personalizadas, antes y después de realizar la copia de seguridad.

3.9. Connaisseur

Connaisseur. Es una herramienta diseñada bajo los valores de la seguridad, usabilidad y compatibilidad. Es extensible y permite el uso de la firma digital generada por las herramientas, Notary y Singstore.

4. MITRE ATT$CK framework [11.6][11.7]

Según lo define MITRE : ”ATT&CK® es una base de conocimiento accesible a nivel mundial de tácticas y técnicas del adversario basadas en observaciones del mundo real. ”[1]

Para llegar a ser una base de conocimiento, ATTACK a través de las observaciones realizadas sobre los datos obtenidos en los diferentes ataques analizados, extrae información para intentar deducir lo que pretende un atacante cuando se realiza su ataque. El conocimiento extraído se utilizará como base parar prevenir futuros ataque.

Este framework *es* abierto y permite que la comunidad de usuarios proponga nuevas mejoras o facilite datos sobre nuevos ataques.

[1] MITRE ATT&CK https://attack.mitre.org/

4.1. La matriz.

La *matriz* se encuentra organizada en forma tabular, compuesta en las columnas por las tácticas (métodos utilizados) y en las filas nos encontramos con las técnicas y subtécnicas (estrategia o acciones seguidas, para conseguir los objetivos definido en la táctica).

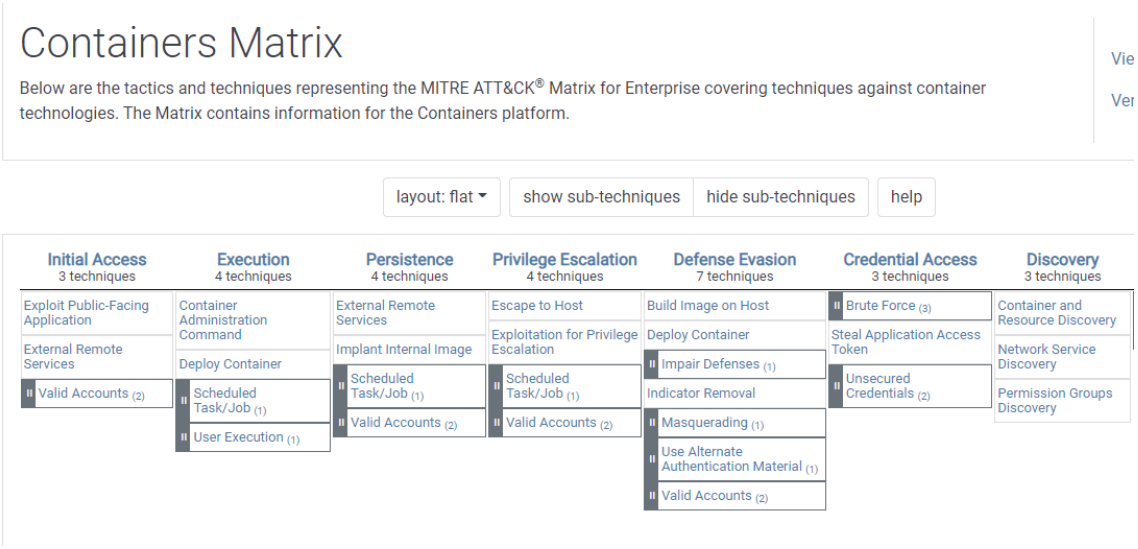

Imagen 7. Disposición de elementos en la matriz MITRE

En matriz de acceso web, todos los elementos contienen un hipervínculo, que nos dirigen a los datos particulares (detalle) del elemento pulsado.

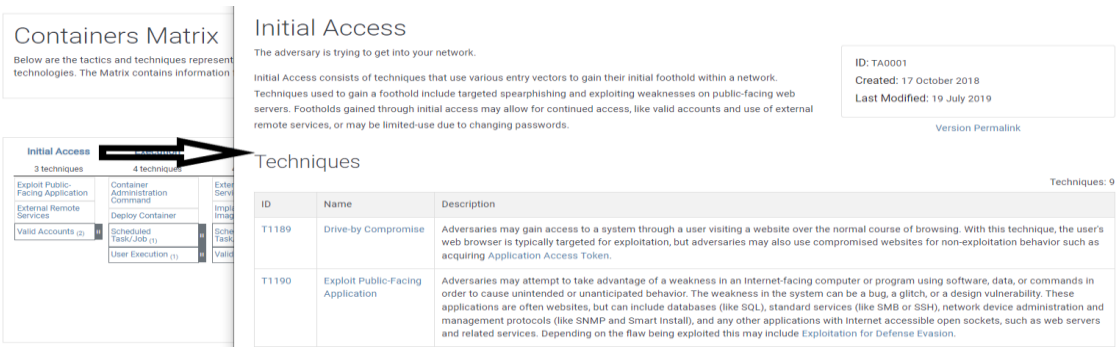

Imagen 8. Técnicas MITRE

Imagen Enlace de táctica con su ficha de detalle.

La matriz nos permite filtrar datos, para diferentes sistemas operativos y plataformas (Contenedores, macOS, Windows, Linux, Azure AD, PRE, Redes, Office 365, Google Workspace, SaaS, IaaS.)

4.2. Las técnicas y subtécnicas

Son las estrategias o acciones seguidas, para conseguir los objetivos definidos en la táctica.

En su ficha de detalle recogen:

- Una pequeña descripción sobre las acciones que se realizan para conseguir el objetivo
- En caso de las técnicas, incluye las subtécnicas que utiliza, si las tiene.
- Como se mitigan sus efectos
- Los procedimientos que debemos realizar para detectarlas.

4.3. Relación entre los diferentes componentes

Cuando se realiza un ataque por parte de un grupo atacante, se realiza mediante técnicas o software que implementa dichas técnicas. Con los datos capturados de un ataque detectado se realiza un análisis, y con la información obtenida se podrá clasificar la técnica dentro de una táctica.
La mitigación se encarga de prevenir las tácticas.

Imagen 9. Relación elementos Attack

4.4. Elección de matriz

Para el desarrollo de este trabajo, pese a que Mitre tiene implementada una matriz para contenedores, creemos que la matriz de Kubernetes creada por Microsoft, nos facilitará información más específica, sobre los ítems a los que deberíamos prestar más atención en su seguridad.

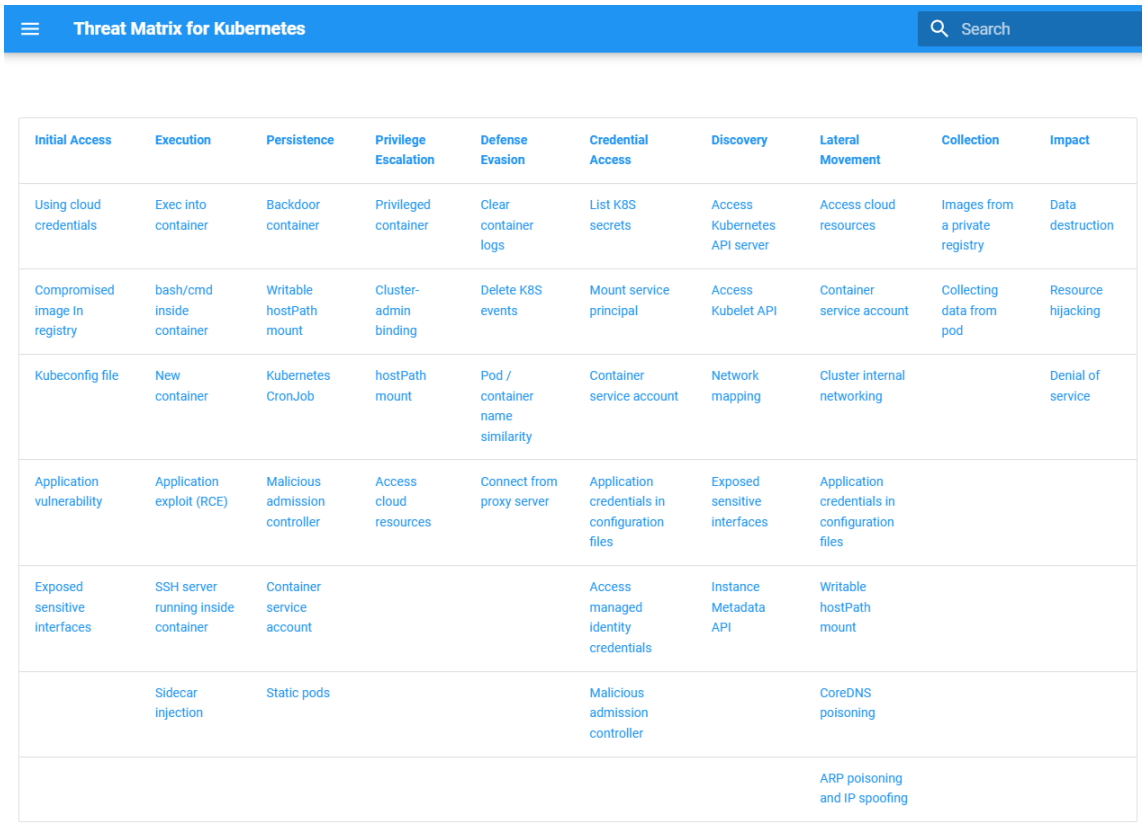

Imagen 10. Matriz de ataques de Microsoft

4.5. Tácticas, técnicas y procedimientos (TTPs)

Para realizar el análisis a los datos contenidos en matriz, procederemos a analizar todas las técnicas que corresponden a una táctica y de izquierda derecha de la matriz.

4.5.1. Acceso inicial

Esta primera táctica que nos encontramos agrupa las técnicas que permiten a los atacantes el acceso al clúster (de manera directa o a través de otros recursos maliciosos) para intentar comprometer los componentes que forman el clúster.

4.5.1.1. Uso de credenciales en la nube

La obtención por parte de los atacantes de las credenciales de acceso posibilita que pueda comprometerse el clúster obteniendo acceso a la capa de administración.

- Mitigación:
    - Uso de autenticación multifactor.
    - Permitir acceso solo a direcciones IP de confianza.
    - Política RBAC limitando privilegios y acciones para acceder al kubeconfig.

4.5.1.2. Imágenes comprometidas en el registro

Las imágenes de contenedores obtenidas de fuentes no fiables pueden ser imágenes comprometidas que faciliten el acceso de un atacante a la administración del clúster.

- Mitigación:
    - Las imágenes han de proceder de registros confiables.
    - Protección del repositorio de imágenes o crear un registro de contenedores seguro.

4.5.1.3. Fichero de Kubeconfig

El archivo kubeconfig, es utilizado por kubectl para organizar la información del clúster, contiene las credenciales del clúster y su dirección. Con la información un atacante podría acceder al clúster.

Se pude obtener a través de comandos (“az aks getcredential” en AKS o para GKE con “gcloud container clusters get-credentials”)

- Mitigación:
    - Utilización en el api-server de una autenticación por parte de terceros que evite el uso de certificados u otro tipo de acceso por contraseña.
    - Solución de detección de intrusos en la red y uso de firewall.
    - Política RBAC limitando privilegios y acciones para acceder al kubeconfig.

4.5.1.4. Vulnerabilidades en aplicaciones (CVE-2014-3120)

La seguridad en las aplicaciones que se crean es primordial para el mantenimiento seguro del clúster. Aplicaciones con vulnerabilidades podrían ocasionar el acceso de un atacante al clúster, ser utilizadas para un ataque de denegación de servicio o dar acceso a otras aplicaciones del clúster.

- Mitigación:
    - Para mitigar esta técnica, podemos ir desde a la implantación de pruebas, hasta la utilización de sistemas de

análisis estático o dinámico que nos dé cierta seguridad que los despliegues de esos contenedores no producirán problemas.
- unido a estos despliegues el control de privilegios de cuentas de servicio y usuarios que deberían ser los mínimos para que puedan realizar las funciones asignadas

4.5.1.5. Interfaces sensibles expuestas

Las interfaces sensibles dentro del clúster de Kubernetes son aquellas que han sido diseñadas para ser accesibles por los usuarios, deberían poseer algún tipo de autenticación para acceder a ellas.

Todas las interfaces que realicen tareas de vital importancia deberían encontrarse separada del resto de interfaces.

- Mitigación:

    - Limitar acceso a las interfaces a través de autenticación.
    - Políticas de red que limiten el acceso a las interfaces sensibles.

4.5.2. Ejecución

La táctica, nos encontramos todas aquellas técnicas en las que el atacante ejecuta código en nuestro clúster.

4.5.2.1. Exec en contenedores

El atacante accede al clúster y si dispone de permisos de ejecución, puede ejecutar código malicioso mediante el comando exec en Kubectl.

- Mitigación:

    - Se debe establecer una política de mínimo privilegio, evitando que los usuarios ejecuten Exec
    - Restringir los comandos de ejecución en el Admision controller que se encuentra en el API-server

4.5.2.2. Ejecución bash / cmd en contenedores

En esta técnica el atacante utiliza los permisos que posee para ejecutar scripts bash con código malicioso y comprometer el clúster.

- Mitigación:

    - Eliminar los terminales que se ejecutan dentro de las imágenes de los contenedores.
    - Mediante los módulos de seguridad de Linux (LSM) se deben restringir las capacidades en tiempo de ejecución.

4.5.2.3. Creación de nuevos contenedores

El atacante dispone de permisos para crear pods o controladores (Replicaset \ DaemonSet \ Deployment), utilizando estos objetos para crear un contendor nuevo y en ejecutar su código malicioso en él.

- Mitigación:

    - Restringir que los usuarios y cuentas de servicio tengan permisos para crear nuevos pods o controladores.

4.5.2.4. Exploits en aplicación de escritorio remoto (CVE-2014-3120)

La técnica consiste en la ejecución de código aprovechando que una aplicación implementada en el clúster permite ejecución de código remoto. El atacante podría además realizar peticiones al servidor API utilizando las credenciales de la cuenta de servicio, si esta se encuentra montada.

- Mitigación:

    - Creación de cuentas de servicio única para cada aplicación, y que los tokens de acceso solo se instalen en caso de ser necesario.
    - Una política de restricción de ejecución prevendría la ejecución de código.
    - Bloqueo de imágenes de aplicaciones con vulnerabilidades

4.5.2.5. Ejecución de SSH dentro del contenedor

Esta técnica utiliza el servidor SSH instalado en el contenedor, para realizar ejecución de código. El atraque se produce porque el atacante ha

obtenido credenciales mediante el uso de fuerza bruta o por otro tipo de ataques.

- Mitigación:

    - Limitar el uso de SSH en contenedores. Y limitar los accesos del resto de nodos al contenedor que lo usa. Se ha de auditar todo el código creado desde SSH.

4.5.2.6. Inyección Sidecar

Un contenedor con sidecar es aquel que en el despliegue lo hacen con un contenedor secundario que sirve de apoyo para el funcionamiento del contenedor principal. Con el fin de ocultarse, los atacantes pueden inyectar un contenedor con sidecar en un pod legítimo, evitando crear un pod nuevo.

- Mitigación:

    - Ejecución de escaneos en las imágenes del contenedor para comprobar que estas no permitan la inyección de código.
    - Limitar la creación de pods y controladores a usuarios y cuentas de servicio.
    - Limitar las capacidades de los contenedores solo al uso que requiere.

4.5.3. Persistencia

La persistencia es una táctica con la que el atacante intenta crear una nueva manera de acceso para persistir en el clúster, por si en algún momento pierde su punto de acceso inicial.

4.5.3.1. Puerta trasera en contenedor

Los atacantes ejecutan su código en un contenedor del clúster. Mediante el uso de daemonsets o deployments, se garantizan que su código se ejecute en uno o en varios nodos del clúster.

- Mitigación:

    - Política de menor privilegio, limitando la creación de pods y controladores
    - Limitar de montaje de volúmenes mediante el controlador de admisión.
    - Escaneo de vulnerabilidades en busca de código malicioso.

4.5.3.2. Montaje de volúmenes con permisos de escritura en el servidor

Los volúmenes hostPath montan un archivo o directorio desde el host en el pod. Un usuario con permisos de creación de contenedores, puede hacerlo añadiendo un volumen hostPath, que luego puede utilizarse para ganar persistencia en el clúster.

- Mitigación:
    - Limitar las rutas en las que se permite el montaje parcial o completo y marcar las rutas de host como de solo lectura.

4.5.3.3. Cronjobs

Utilización del programador de tareas con el fin de crear contenedores que ejecuten código malicioso.

- Mitigación:
    - La política de roles debe controlar las cuentas pon permisos de creación de contenedores

4.5.3.4. Controlador de admisión malicioso

El controlador de admisión permite al API server capturar solicitudes y aplicarle reglas que contiene codificadas y que darán como resultado la validación de la solicitud, si ésta cumple las reglas que se encuentran codificadas en el controlador de admisión.
El controlador de admisión está formado por el controlador de validación (*ValidatingAdmisionWebHook*) y por el controlador de mutación (*MutatingAdmisionWebHook*). El controlador de mutación puede alterar las solicitudes capturadas por el controlador de admisión.
El ataque intenta aprovechar la capacidad de modificar solicitudes del controlador de mutación, con el fin de ganar persistencia o ajustar los permisos del contenedor.

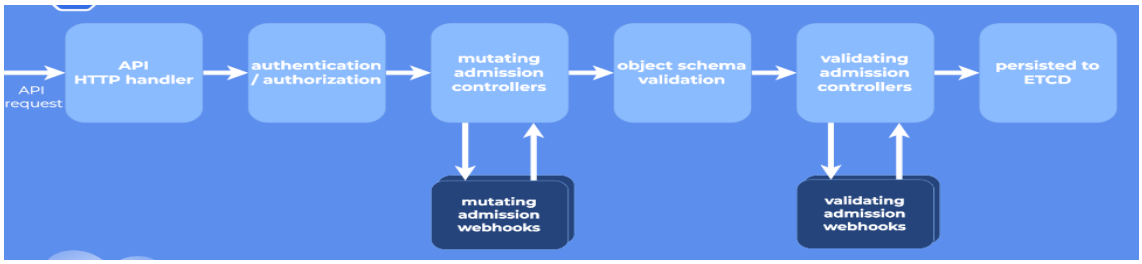

Imagen 11. Flujo de solicitudes en la API [16]

- Mitigación:
    - Aplicar política de RBAC con privilegio restringiendo el control sobre el controlador de validación y de mutación a usuarios y cuentas de servicio del clúster.

4.5.4. Escalado de privilegios

Esta táctica recoge como los atacantes intentan obtener mayores permisos a fin de poder controlar el clúster bien sea agregando nuevos permisos o utilizando roles con permisos para realizar las acciones que necesitan.

4.5.4.1. Contenedor privilegiado

Los contenedores privilegiados son aquellos que ejecutan con permisos de root por lo que el contenedor puede realizar casi todas las acciones que se realizan directamente host. El acceso a estos contenedores permite la creación de nuevos contenedores privilegiados y da accesos a los recursos del host.

- Mitigación:
    - Limitar y controlar el uso de contendores privilegiados.
    - La política RBAC o control de roles debe controlar la asignación de permisos para crear contenedores privilegiados.
    - Política de garantía de imagen de contenedor que garantice que los contenedores provienen de una fuente de segura, han pasado los escaneos de vulnerabilidades pertinentes.

4.5.4.2. Roles con función de administrador

Este tipo de roles tienen acceso completo al clúster, y a su vez puede asignar nuevos enlaces a administración a otros roles o dar privilegios elevados a otros roles.

- Mitigación:
    - Limitar los permisos que se conceden a los diferentes roles y por lo tanto la que ha de controlar la asignación de este permiso y a quien le faculta para poder dar este permiso.

4.5.4.3. Montaje de volúmenes en la ruta del servidor

Los volúmenes hostPath montan un archivo o directorio desde el host en el pod. Un atacante puede obtener acceso al host y a sus recursos y desde ahí comprometer otros contenedores alojados en el mismo host.

- Mitigación:
    - Limitar las rutas en las que se permite el montaje parcial o completo y marcar las rutas de host como de solo lectura.

4.5.4.4. Acceso a recursos cloud (Solo aplicable en cloud)

Esta técnica es utilizada por una atacante que consigue acceder a un contenedor en la nube, para acceder a otros recursos de la nube fuera del clúster. Para ello utilizan cuentas de servicio, utilizadas por el proveedor para realizar gestiones en nuestro clúster.

- Limitar las rutas de archivos que permiten montaje de host o rechazar los montajes.
- Pods con identidades asignadas dedicadas.
- Restringir el acceso de los pods a los servicios de metadatos.

4.5.5. Evasión

Esta táctica recoge aquellos mecanismos utilizados para evitar que se descubra que el clúster ha sido atacado, con el fin de evitar que se tomen medidas para mitigarlo.

4.5.5.1. Limpieza de logs

Se intenta que no se vean los registros con los tiempos del sistema que puedan dar información de una actividad por parte de un atacante.

- Mitigación:
    - Guardar los ficheros de los en un lugar independiente.

- Restringir el acceso a los registros del contenedor.

4.5.5.2. Borrado de eventos

Los eventos de kubernetes recogen el paso a paso de acciones relevantes para la seguridad del clúster o sus componentes.

- Mitigación:

    - Guardar los ficheros de los en un lugar independiente.
    - Restringir los permisos para eliminar eventos.

4.5.5.3. Pod y contenedor nombre similares

La utilización de sufijos en el nombre de los elementos que son creados por los controladores del clúster puede posibilitar la ocultación de elementos maliciosos dentro del clúster

- Mitigación:

    - Política de garantía de imagen de contenedor que garantice que los contenedores provienen de una fuente de segura, han pasado los escaneos de vulnerabilidades pertinentes.

4.5.5.4. Conexión desde el proxi (Solo aplicable en cloud)

Uso de conexiones a través de proxy o conexiones anónimas con el fin de evitar la identificación por medio de la IP.

- Mitigación:

    - Restringir el acceso al servidor API desde direcciones IP conocidas.
    - Permitir acceso solo de redes proxi conocidas.
    - Solicitar autenticación al servidor de la API.

4.5.6. Acceso con credenciales.

Esta táctica recoge el conjunto de acciones que van destinadas a que un usuario pueda adquirir cualquier tipo de credenciales de acceso que luego de capacidad de acceso y maniobra dentro del clúster.

4.5.6.1. Listar secretos

Los secretos son el almacén utilizado por Kubernetes para almacenar de manera protegida credenciales. Por lo que un usuario con acceso a listar podría obtener los datos guardados y desencriptarlos, ya que estos están en base 64.

- Mitigación:
    - Limitar el acceso a secreto y de los usuarios que tienen permiso para listarlos.
    - Restringir el acceso a etcd.
    - Eliminar los secretos no utilizados en el clúster.

4.5.6.2. Montaje en servidor con cuenta de servicio

Una vez que se produce el acceso a un contenedor, puede utilizar se una cuenta de servicio sin la correcta política de roles aplicada para acceder a otros sitios del sistema.

- Mitigación:
    - Política de roles asignando los permisos mínimos a las cuentas de servicio
    - Implementar una política en la que las rutas de host sean de solo lectura o limitar las rutas en las que se puede realizar un montaje de host.

4.5.6.3. Acceso con cuenta de servicio a contenedor

Esta técnica está relacionada con la anterior o es la base para ella. Como ya hemos comentado las cuentas de servicio pueden dar acceso a todo el servidor y permitir al atacante actuar libremente.
Si existe una política RBAC el atacante solo podrá realizar las acciones permitidas.

- Mitigación:
    - Correcta política RBAC con privilegios mínimos.

4.5.6.4. Credenciales de aplicación en ficheros de configuración

Los ficheros de configuración disponen de una menor protección que los secretos. Si se produce un acceso indebido al contenedor, el atacante puede hacerse con los secretos y utilizarlos.

- Mitigación:
    - Evitar usar credenciales de texto en los ficheros de configuración.

4.5.6.5. Acceso con credenciales de identidad administrada

Los proveedores de servicios como método para asignar recursos les asignan identidades administradas. Si un atacante tiene acceso a un nodo, puede acceder a la máquina virtual a través de servicio de metadatos de instancia (IMDS) y podría obtener el token para obtener mayores permisos.

- Mitigación:
    - Limitar el acceso al servicio de metadatos desde los pods.
    - Asignar una identidad de nube de dedicada a cada pod y evitar que utilicen la identidad del nodo en la nube.

4.5.6.6. Controlador malicioso de admisión

Esta técnica se ha descrito en el punto 4.5.3.4.
Esta técnica ha sido catalogada dentro de dos tácticas.

4.5.7. Descubrimiento

Esta táctica recoge las técnicas que permiten al atacante explorar todo el entorno del clúster una vez que ha conseguido acceder y luego mediante un movimiento lateral acceder a otros recursos.

4.5.7.1. Acceso a la Api server de kubernetes

Mediante esta técnica, se envían peticiones API REST, para ir recabando información de los contenedores, secretos y otros recursos.

- Mitigación:

- La política RBAC debe establecer una configuración que limite los privilegios de usuarios y cuentas de servicio que tienen acceso a la API de kubernetes.
- En la nube, restringir el acceso a IPs de confianza

4.5.7.2. Acceso a la Api server de kubelet

Un atacante con acceso de red sobre host realiza peticiones API REST de kubelet las siguientes consultas con el fin de obtener los pods y la segunda datos sobre el nodo.

- https://[IP_NODO]:10255/pods/
- https://[IP_NODO]:10255/spec/

- Mitigación:

  - Evitar usar el puerto no seguro 10255.
  - Establecer políticas de red de bloqueo del puerto de kubelet bloqueando los puertos 10250 y 10255.
  - Políticas RBAC que limiten la comunicación la comunicación con kubelet y solo las cuentas de servicio con los permisos pertinentes puedan comunicarse con kubelet.
  - Limitar los permisos de Kubelet a pods y nodos.

4.5.7.3. Mapeo de red

Se intenta mapear la red del clúster y poder realizar búsquedas para obtener información sobre las aplicaciones en ejecución y posteriormente intentar alguna vulnerabilidad conocida. Una vez que se obtiene acceso a un contenedor, tiene continuar con la técnica al resto de la red ya que no existen restricciones de comunicación entre los nodos por defecto.

- Mitigación:

  - Políticas de segmentación de red para limitar el tráfico entre nodos.

4.5.7.4. Acceso al panel de control

El panel de control mediante su cuenta de servicio dispone de permisos determinados por enlace para realizar ciertas acciones sobre el clúster. Por lo que conseguir acceso al panel nos da posibilidades de obtener información sobre otros recursos.

- Mitigación:

    - Si no se necesita el panel (un punto menos de acceso a los recursos) lo mejor sería deshabilitarlo.
    - En caso de ser necesario estableceríamos la correspondiente política RBAC limitando los privilegios de la cuenta de servicio.
    - Restricción de entrada mediante la política de red.

4.5.7.5. **Instancia de la Api de metadatos** (Solo aplicable en cloud)

Los proveedores de la nube facilitan servicios de metadatos de instancia para que los usuarios puedan obtener información acerca de los recursos del host que aloja el clúster (red, discos,.), a pesar de que estas instancias solo son accesibles desde el clúster en cuestión, si este es comprometido daría posibilidades al atacante de realizar llamadas API a la instancia y poder obtener metadatos sobre la instancia.

- Mitigación:

    Restringir el acceso de pods al servicio de metadatos.

4.5.8. Movimiento lateral

Esta táctica recoge las técnicas que una atacante utiliza para moverse dentro del entorno atacado (contenedores, elementos relacionados como el host alojador o el entorno en la nube).

4.5.8.1. Acceso a recursos del cloud

Los atacantes que obtienen acceso a las credenciales de cuentas creadas para dar servicio por parte del proveedor pueden usar estas credenciales para acceder o modificar los recursos de la nube.

- Mitigación:

    - Limitar las rutas de archivos que permiten montaje de host o rechazar los montajes.
    - Pods con identidades asignadas dedicadas.
    - Restringir el acceso de los pods a los servicios de metadatos.

4.5.8.2. Cuenta de servicio del contenedor

La cuenta de servicio, si no se han aplicado políticas RBAC, puede disponer de permisos determinados por enlace para realizar ciertas acciones sobre el clúster. Por lo que conseguir acceso al panel nos da posibilidades de obtener información sobre otros recursos.

- Mitigación:
    - Establecer la política RBAC adecuada sobre la cuenta de servicio.
    - Deshabilitar el montaje de volúmenes automático de la cuenta de servicio.

4.5.8.3. Red interna del Clúster

Esta técnica aprovecha que por defecto no existen restricciones de red para que una vez se ha accedido a un contenedor tiene acceso a la red de otros contenedores del clúster.

- Mitigación:

    Utilizamos la segmentación de red para controlar la comunicación entre los pods.

4.5.8.4. Credenciales en archivos de configuración de aplicaciones

Suele ser habitual que los desarrolladores guarden las credenciales de las aplicaciones en los ficheros de configuración. Mediante consultas API o mediante el terminal, se puede acceder a los ficheros de configuración y obtener las credenciales almacenadas en los ficheros.

- Mitigación:

    Revisión de los ficheros de configuración para evitar que en ellos se encuentren secretos.

4.5.8.5. Montaje de volúmenes con permisos de escritura en el servidor

Los volúmenes hostPath montan un archivo o directorio desde el host en el pod. Un usuario con permisos de creación de contenedores, puede hacerlo añadiendo un volumen hostPath, que luego puede utilizarse para ganar persistencia en el clúster.

- Mitigación:

    - Limitar las rutas en las que se permite el montaje parcial o completo.
    - Marcar las rutas de host como de solo lectura.

4.5.8.6. Alteración del CoreDNS

CoreDNS es el servicio de DNS de kubernetes y configurable a través del archivo Corefile y que se almacena en el ConfigMap. Este ataque intenta la modificación del archivo y alterar el comportamiento DNS del clúster.

- Mitigación:

    Utilización de RBAC y privilegio mínimo sobre ConfigMap.

4.5.8.7. Envenenamiento ARP y usurpación de IP

Envenenamiento ARP consiste en modificar la tabla ARP haciendo pasar la dirección MAC del atacante con la dirección IP de otro equipo.

Kubenet es un complemento de red básico que implementa la comunicación entre nodos a través de un puente o políticas de red. Normalmente se utiliza por los proveedores de la nube.

Al ser un puente un componente de capa nivel 2, posibilita que pueda sufrir envenenamiento ARP facilitando otros movimientos laterales al atacante.

- Mitigación:

    Utilizar interfaces de red que no permitan envenenamiento por ARP.

4.5.9. Colección

Esta táctica agrupa las técnicas utilizadas para recopilar datos del clúster.

4.5.9.1. Imágenes desde registro privado

Esta técnica aprovecha el uso de repositorios privados para evitar que un atacante pueda acceder a nuestros contenedores. Para obtener la imagen a través de Docker se necesita el token de acceso para poder descargar la imagen. Una mala gestión en la protección del token puede ocasionar que un atacante disponga de acceso a nuestro repositorio de imágenes, disponiendo de ellas sin límite y pudiendo comprometer el clúster en sucesivas instalaciones.

- Mitigación:
    - Utilización de un IAM
    - Política adecuada de RBAC que el evite el acceso a secretos a usuarios no autorizados.

4.5.9.2. Recopilación de datos del pod

Mediante el uso de comandos administrativos, el atacante puede tener acceso directo a la información del pod.

- Mitigación:

    Política adecuada de RBAC que el evite que controlen o ejecuten comandos.

4.5.10. Impacto

Su nombre es descriptivo de esta táctica donde las técnicas utilizadas intentan ocasionar destrucción o influir en su comportamiento normal del sistema.

4.5.10.1. Destrucción de datos

Intento por parte de los atacantes de destruir datos o elementos del clúster.

- Mitigación:

- Uso de un proveedor en la nube para almacenar los datos de la aplicación.
- Copia de seguridad de volúmenes montados en pods.

4.5.10.2. Secuestro de recursos

Técnica que consiste en realizar tareas a través de un recurso comprometido.

- Mitigación:
    - Utilización de políticas RBAC para restringir la creación de pods.
    - Utilización de módulos de seguridad como AppArmor y SELinux.

4.5.10.3. Denegación de servicio (CVE-2019-1002100)

Consiste en realizar un ataque de denegación que ocasiona que el servicio no se encuentre disponible para usuarios legítimos.

- Mitigación:
    - Permitir el acceso al servidor API solo a direcciones IP conocidas.
    - Limitar recursos en los contenedores.
    - L**imita**r la ejecución de procesos no deseados.

5. Seguridad en Devops aplicada a *fases de desarrollo*.

En esta sección vamos a desarrollar una primera parte de la implementación del deployment, que consiste en un proceso de integración continua CI/CD.
Mediante este proceso llegaremos a tener una aplicación segura lista para ejecutar en nuestro clúster de Kubernetes.

5.1. Preparación del entorno de trabajo

- El primer paso sería comprobar que en la BIOS existe la opción de virtualización y que esta se encuentra activada.

- Creación de una máquina virtual en VMware Workstation en su última versión y sobre ella se ha instalado como sistema operativo Windows 10 con versión Educativa. (La elección de la versión educativa se debe a que la virtualización Hyper-V solo se puede emular en esta versione o en estas otras Pro o Enterprise.)

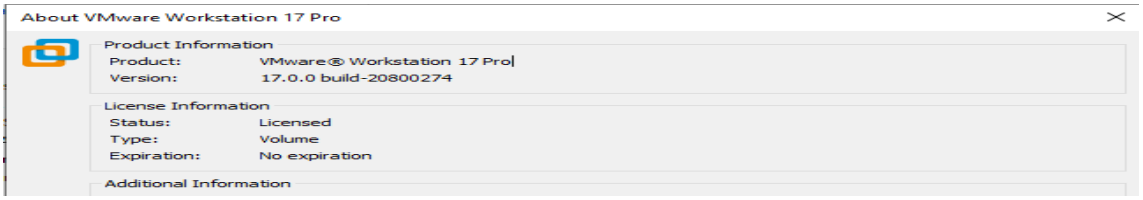

Imagen 12. Versión VMware

En cuanto a las características que hemos configurado en la máquina virtual son las siguiente. Inicialmente establecimos una cantidad de memoria inferior, pero el uso de Docker en su versión de escritorio ocasiona un gran uso de recursos por lo que se ha decidido establecer 12GB de memoria.

Es importante activar en VMware, la virtualización para que el nuevo equipo virtualizado pueda coger la virtualización del equipo anfitrión.

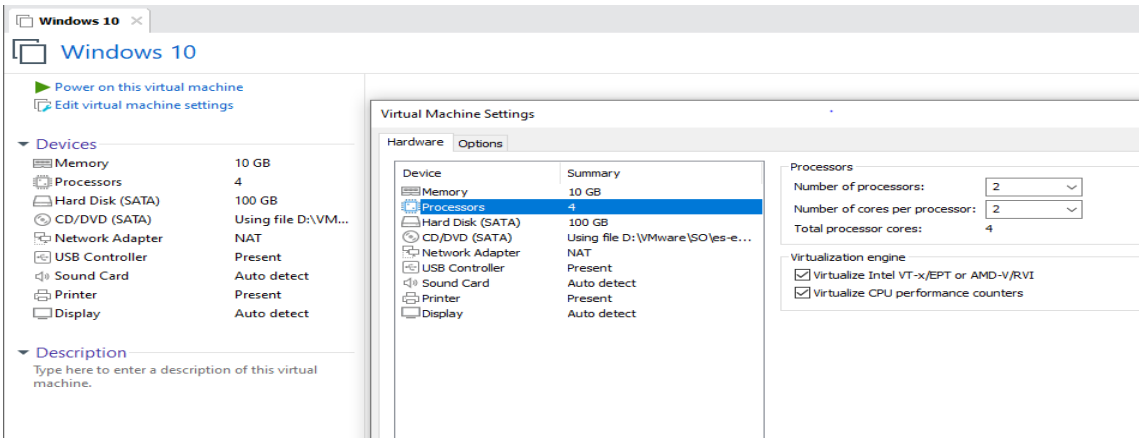

Imagen 13. Requisitos hardware maquina virtualizada

- Una vez que tenemos máquina y sistema operativo, procederemos a activar Hyper-V en el equipo virtualizado.

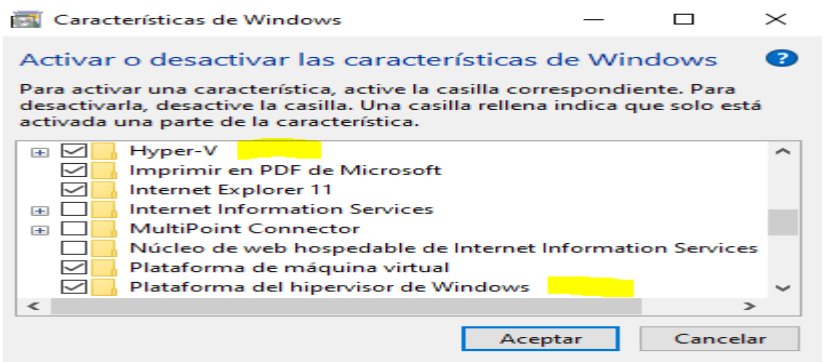

Imagen 14. Características de Windows para virtualizar

- Una vez que hemos adquirido las características necesarias, procedemos a instalar minikube, pero antes, instalamos el gestor de paquetes para Windows, chocolatey.

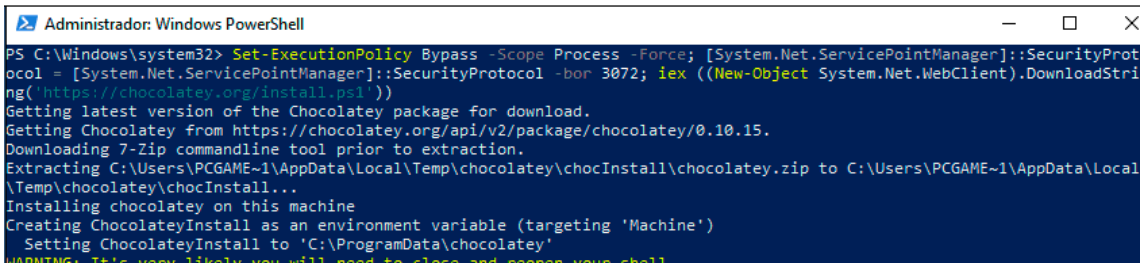

Imagen 15. Instalación de Chocolatey

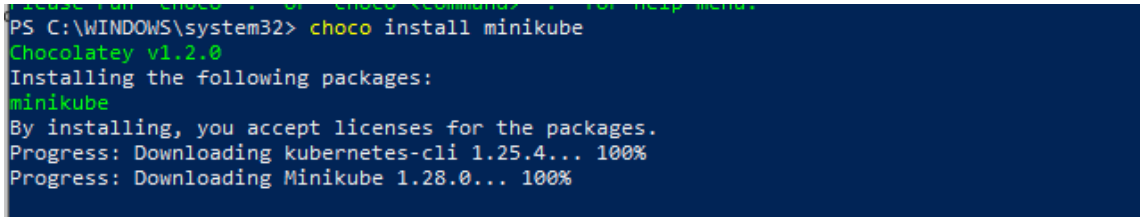

Imagen 16. Instalación de minikube

- Una vez que tenemos instalado minikube podemos lanzar la simulación del clúster, minikube ya incorpora kubectl. Por lo que procedemos a iniciar nuestro clúster bajo virtualización Hyper-V.
  Se ha utilizado el comando “--force” debido a un bug que existe con la ejecución de PowerShell. Este problema ocasiona que no se detecte que se está ejecutando PowerShell como administrador.

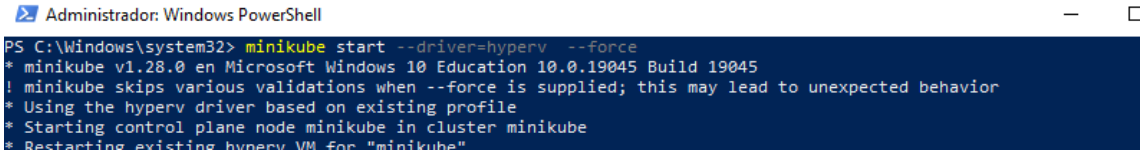

Imagen 17. Iniciamos nuestro clúster

- Una vez iniciado podemos comprobar en la plataforma de virtualización de Windows, como nuestro clúster se virtualizado en ella.

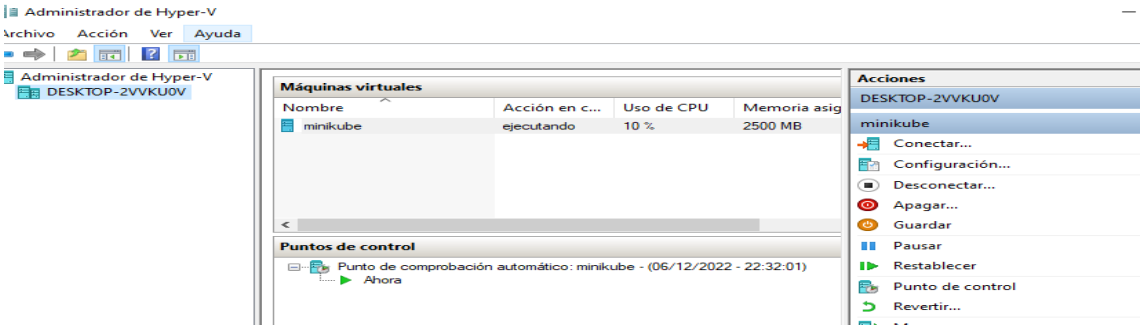

Imagen 18. Administrador de Hyper-V

- Una vez que hemos comprobado que nuestro clúster funciona, comprobamos que realmente devuelve datos.

```
PS C:\Windows\system32> kubectl get namespaces
NAME              STATUS   AGE
default           Active   3d21h
kube-node-lease   Active   3d21h
kube-public       Active   3d21h
kube-system       Active   3d21h
tfm               Active   3d21h
```

Imagen 19. Petición de los namespaces del cluster.

Ya tememos nuestro cluster funcionando.

El siguente paso es la creacion de la aplicación que desplegaremos en nuestro cluster.

5.2. Codificación

- Para nuestra aplicación web nos decantamos por una aplicación web MVC CORE.

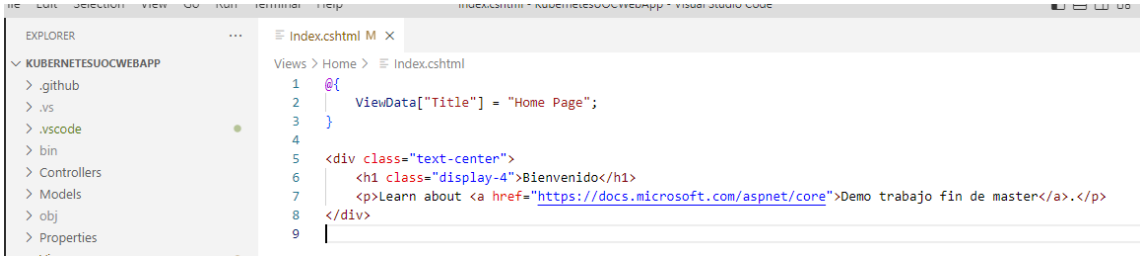

Imagen 20. Código HTML

- Antes de crear nuestra aplicación creamos un repositorio privado de código en GitHub. El cual nos garantizará control de versiones y otros elementos que detallaremos más adelante.

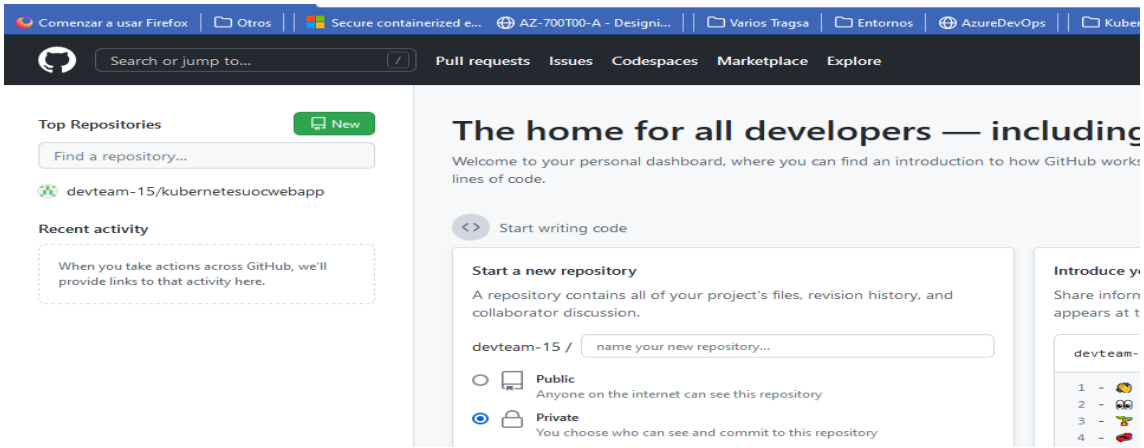

Imagen 21. Repositorio privado de GitHub

- Realizamos la instalación de docker desktop mediante el instalador.

```
PS C:\Windows\system32> docker --version
Docker version 20.10.21, build baeda1f
PS C:\Windows\system32>
```

Imagen 22. Versión de Docker

- En GitHub procedemos a crear una toquen de acceso que nos servirá para publicar nuestra imagen en el registro seguro de contenedores de GitHub.

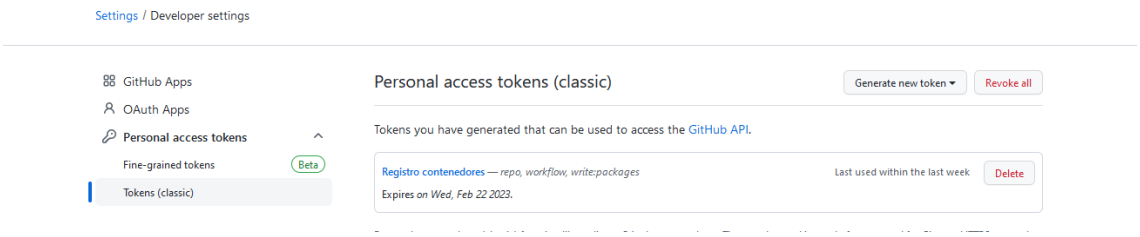

Imagen 23. Almacén de tokens GitHub

- Utilizamos una instancia de Visual Studio Code y generamos una aplicación web de “Dummie”.

  Hemos agregado al proyecto web el fichero dockerfile al proyecto con los elementos que necesitamos para crear el contenedor.

  En este paso hemos realizado a través de Git-cli una subida del código a nuestro repositorio en GitHub quedando versionado y a salvo usuarios no deseados.

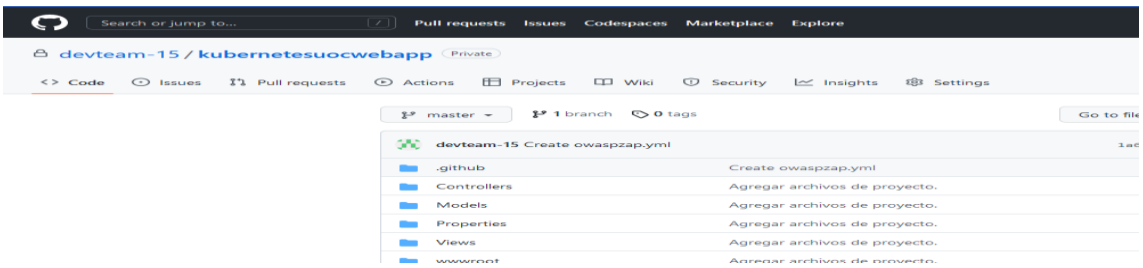

Imagen 24. Repositorio de codigo

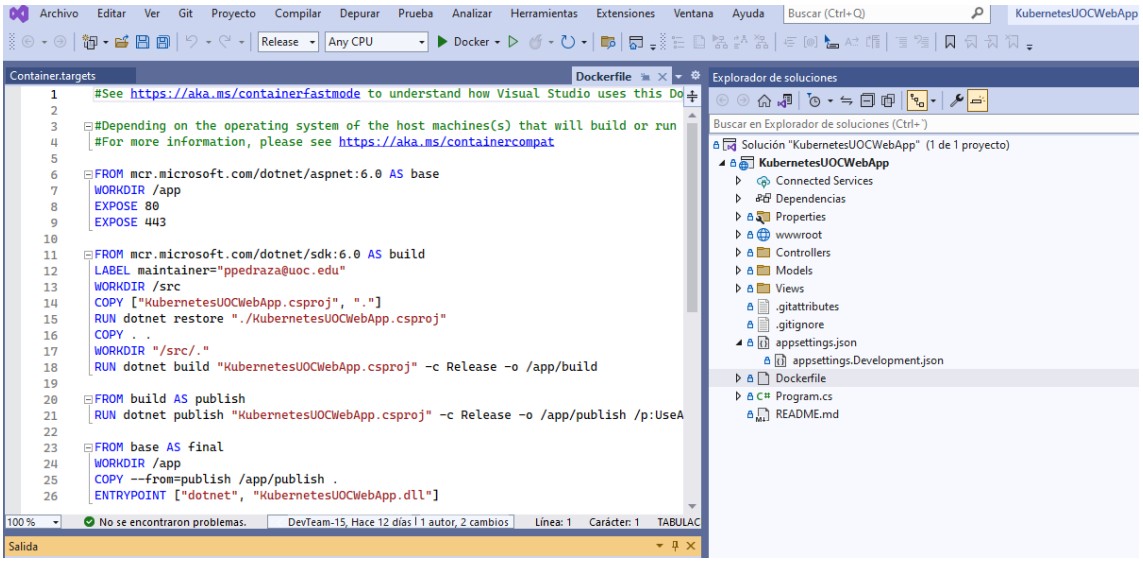

Imagen 25. Aplicación lista para hacer build a docker

- Procedemos a crear la imagen del contendor en docker. Y aprovecharemos para llamarla del siguiente modo ghcr.io/usu_github/repositorio.

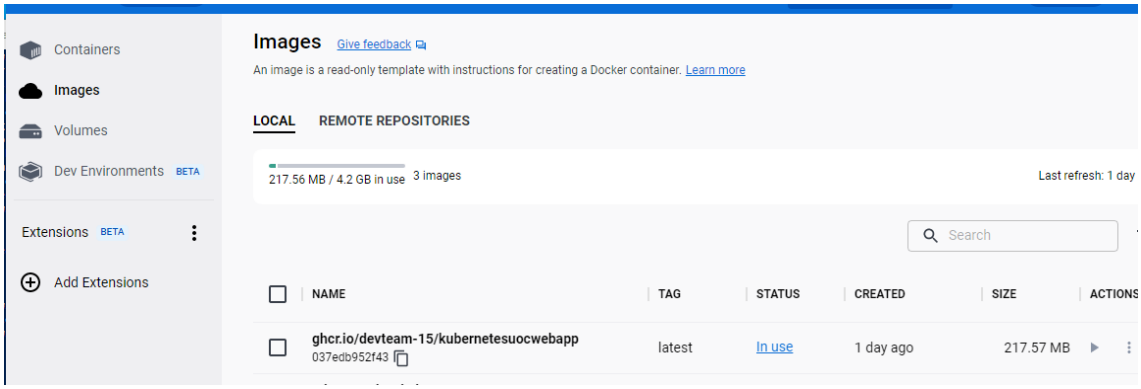

Imagen 26. Imagen Docker para repositorio seguro

- Realizamos un docker scan para realizar un análisis de código estático y si todo es correcto subiremos la imagen al registro.

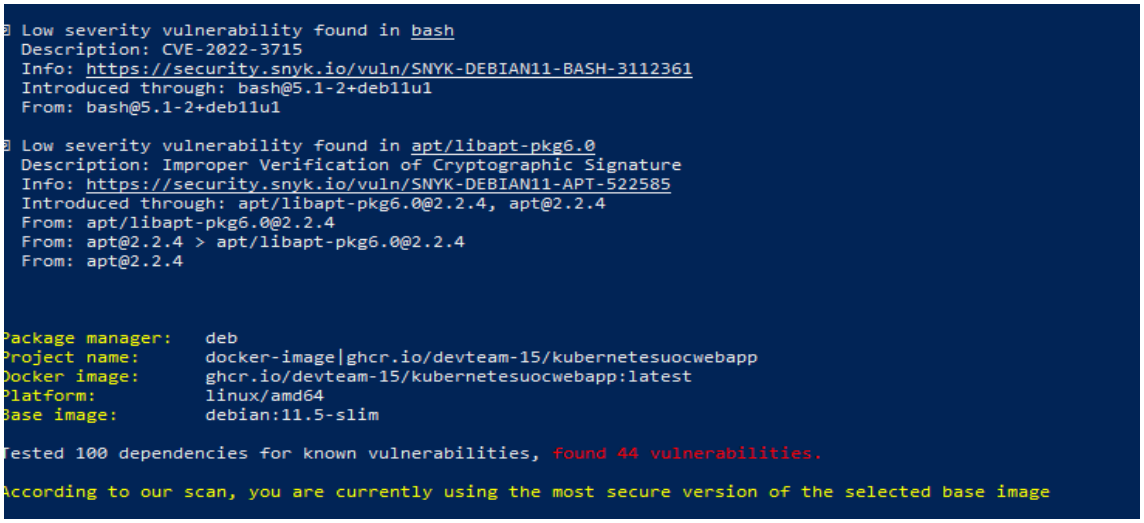

Imagen 27. Vulnerabilidades detectadas en análisis estático

En nuestro caso existen vulnerabilidades sobre la imagen base, y esta es la más segura que existe.

5.3. Compilación

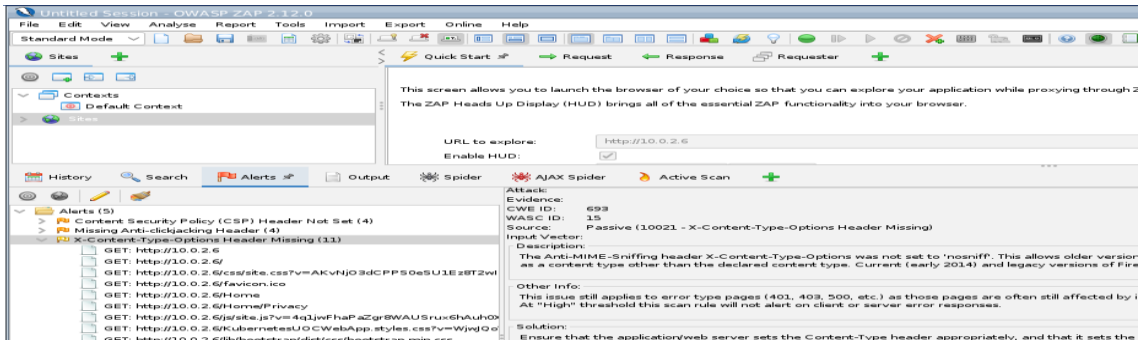

Imagen 28. Instalamos ZAP

En nuestro caso existen vulnerabilidades sobre la imagen base, y esta es la más segura que existe.

- Una vez que obtenemos la imagen procederemos a guardarla en el registro de contenedores mediante docker push...

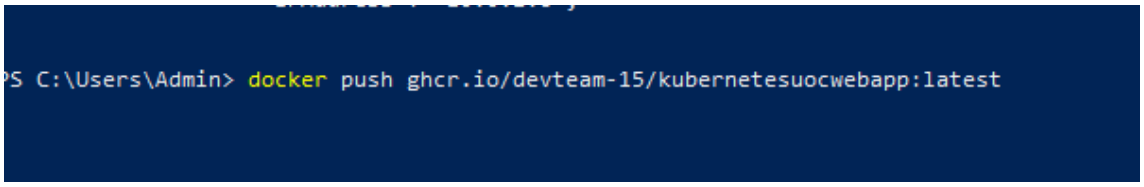

Imagen 29. Docker push

- Una vez que tenemos la imagen en el repositorio procedemos a instalarla en nuestro clúster, mediante SSH descargamos la imagen.

6. Kubernetes, componentes y funcionamiento

6.1. Descripción de los componentes de Kubernetes

Kubernetes es una plataforma de orquestación de contenedores de código abierto y entre sus funciones están despliegue, balanceo de carga, escalado y gestión de contenedores.

La arquitectura de un clúster de Kubernetes se encuentra formada por nodos, que son las máquinas de trabajo que pueden ser físicas o virtuales **encargadas de ejecutar las cargas de trabajo, típicamente en formato contenedores.**

La arquitectura de un clúster de Kubernetes se encuentra formada por nodos, que son las máquinas de trabajo que pueden ser físicas o virtuales.
Dentro de estos nodos nos encontramos con unos que administran el clúster, conocidos como “Control Plane” y nodos que realizan el trabajo y son denominados como Workers.

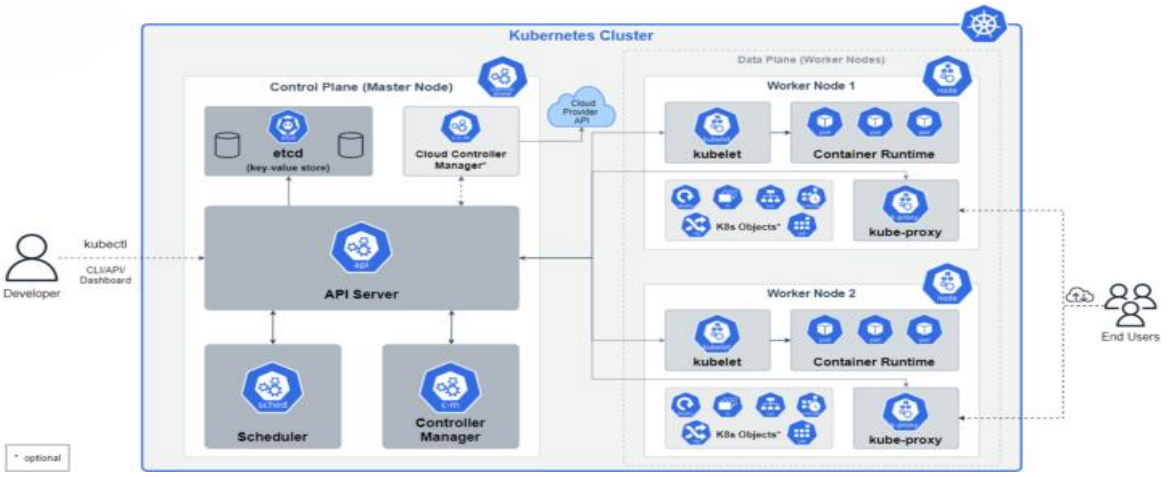

Imagen 30. Componentes de un clúster de Kubernetes

Control Plane:

Son los nodos que controlan el clúster. En la imagen anterior aparecen representados los elementos más importantes que lo componen. Estos elementos suelen instalarse en la misma máquina, aunque pueden instalarse en máquinas diferentes.

- Kube-apiserver (API). Este es el punto de comunicación del nodo principal con el resto de los nodos del clúster.
- etcd. Este componente es una base de datos de tipo clave-valor de alta disponibilidad en la cual se persiste el estado del clúster. Cabe recalcar que si este componente se corrompe el clúster **dejará** de funcionar, por lo que es muy importante tener un plan de copias de seguridad.
- Kube-scheduler: Se encarga de decidir donde se despliega un nodo. Utiliza el consumo de recursos para rastrear y manejar las cargas de trabajo de un nodo en el clúster.
- Kube-controller-mánager. Se encarga de ejecución de los controladores (procesos de escucha de eventos que se producen en el clúster.)
- Cloud-controller-mánager. Este componente solo lo encontramos si existe integración con algún proveedor en la nube. Ejecuta los controladores en la nube de igual manera que el kube-controller-mánager lo *realiza* para entornos locales.

Nodos:

Los componentes que describimos a continuación podrían ejecutarse además de en los nodos trabajadores, en el Control Plane, aunque esto no suele realizarse normalmente.

- Container-Runtime. El motor de contenedores como su nombre indica se encarga de la ejecución de los contenedores, las versiones modernas utilizan el motor de CRI-O (es una implementación realizada a partir de Container Runtime Interface).

Este componente es necesario en todos los nodos donde se desplieguen Pods.

- Kubelet. Ejecuta los contenedores en los Pods.
- Kube-proxy. Se encarga del control del tráfico de red del clúster.

6.2. Funcionamiento de un clúster de Kubernetes

Una vez que conocemos los componentes principales del clúster, procederemos a realizar una breve explicación de cómo funciona el flujo de trabajo de Kubernetes.

6.2.1. A través del cliente kubectl, mediante programación o usando una interfaz, se generan unas llamadas API Rest y lo enviará a Kube-apiserver, el cual es el encargado de la gestión de peticiones al clúster, tanto internas como externas.

6.2.2. Una vez que Kube-apiserver ha gestionado las llamadas recibidas mediante API Rest, llama al proceso Kube-scheduler el cual seleccionará un nodo apropiado de entre los disponibles para ejecutar un Pod.

6.2.3. Kube-Scheduler devuelve a Kube-apiserver la respuesta que contendrá la información del nodo que ha de ejecutar la petición.

6.2.4. Kube-apiserver, ser comunica con el nodo seleccionado a través del componente Kubelet y utilizando el container-runtime se encarga de crear y ejecutar los contenedores que pertenecen al Pod.

6.2.5. Una vez que los pasos anteriores se han realizado correctamente, información correspondiente a esta tarea se guardaran en etcd, que almacenará todos los datos de nuestro clúster

6.2.6. Este trabajo y sus especificaciones se grabarán en una base de datos centralizada etcd. Su trabajo consiste en conservar y proporcionar acceso a todos los datos del clúster.

6.2.7. Kube-proxy se encargará de enrutar todo el tráfico de red a su destino correspondiente.

Una vez que conocemos como se relacionan los elementos dentro del clúster, con los recursos disponibles y con los elementos externos del clúster, nos ayudará a poder superar la tarea de proteger un clúster de Kubernetes.

Si bien Kubernetes ha sido pensado para gestionar desde pequeños a grandes clústeres de contenedores, esto hace que, a mayor tamaño, mayor complejidad en la gestión, al ser un sistema abierto ocasiona que el usuario deba tomar varias decisiones, que nos permite ir desde la elección del sistema operativo a utilizar, o si el clúster será local o se realizará en un proveedor en la nube. Por lo tanto, con respecto a la seguridad ocurre de igual manera, ya que, si bien las interfaces abstraen al usuario de toda la gestión de la infraestructura, es el usuario el que ha tener un plan de seguridad integral, que ha de analizar los posibles puntos de ataque y deberá aplicar las practicas recomendadas.

6.3. Implementación en Kubernetes

Este punto sirve como demostración de cómo realizar un deployment básico en un clúster de Minikube

Imagen 31. Acceso a minikube ssh

- Creamos un fichero .yaml para realizar el deployment de nuestra imagen.

Imagen 32. Fichero de deployment e instalación

- Comprobamos que el deployment es correcto

```
PS C:\Windows\system32> kubectl get deployments
NAME                  READY   UP-TO-DATE   AVAILABLE   AGE
kubernetesuocwebapp   3/3     3            3           17s
PS C:\Windows\system32>
```

Imagen 33. Comprobación del deployment ok

- Exponemos la aplicación a internet

```
PS C:\Windows\system32> kubectl expose deployment kubernetesuocwebapp --type=NodePort
service/kubernetesuocwebapp exposed
PS C:\Windows\system32>
```

Imagen 34. Exponemos la aplicación a internet.

- Probamos que accedemos al clúster.

```
PS C:\Windows\system32> minikube service kubernetesuocwebapp --url
http://172.26.75.143:32497
PS C:\Windows\system32>
```

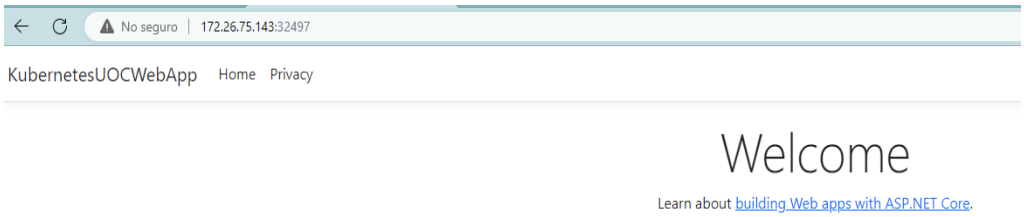

Imagen 35. Web publicada en internet

7. Seguridad en el clúster con MITRE.

7.1. Autenticación multifactor

La autenticación multifactor es una capa de seguridad intermedia utilizada en el proceso de inicio de sesión. El usuario que intenta logarse en el sistema han de combinar tecnologías de verificación o factores de autenticación de dos o más grupos.
Los factores se dividen en tres grupos: algo que conoces (contraseña), algo que tienes (notificación recibida en dispositivo móvil) y algo que eres (huella dactilar).

Esta solución sería aplicable a clúster en la nube y asociadas con un gestor de identidades y acceso (IAM)

Imagen 36. Autenticación multifactor

7.2. Principio de privilegio mínimo.

El principio de mínimo privilegio establece que cada proceso ha de acceder solo a los recursos y a la información necesario para realizar su función. Kubernetes dispone de métodos de autenticación y autorización para habilitar el control de acceso y realizar la asignación de permisos, que permiten a los usuarios interactuar con los objetos de kubernetes en el clúster. El mecanismo de asignación de permisos es lo que se conoce como acceso basado en funciones / roles (RBAC).

7.2.1. Autenticación.

Existen dos tipos de entidades que pueden acceder al clúster, los usuarios y las cuentas de servicio.
Las cuentas de servicio son gestionadas de recurso “Service account” y son utilizada por los procesos que acceden a la API.
Las cuentas de usuarios son gestionadas desde fuera del clúster, ya que kubernetes no dispone de ningún recurso que represente a un usuario o grupo de estos.

Kubernetes permite la utilización de diferentes formas de autenticarse como son certificados del cliente, tokens o proxies de autenticación. Se permite también el uso de varios simultáneamente, pero una vez que el uno autentique, se procederá a la fase de autorización.

Minikube por defecto se encuentra configurado con autenticación de certificados de cliente. Este certificado se crea con la instalación y existe uno en la maquina donde se instala minikube para firmar las autorizaciones y el mismo certificado en el nodo maestro del clúster para validar las peticiones que llegan firmadas.

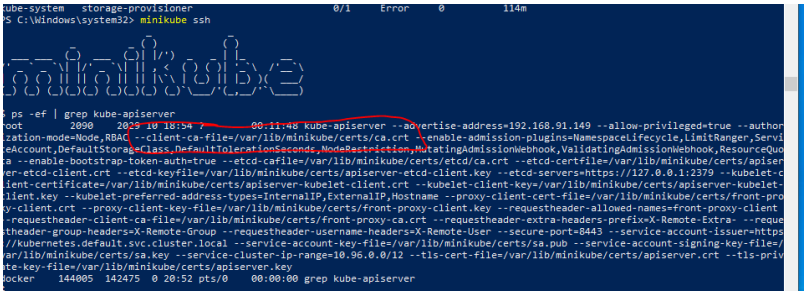

Imagen 37. Ruta certificados digitales

Para autenticar al usuario el primer paso es crear su clave publico/privada. Ex

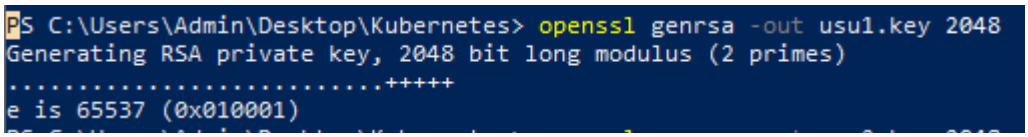

Imagen 38. Generación clave privada usuario

A continuación, creamos la petición de firma del usuario. En nuestro ejemplo, le asignamos un grupo (desarrollo).

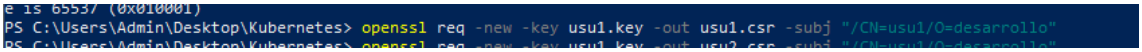

Imagen 39. Solicitud de firma a CA

El certificado se firma con la clave CA de minikube para sea válido.

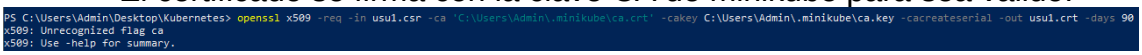

Imagen 40. Firma de certificado con clave CA

Se agregan las credenciales de usuario en la máquina virtual.

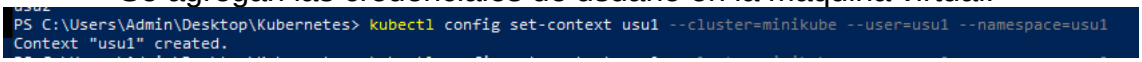

Imagen 41. Creación de contexto de prueba

Comprobamos que se ha realizado la inclusión del usuario en la lista de usuarios del clúster.

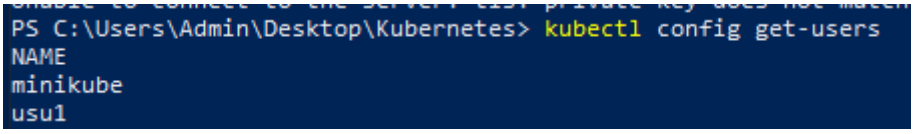

Imagen 42. Test de nuevo usuario

La información sobre el usuario se almacena en el fichero de configuración de Kubectl (Config). Este fichero incluye también, información acerca de los clústeres y los contextos (guardan información de como una cuenta se relaciona con un clúster.)

7.2.2. Autorización.

Cuando se crea un usuario o cuenta de servicio, por defecto no tienen asociado ningún tipo de permiso, por lo que no pueden realizar ninguna acción en el clúster.

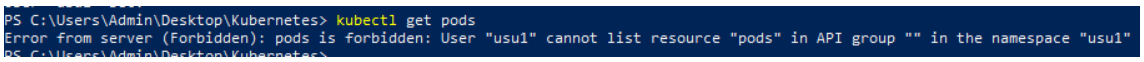

Imagen 43. comprobación restricción de permisos

Mediante la Api de kubernetes y otros recursos, podremos definir los permisos que se asignan a cuentas de servicio, usuarios y grupos.

Existen cuatro maneras de autenticar:

Node: Concede una autorización especial a las peticiones realizadas desde kubelet.

ABAC (Attribute-Based Access Control): Las políticas se definen a partir de atributos que poseen los usuarios.

RBAC (Role-Based Access Control): Control de acceso basado en roles.

Webhook: Cuando se produce un evento, se realiza una llamada a un punto de acceso que recibe la información del evento y toma la decisión de permitirlo o no el acceso.

Para activar RBAC en el clúster se ha de ejecutar el siguiente comando:

```
kube-apiserver --authorization-mode=Example,RBAC --other-options --more-options
```

En nuestro caso, minikube por defecto trae configurado "*Node, RBAC*".

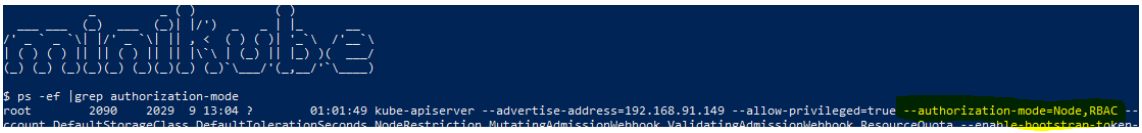

Imagen 44. comprobación authorization-mode

RBAC se basa en tres elementos principales:

- Rol*:* Define las acciones que se pueden realizar dentro del clúster o de un espacio de nombres.
  Existen dos tipos de recursos que representan los roles:
  Clúster-role. Indica que el rol se puede utilizar en todo el clúster.
  Role. Representa elementos que solo se pueden utilizar en un espacio de nombres

- Sujeto: Puede ser usuarios, grupos o cuentas de servicio.

- Asignación: Recurso que recoge la relación entre un rol y un sujeto.
  Existen dos tipos de recursos para realizar las asignaciones.
  RoleBinding. Asigna permisos a nivel de espacio de nombres.
  ClusterRoleBinding: Asigna permisos a nivel de clúster.

La asignación de permisos a los usuarios se realizaría del siguiente modo:

Creamos un fichero de manifiesto que contiene la definición del rol y de asignación de permisos al usuario.

```yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: pods-reader
  namespace: usu1
rules:
  - verbs:
    - "get"
    - "watch"
    - "list"
    apiGroups:
      - ""
    resources:
      - "pods"
```

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: usu1-pods-reader
  namespace: usu1
subjects:
  - kind: User
    name: usu1
    apiGroup: rbac.authorization.k8s.io
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: pods-reader
```

Imagen 45. Ficheros yaml Role-Rolebinding

El manifiesto de para la creación del rol, incluye una sección *Rules*, en la cual se definen los permisos (acciones permitidas, grupos y recursos para los que se permiten las acciones).

El manifiesto de asociación incluye una sección sobre el sujeto sobre el que se realiza la asignación y otra en la que se referencian los datos del rol que se asociar al sujeto.

Comprobamos como realmente la asignación de permisos nos permite el acceso a obtenerlos pods y no para realizar otras acciones o acceder a otros recursos.
Para ello tenemos que acceder al contexto de administrador (minikube) que posee todos los permisos, y ejecutar los ficheros de manifiesto de Role y de RoleBinding.

```
PS C:\Users\Admin\Desktop\Kubernetes> kubectl apply -f .\usu1_pods_reader.yaml
Error from server (Forbidden): error when retrieving current configuration of:
Resource: "rbac.authorization.k8s.io/v1, Resource=roles", GroupVersionKind: "rbac.authorization.k8s.io/v1, Kind=Role"
Name: "pods-reader", Namespace: "usu1"
from server for: ".\usu1_pods_reader.yaml": roles.rbac.authorization.k8s.io "pods-reader" is forbidden: User "usu1" cannot get resource "roles" in API group "rbac.authorization.k8s.io" in the namespace "usu1"
PS C:\Users\Admin\Desktop\Kubernetes> kubectl config use-context minikube
Switched to context "minikube".
PS C:\Users\Admin\Desktop\Kubernetes> kubectl apply -f .\usu1_pods_reader.yaml
role.rbac.authorization.k8s.io/pods-reader unchanged
PS C:\Users\Admin\Desktop\Kubernetes> kubectl apply -f .\usu1_rolebinding_reader.yaml
rolebinding.rbac.authorization.k8s.io/usu1-pods-reader unchanged
PS C:\Users\Admin\Desktop\Kubernetes> kubectl config use-context usu1
Switched to context "usu1".
PS C:\Users\Admin\Desktop\Kubernetes> kubectl get po
No resources found in usu1 namespace.
PS C:\Users\Admin\Desktop\Kubernetes> kubectl config current-context
usu1
PS C:\Users\Admin\Desktop\Kubernetes> kubectl get events
Error from server (Forbidden): events is forbidden: User "usu1" cannot iist resource "events" in API group "" in the namespace "usu1"
PS C:\Users\Admin\Desktop\Kubernetes>
```

Imagen 46. creación de permisos de usuario.

Se comprueba que se pueden listar pods en el espacio de nombres usu1 y que no se pueden obtener los eventos.

Esta configuración es un ejemplo básico, para asignar permisos a los sujetos.
Para cumplir con una buena política de mínimos privilegios, es importante recordar que:

- Solo deben de asignarse los permisos mínimos necesarios para la realización de la tarea/función que se vaya a desarrollar.
- Uso de *RoleBinding* que solo asigna permisos en un espacio de nombres, en lugar de *ClusterRoleBinding*.

- Evitar el uso de comodines (**“*”**), y es muy poco recomendable utilizarlos para dar permiso a todos los recursos, debido a que Kubernetes es un sistema extensible y esta regla se aplicará también a los futuros elementos que existan.
- Evitar el uso del rol **cluster-admin** debido a que concede acceso a todo el clúster (está asociada por ClusterRole al grupo *system:master)*

7.3. **Uso de credenciales en texto plano en los ficheros de configuración.**

Los ficheros de configuración son el lugar más habitual para almacenar ya que las credenciales son consideradas como una parte la configuración de las aplicaciones (claves de aplicación, base de datos, SSH).

El desarrollo de aplicaciones para la nube ha supuesto un cambio en el modelo de seguridad, ya que en un sistema de desarrollo tradicional los entornos eran aislados, y normalmente el acceso a los sistemas de producción eran restringidos a pocos usuarios.

El uso de DevOps ha originado que el grupo de desarrollo tenga acceso a los entornos de producción, y por consiguiente podría obtener los datos de acceso existentes en los ficheros de configuración. Un usuario comprometido, podría ocasionar grandes perjuicios de seguridad a toda la infraestructura.

Las credenciales se consideran datos sensibles lo correcto sería almacenarlas de forma externa a la aplicación. Una solución menos correcta sería el uso de ficheros de configuración encriptados, pero nos originaria el problema del almacenamiento de su clave de desencriptado.

Imagen 47. Captura credenciales texto plano

Kubernetes posee un objeto en su modelo que ha sido diseñado para almacenar datos confidenciales denominado **Secret** (secreto), que permite su almacenamiento encriptado.

Por lo explicado anteriormente utilizaremos Secret para prevenir el uso de credenciales en texto plano en los ficheros de configuración.

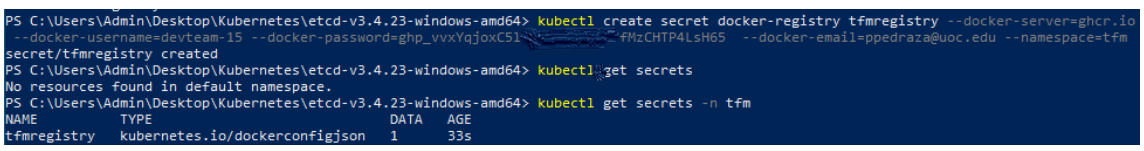

Imagen 48. Creación de secreto

Siguiendo el principio de mínimo privilegio, la declaración de secretos debe realizarse para el espacio de nombres se vayan a utilizar.

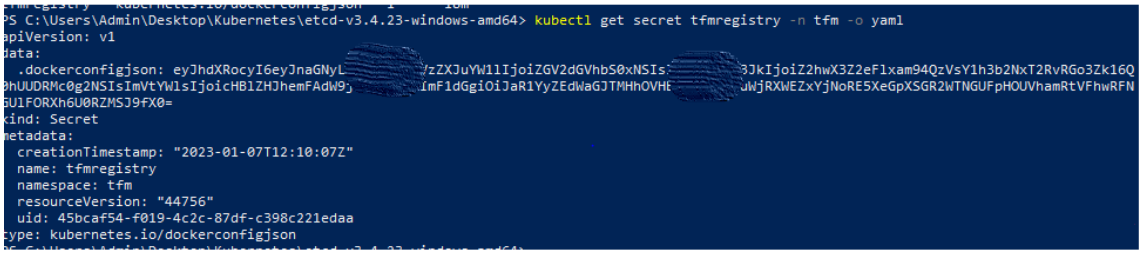

Imagen 49. Contenido de secreto

Podemos comprobar que el contenido del secreto se encuentra codificado (base64), pero no encriptado.

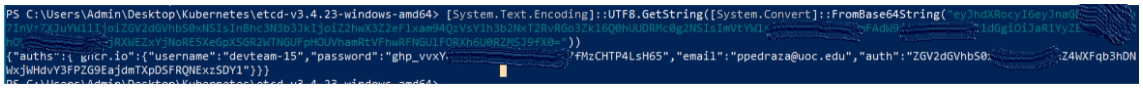

Imagen 50. Secreto decodificado

Para hacer uso de los secretos, se ha de agregar al fichero de manifiesto, las siguientes especificaciones:

```
spec:
    ……
    imagePullSecrets:
     - name : tfmregistry
```

7.4. Usar almacén de secretos administrados

Es importante que las organizaciones se conciencien en la necesidad de realizar la gestión de los secretos(almacenamiento, auditoria, rotación).

La gestión de secretos ha de tener en cuenta varios aspectos para que sea exitosa:

- Alta disponibilidad. Acceso rápido y confiable.
- Centralización y estandarización a todos los niveles organizacionales.
- Control de acceso. Los usuarios solo deben poder acceder a los recursos necesarios para realizar su función(principio de privilegio minino)
- Automatización de la gestión de secretos.
- Auditoria que revise la fortaleza del sistema de secretos.
- Uso de TLS para prevenir él envió de texto sin formato.
- Uso de políticas de requisitos mínimos para secretos(longitud, patrón.)
- Metadatos que faciliten la migración a otro sistema en caso de ser necesario.

Como hemos comentado en el punto [6.3], Kubernetes dispone de Secret para almacenar los datos sensibles, pero que lo más recomendable es la utilización de un almacén externo para guardar esos datos.

La resolución de este problema se mueve a pila tareas trabajos futuros por falta de tiempo.

7.5. Habilitar el acceso Just-In-Time (JIT) al servidor API

Este problema de seguridad se produce cuando una tarea requiere de permisos elevados temporalmente y hay que proveer los permisos necesarios para que se realicen.
Al ser la asignación de permisos un proceso manual en Kubernetes, puede ocurrir que un administrador olvide revocar los permisos concedidos y existan cuentas con permisos elevados sin necesitarlos.

Podríamos considerar algunas soluciones para prevenir este problema como crear cuentas temporales con los permisos necesarios para la ejecución de la tarea (Establecemos caducidad próxima para los certificados).

Los proveedores de clúster en la nube disponen de servicios de IAM integrados con el servicio de Kubernetes y que permiten una mejor gestión de los permisos de usuario (En Azure encontramos AD (Active Directory) que puede utilizarse y se encuentra activada RBAC en el clúster de Kubernetes.)

7.6. Controlador de admisión (NodeRestriction)

Los controladores de admisión son un complemento de kubernetes, consisten en fragmentos de código que compilan con el kube-apiserver y que permiten ampliar las opciones de seguridad.

Una vez que una solicitud ha sido autenticada y autorizada, el controlador de admisión se encarga de interceptar la solicitud que llega al servidor API y ejecuta el código que contiene en la solicitud, antes de que esta se aplique. Si la ejecución cumple con los requisitos, el controlador autoriza la petición, de lo contrario la deniega.

El uso de ***NodeRestriction*** garantiza que Kubelet solo pueda modificar los objetos Nodo y Pod en la API, en el nodo en el que se encuentra kubelet. Esto evita que, si un nodo es atacado, pueda comprometer al resto de nodos del clúster.

Su activación en el clúster se realiza activando en el manifiesto de API Server el atributo `--enable-admission-plugins=NodeRestriction` o agregándolo como parámetro cuando se inicia Minikube.

```
PS C:\Users\Admin\Desktop\Kubernetes> minikube start --enable-admission-plugins=NodeRestriction
```

Imagen 51. Activación de NodeRestriction

Comprobamos mediante el manifiesto, que nuestro nodo ya tiene activada esta característica para prevenir que se pueda acceder desde otros nodos al nodo master.

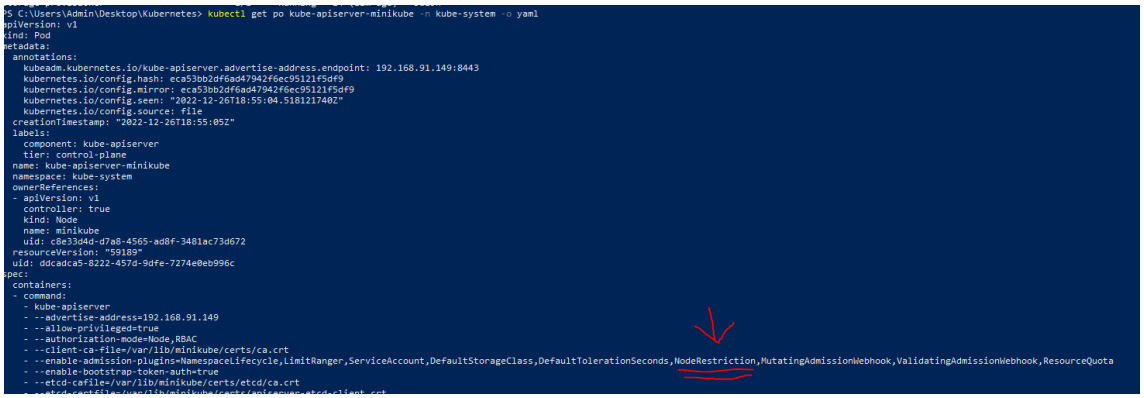

Imagen 52. Comprobación NodeRestriction activado

7.7. Uso de contenedores permisivos.

Las políticas de seguridad de pod (PSP) fueron introducidas en kubernetes para limitar los daños que podían ocasionar los pods maliciosos en el clúster. PSP funciona de manera similar a los controladores de admisión, pero presenta algunas deficiencias importantes:

- Modelo de autorización defectuoso.
- Son difíciles de implementar.
- Api inconsistente.
- Requieren de conocimientos de seguridad ya que se basan en la seguridad de Linux.

Estos problemas originan el cambio, y las políticas de seguridad pod son sustituidas por **Pod Security Admision** (PSA), que define tres políticas de seguridad (Políticas de Seguridad Standard - PSS):

- *Privileged*: política sin restricciones.
- *BaseLine*: política de restricción mínima que evita las técnicas escalado de privilegios conocidas .
- *Restricted*: Es más restrictivo que la política BaseLine, aplicando las mejores prácticas de seguridad.

PSA permite que usuarios no expertos, puedan aplicar políticas seguridad básica a los pods, sin poseer grandes conocimientos en seguridad.

Existen tres modos en el que se puede ejecutar PSA:

- *enforce*: Si no se cumple con la política se rechaza la solicitud del pod.
- *audit* : La infracción de la política se almacena en el registro de auditoría, pero se permite la solicitud del pod.
- *warn*: La infracción de la política muestra una advertencia, pero se permite la solicitud del pod

Establecemos la política con el siguiente comando:

- Privileged :

kubectl label --dry-run=server --overwrite ns --all pod-security.kubernetes.io/enforce=privileged

- BaseLine:

kubectl label --dry-run=server --overwrite ns --all pod-security.kubernetes.io/enforce=baseline

- Restricted:

kubectl label --dry-run=server --overwrite ns --all pod-security.kubernetes.io/enforce=restricted

Imagen 53. Ejecución de PSA

Una vez que aplicamos PSA obtenemos información de los Pods que no cumplen con la nueva política aplicada.

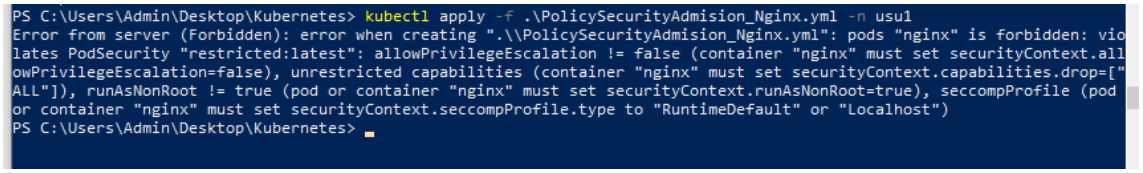

Imagen 54. Ejecución de deployment con PSA

Con PSA activado los controladores, podemos evitar la ejecución de contenedores permisivos (contenedores privilegiados, contendores con volúmenes sensibles o con excesivas capacidades.)

7.8. Restringir comandos ejecutivos en pods.

El contexto de seguridad de kubernetes (Security Context) es una herramienta que permite asignar permisos de seguridad en el servidor host a los recursos del clúster (Pods y contenedores)

El contexto de seguridad podría asimilarse a RBAC en su funcionamiento, pero existen algunas diferencias entre ellos:

- RBAC tiene mayor alcance en cuanto a los recursos a los que se puede aplicar (Pods, nodos, incluso al clúster.)
- **RBAC** otorga los permisos basándose en los verbos, mientras que el contexto de seguridad solo permite asignar tipos específicos de capacidades predefinidas.
- El **contexto de seguridad** es más extensible ya que puede integrarse con herramientas externas. RBAC por el contrario no puede utilizar herramientas externas para definir políticas.
  Un *contexto* interno se configuraría agregando al manifiesto de creación del pod el siguiente fragmento de código:

```
spec:
securityContext :
        runAsUser : 1000
        runAsGroup : 3000
```

Esta configuración de Pod indica a Kubernetes que el usuario del Pod es el 1000 y su grupo 2000. Para este usuario y grupo existirá una configuración previa de permisos en el subsistema de Linux.
Los contextos de seguridad al gestionar permisos y privilegios basados en Linux, por lo que en la ejecución de pruebas que hemos realizado, el contexto de seguridad no sería efectivos al estar realizada en un sistema operativo Windows.

Para evitar el uso de comandos ejecutivos en los Pods con RBAC, sería suficiente con no incluir el recurso **pods/exec** en los manifiestos de creación . Es importante recordar que tampoco deben utilizarse los comodines (“*****”) para la autorización de recursos, ya que estaríamos autorizando todos los recursos, incluido (pod/exec).

```
apiGroups:
  - ''
resources:
  - pods/exec
```

7.9. Restricción de acceso al servidor API

Kubernetes no dispone de componentes nativos para proteger el tráfico de entrada/salida entre clúster internos y redes externas que permita la protección del Api server.

El uso de un firewall de aplicaciones permite controlar el tráfico de red con el exterior de clúster y nos aporta los siguientes beneficios:

- Evita fugas de datos y malware.
- Permite el filtrado de IP y URL.
- Bloqueo de puertos abiertos.

Kubernetes sí que dispone de complementos de interfaz de contendores (CNI) que añaden o mejoran las funciones del interfaz de red nativo de Kubernetes.

Existen diversas alternativas como Calico, Flannel, Cillium, Canal y según las necesidades podamos seleccionar el más adecuado para nuestra implementación. Aprovecharemos la necesidad de la instalación de un firewall de aplicaciones, para cambiar el CNI de que facilita Minikube (kindnet), por uno con firewall integrado.

En nuestro proyecto nos decantaremos por la instalación de Cilium.

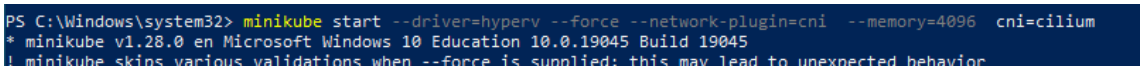

Imagen 55. Inicio de clúster con Cilium

Cilium es una solución de código abierto que proporciona control, seguridad y observación de redes, en entornos nativos de la nube. Funciona a nivel de capa 3 de red y enruta los paquetes mediante el protocolo BGP (Border Gateway Protocol) entre los nodos.

BGP es un protocolo de enrutamiento dinámico, de aprendizaje automático debido a que las tablas de rutas se reenvían a la red cuando se produce un cambio en ellas.

Con respecto a minikube, *cilium* es compatible con políticas de red, y proporciona un conjunto de capacidades más completo que la política de red que contiene la versión completa de Kubernetes.

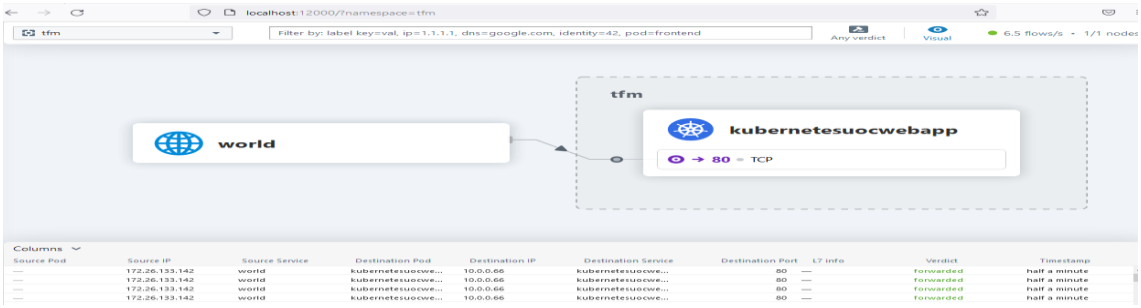

Imagen 56.Monitor de red hubble

Hubble es una herramienta puede instalarse junto a Cilium que proporciona visibilidad sobre la infraestructura de red, comportamiento de servicios, y sobre la comunicación en el clúster.

7.10. Restringir el acceso de contenedores en tiempo de ejecución al entorno Host con LSM (Linux Security Modules)

Linux security modules proporciona mecanismo seguro anclaje a las nuevas extensiones del kernel del Linux.

eBPF(extended, Berkeley Packet Filter) es una tecnología nueva, que utiliza LSM para ejecutar pequeños programas en el kernel de Linux. La ejecución de estos programas en el kernel de Linux permite que se puedan agregar nuevas capacidades al núcleo sistema operativo. Algunos complementos de Kubernetes utilizan eBPF para gestionar el tráfico de red, en programas de rastreo, para realizar el seguimiento de conexiones TCP, estadísticas…

Cilium se ha desarrollado sobre tecnología **eBPF**, lo cual permite obtener datos de aspectos como el tráfico en la red, el sistema de archivos, los procesos y llamadas al sistema y compararlos con los datos de ataques conocidos, lo que permite detectar los patrones de ataques y emitir alertas.

AppArmor es un módulo de seguridad del kernel de Linux. Su funcionalidad permite limitar los recursos que pueden utilizar los programas, disminuyendo de este modo la superficie de ataque. Su funcionamiento se basa en la creación de un perfil con los recursos que puede utilizar un contenedor. Cuando no se cumple con alguna de las características recogidas en el perfil, AppArmor informa de la infracción cometida. Cuanto más completa sea la información recogida por el perfil, mayor será la seguridad que aporte AppArmor.

SELinux utiliza LSM para proporcionar los recursos para extender la política de seguridad del Kernel de Linux, permite la implementación de políticas de control

de acceso obligatorio (MAC) y basadas en roles (RBAC). Su configuración se realiza en Kubernetes a través del contexto de seguridad.

```
securityContext:
  seLinuxOptions:
    level: "s0:c123,c456" #Sensibilidad y categoría.
```

SELinux y AppArmor protegen al host agregando una capa aislamiento con respecto al contenedor.

7.11. Segmentación de red

Por defecto en Kubernetes no existen restricciones para el tráfico de red dentro del clúster y esto puede ocasionar problemas de seguridad si algún nodo es accedido por algún atacante.

Para realizar la segmentación de red disponemos de las **Políticas de Red** (Network Policies). Mediante ellas podemos limitar que solo ciertos pod, namespaces o rangos de IP puedan comunicarse entre sí.

Un ejemplo de política de red es el siguiente:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-allow-tfm-namespace
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          purpose: tfm
```

Imagen 57. Política de red

Segmentación de red.
Política de red basada en etiquetas.

La resolución de este problema se mueve a pila tareas trabajos futuros por falta de tiempo.

7.12. Prevención de envenenamiento por ARP

El protocolo de resolución de direcciones (ARP, Address Resolution Protocol) es utilizado por las capas 2 y 3 en el modelo OSI de red.
Este protocolo se encarga de asociar una dirección MAC (capa 2) a una dirección IP (capa 3).
Un ataque por envenenamiento intenta alterar la tabla donde se almacena la relación de las direcciones MAC y las IP. Este ataque solo es posible si el atacante y el equipo comprometido se encuentran en el mismo segmento de capa 2.

Para prevenir este ataque, será necesaria la utilización de CNI que prevenga este tipo de ataques.

El uso de *Cilium*, previene los ataques de envenenamiento ARP ya que enruta sus paquetes de red en capa 3 por lo que el envenenamiento ARP no es posible con este CNI.

7.13. Limitar el uso de recursos en contenedores.

El uso de recursos hardware por los elementos en un clúster, es un punto de vital importancia para garantizar el correcto funcionamiento del clúster.
Kubernetes, es un entorno dinámico que permite ir variando los elementos hardware de los que dispone, estos no son infinitos.
Los clústeres de kubernetes inicialmente no limitan el uso que los contenedores hacen de los recursos hardware.

Se considera una buena práctica establecer los recursos que los contenedores necesitan y limitar el consumo de recursos, por si existe algún problema en algún contenedor que abuse del uso de recursos hardware (fuga de memoria) y afecte al funcionamiento de otros contenedores. En relación con la seguridad, limitando los recursos que pueden utilizar los contenedores, evita que suframos ataques de denegación de servicio (Denial of Service, DOS).

Los *ataques de denegación de servicios* consisten en inundar el sistema de peticiones. El excesivo número de peticiones produce un aumento en el consumo los recursos disponibles en un host o clúster (red, cpu, memoria), ocasionando que el sistema atacado no funcione correctamente o deje de dar servicio al no disponer de recursos.

Kubernetes permite que se limiten tres tipos de recursos: memoria, cpu y tamaño de páginas de memoria (solo disponible para contenedores Linux.)

En el caso de la limitación de memoria del contenedor, si un proceso consume recursos de los permitidos, kubelet es el encargado de matar dicho proceso, en el caso de proceso con ID=1 tiene un tratamiento especial, y el motor de contenedores emite una señal Out Of Memory (OOF) al nodo, para que se reinicie el contenedor.

La asignación de recursos a un contenedor es un proceso que ha de realizarse con precaución, dado que una asignación por debajo de lo requerido ocasionará un mal funcionamiento de nuestro contenedor, por el contrario, un exceso de recursos ocasionaría un desperdicio ya que estos quedan asignado al contenedor y no pueden ser utilizados por otros contenedores. La cantidad de recursos utilizada por el clúster es la suma de recursos asignados a cada contenedor, por lo que asignar recursos en exceso podría evitar despliegues de nuevos contenedores por falta de recursos.

Asignaremos los recursos y las limitaciones al contenedor mediante el fichero de manifiesto. Se añadirán las siguientes etiquetas:

```
spec:
    containers:
     - name: kubernetesuocwebapp
       imagePullPolicy: Never
       image: ghcr.io/devteam-15/kubernetesuocwebapp
       resources:
                requests:
                        memory: "64Mi"
                        cpu: "250m"
                limits:
                        memory: "128Mi"
                         cpu: "500m"
```

Kubernetes permite la asignación de recurso de manera automática a través de otro objeto llamado LimitRange. Este objeto solo está disponible para asignar recurso a Pods, contenedores y volúmenes persistentes. En este caso utilizaríamos el fichero de manifiesto de este objeto para que los recursos sean asignados.

```
apiVersion: v1
kind: LimitRange
metadata:
  name: Limites_por_defecto
spec:
  limits:
  - default:
      memory: 512Mi
    defaultRequest:
      memory: 256Mi
    type: Container
```

En este punto podríamos incluir las “Cuotas de recursos” (ResourceQuota).Este objeto proporcionan restricciones que limitan el consumo de recursos o la cantidad de objetos que se pueden crear por espacio de nombres.
Las cuotas pueden agruparse en tres grupos:

- Computación (cpu, memoria). Establece los recursos asignados para el espacio de nombres.
- Almacenamiento (request.storage, persistentvolumeclaims, ...) Recursos de almacenamiento asignados.
- Contador de objetos: especifica el número máximo de objetos que puede existir.

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: Limite_de_recursos
spec:
  hard:
    pods: "4"
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Una vez que se ha establecido la cuota de registro. El sistema computa los datos en las creaciones o modificaciones de objetos, si los nuevos valores no cumplen los con los requisitos establecidos, se deniega la solicitud con un mensaje.

La comprobación de este problema se mueve a pila tareas trabajos futuros por falta de tiempo.

7.14. Restringir el acceso a archivos y directorios del host

Un volumen es un recurso que nos ofrece Kubernetes para persistir datos en el sistema, como consecuencia de la característica que poseen los contenedores de poder crearse y destruirse y sus imágenes han de permanecer inmutables.
Existen varios tipos de volúmenes, pero nos centraremos en el caso de los volúmenes persistentes.
Los **hostPath** son un tipo de volumen que crea directamente en el sistema de archivos del host. Si el hostPath ha sido creado con permisos de lectura-escritura un atacante podría modificar ficheros, creando enlaces simbólicos (/var/log/pods/<path_to_0.log>) y acceder a todo el sistema de ficheros.

Para prevenir los problemas generados por los hostPath adjuntamos algunas acciones a realizar.

- Evitar el uso cuando no sea necesario.
- El uso de herramientas basadas en LSM (*SElinux, AppArmor*), explicado en el punto 6.11.
- Indicar mediante los ficheros de manifiesto o agregando una política de seguridad pod, aunque este segundo método es menos recomendable, ya que como hemos indicado, agregaría el hostPath a todos los elementos que apliquen la política, en vez de solo a los objetos que los necesiten.

```
spec:
    containers:
    - image: ubuntu
      name: web
      volumeMounts:
      - mountPath: /web
        name: test-volumen
        readOnly: true
    volumes:
    - name: test-volume
      hostPath:
        path: /etc                  #ubicación en el host principal
```

7.15. Restringir el acceso a etcd

Etcd es uno de los componentes más importantes kubernetes, ya que almacena toda la información del Clúster.
Si un atacante accede con permisos de lectura y escritura sobre este componente, tendría control total sobre el clúster, ya que podría modificar la información sobre los componentes y estados .
El permiso de lectura permitirá hacer un escalado.

Es recomendable el uso de TLS para garantizar una comunicación segura, el uso de un firewall que aísle Etcd del resto del entorno y solo permita la comunicación con Api server.

Otra medida sería utilizar una instancia separada para Etcd.

Se ha revisado la política de minikube y por defecto ya ha creado la configuración necesaria para conectar mediante TLS, el servidor API y Etcd.

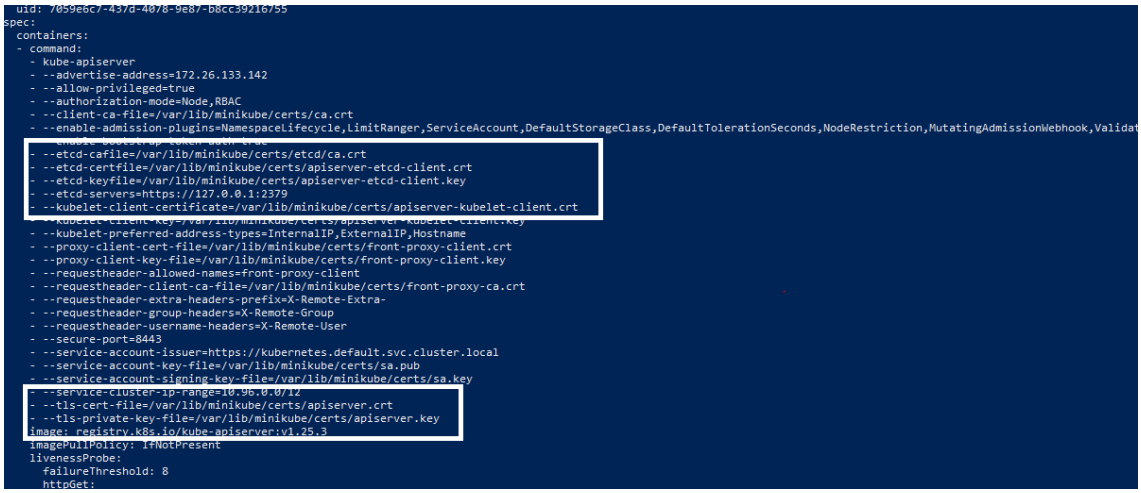

Imagen 58. comprobación TLS activado

7.16. Restringir acceso a manifiestos en pod estáticos controlados por kubelet

El servidor Api se encarga de crear y administrar los Pods a través de Kubelet. Kubernetes posee otro tipo de pods estáticos que son administrados por Kubelet, normalmente son pods utilizado para el funcionamiento interno del clúster. Kubelet mantiene vigilados las pod estáticos y los reinicia en caso de error. También informa a API Server de la existencia de estos pods, los cuales puede listar, pero no puede realizar ninguna operación con ello.

Kubelet para tener conocimiento de los pods estáticos que hay en el nodo utiliza el argumento **--pod-manifest-path**, que contiene la ruta en sistema de archivos del host que contiene los ficheros de manifiesto.
Kubelet dispone de un segundo método para acceder a los ficheros de manifiesto. El acceso lo realiza vía URL y los datos de acceso se encuentran en el argumento **--manifest-url**. El acceso vía URL es considerado como una posible vía de acceso al host, por lo que se recomienda que esta configuración de kubelet se encuentre **deshabilitada**

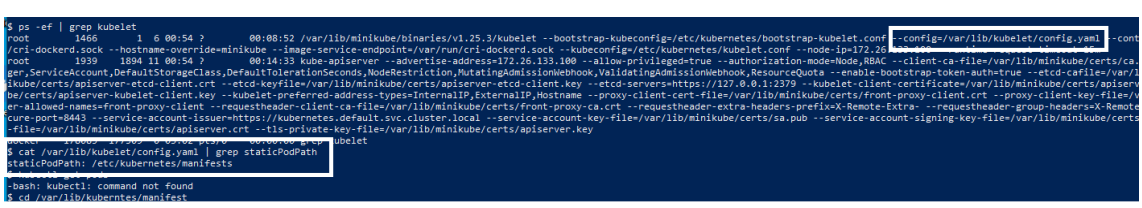

Imagen 59. Ruta staticPodPath

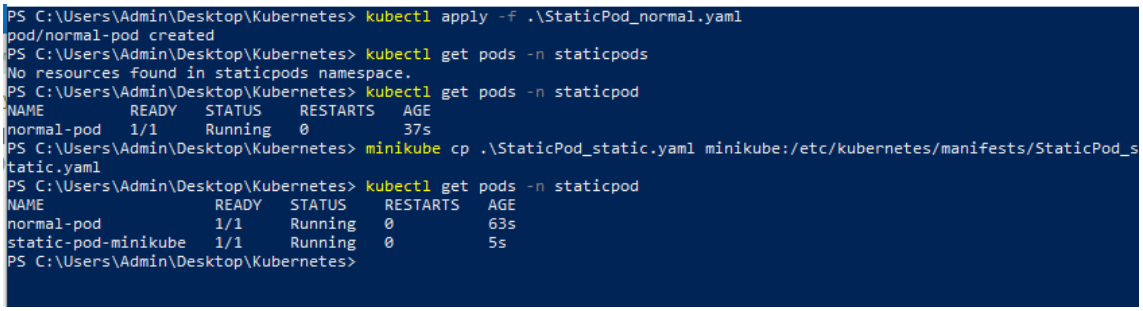

Imagen 60. creación de Pod estático

En el ejemplo se ha desplegado un pod en el namespaces staticpod a través de Api server. En el segundo caso hemos copiado el fichero de manifiesto en la carpeta estática y el pod se ha desplegado utilizando kubelet. Este nodo se recreará automáticamente todas las veces que sea eliminado por Api server, al estar bajo el control de kubelet.

7.17. Deshabilitar auto montaje de la cuenta de servicio

Una *cuenta de servicio* proporciona una identidad a los procesos que son ejecutados por el propio pod. De forma predeterminada, si no se indica una cuenta de servicio especifica durante la creación de un pod, Kubernetes asigna una cuenta de servicio genérica para el espacio de nombres en el que se crea el Pod, lo que origina que, en los diferentes espacios de nombre, no se produzca una separación real de procesos y recursos, dado que comparten la misma cuenta de servicio y desde un pod se podría acceder a recursos utilizados por otro pod en espacio de nombres diferente.

```
apiVersion: v1
  kind: ServiceAccount
  metadata:
    name: tfm-sa
  automountServiceAccountToken: false
```

```
apiVersion: v1
   kind: Pod
   metadata:
    name: kuberneteswebapp
   spec:
    serviceAccountName: tfm-s
    automountServiceAccountToken: false
```

Si existe una cuenta de servicio creada, pero se asigna en el fichero de manifiesto de un pod, predominará la asignación en la creación sobre la asignación automática de kubernetes.

7.18. Estrategia de copia de seguridad de datos.

Etcd es un almacén de datos clave-valor distribuida.
Guarda la información del clúster en todo momento: estado, configuración y metadatos. Debido a la importancia de los elementos que almacena, se debe tener una estrategia copias de seguridad y restauración debido a que si este elemento se corrompe el clúster dejará de funcionar.

En las pruebas realizadas, se ha utilizado un clúster con un solo nodo. Este sería un clúster limitado, recomendable solo pruebas, los entornos de producción

deberían cumplir unos requisitos mínimos para garantizar su buen funcionamiento.
Algunos de estos requisitos son los siguientes:

- El clúster etcd ha de ejecutarse con un numero impar de elemento y mayores a 1 nodo.
- Etcd es un sistema basado en líder, por lo que este ha de enviar señales al resto de nodos para comprobar que siguen funcionando.
- Se recomienda utilizar entornos aislados o máquinas dedicadas para evitar perdida de recursos.

Para realizar la copia de seguridad se utilizará el Pod *etcd-minikube* que nos facilita la instalación de **Minikube** y que se encuentra en el espacio de nombres *kube-system.* Utilizaremos el comando **snapshot save.**

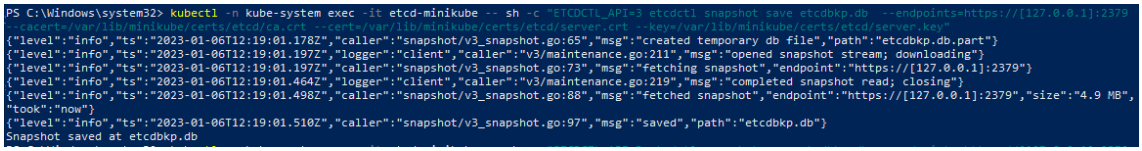

Imagen 61. Snapshot de datos

Para restaurar la copia realizada utilizaremos el comando **snapshot restore**

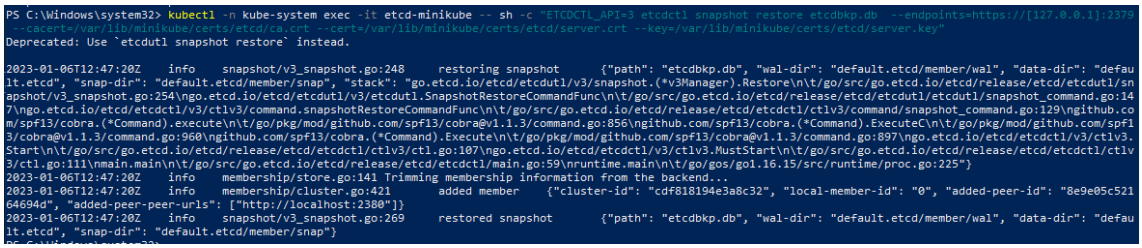

Imagen 62. Snapshot restore

7.19. Usar proveedor de almacenamiento en la nube

Kubernetes dispone de la posibilidad de realizar una copia de seguridad de Etcd en el directorio de archivos del nodo principal, no dispone de una herramienta nativa que nos permita realizar la copia de seguridad fuera del nodo.
La copia de seguridad creada por Kubernetes no incluye el resto de los elementos del clúster como los volúmenes permanentes.

Para la tarea de guardar la información del clúster en la nube, haremos uso de una herramienta externa a kubernetes llamada **Velero.**

Para la instalación del almacenamiento en la nube se ha tenido que crear una cuenta en un proveedor de servicios en la nube, para la prueba se ha utilizado Google Cloud Storage.

- Dentro de **Cloud Storage** se han creado un *proyecto*, *un bucket*, *una cuenta de servicio*, *un rol* y se han asignado los permisos indicados por el creador del software [9.23].

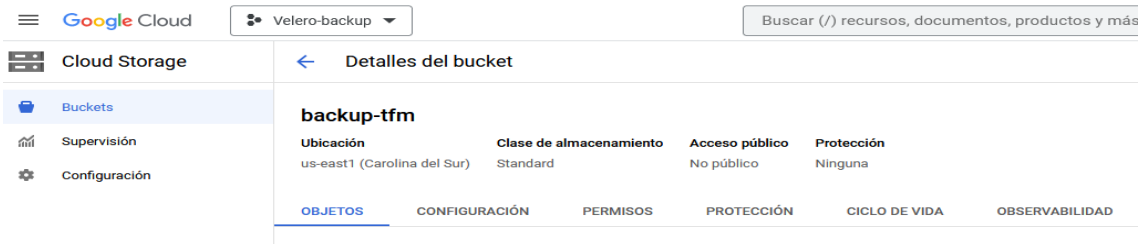

Imagen 63. Detalle bucket Cloud Store

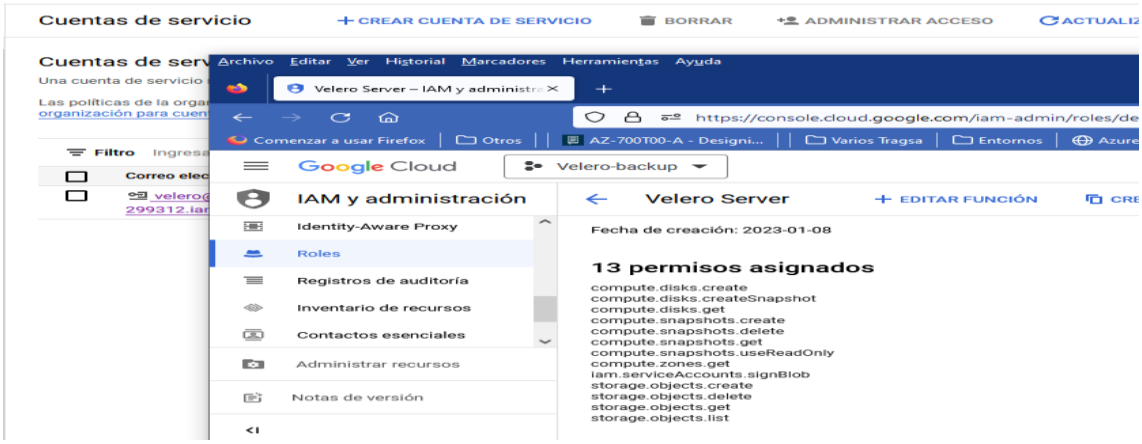

Imagen 64. Asignación de permisos a cuenta de servicio en Cloud Store

- Una vez que hemos configurado la parte del proveedor de servicios, se ha instalado Velero en el host.
  Se configuran los permisos necesarios, y el directorio local para donde se registran las copias antes de subir al almacén remoto.

> choco install velero
> kubectl create secret generic -n velero bsl-credentials --from-file=gcp=</path/to/credentialsfile_serviceaccount>
> velero backup-location create bsl-minikube --provider gcp --bucket backup-tfm --credential=bsl-credentials=gcp

- Se crea el backup para el namespace tfm, con el correspondiente almacenaje de los ficheros en la nube de Google.

```
PS C:\Windows\system32> velero backup create tfm-ns-bkp4 --include-namespaces tfm
Backup request "tfm-ns-bkp4" submitted successfully.
Run `velero backup describe tfm-ns-bkp4` or `velero backup logs tfm-ns-bkp4` for more details.
PS C:\Windows\system32> velero backup get
NAME          STATUS      ERRORS   WARNINGS   CREATED                         EXPIRES   STORAGE LOCATION   SELECTOR
tfm-ns-bkp    Failed      0        0          2023-01-09 00:32:52 +0100 CET   29d       default            <none>
tfm-ns-bkp2   Failed      0        0          2023-01-09 00:43:21 +0100 CET   29d       default            <none>
tfm-ns-bkp3   Failed      0        0          2023-01-09 01:03:06 +0100 CET   29d       default            <none>
tfm-ns-bkp4   Completed   0        0          2023-01-09 01:56:41 +0100 CET   29d       default            <none>
PS C:\Windows\system32>
```

Imagen 65. Realización de backup con Velero

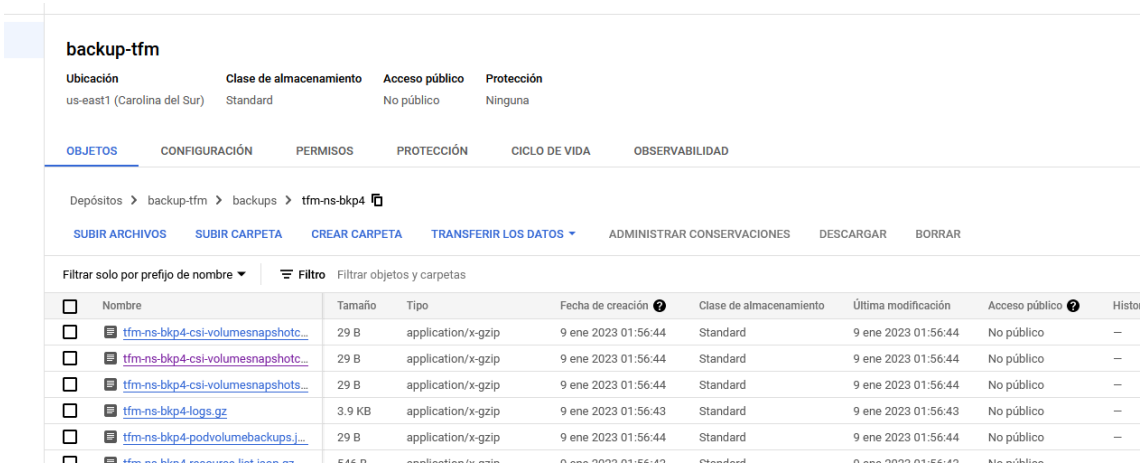

Imagen 66. Backup en Cloud Store

7.20. Entorno CI/CD seguro

Esta tarea de ha desarrollado en su mayor parte en el punto [5] de este trabajo. Las medidas implementadas por mencionar algunas:

- Repositorio privado de código, para evitar accesos no deseados.
- Análisis SAST y DAST
- Almacenamiento de imágenes en repositorio seguro.

7.21. Política de garantía de imagen

La utilización de imágenes obtenidas de repositorios remotos a través un medio no confiable como internet, debería hacernos pensar en la integridad (que el contenido no ha sido manipulado o que proviene de la fuente correcta) de la imagen que se está obteniendo.
Docker para dar confiabilidad a sus imágenes dispone de la funcionalidad **Content Trust,** que permite el uso de firmas digitales para validar los datos enviados o recibidos de repositorios remotos.

Notary es un proyecto que forma parte de Cloud Native Computing Foundation (CNCF). Fue creado con el fin de garantizar la integridad total en las transacciones de archivos utilizando firmas digitales.

Docker utiliza Notary para almacenar los datos de las imágenes firmadas.

La salvaguarda de una imagen en **repositorio de código** seguro de GitHub mediante docker, realiza internamente la firma de la imagen, por lo que no es necesaria la realización de ninguna tarea sobre la imagen ya que esta se almacena firmada digitalmente.

Kubernetes no posee soporte nativo para *Docker Content Trust* por lo que se utilizara la herramienta **Connaisseur.** Esta herramienta es un conector de admisión y se encarga de comprobar que los datos de la solicitud para la creación de un Pod están firmados y que su contenido corresponde con la firma. Si la validación es correcta permite la solicitud, de lo contrario la rechaza.

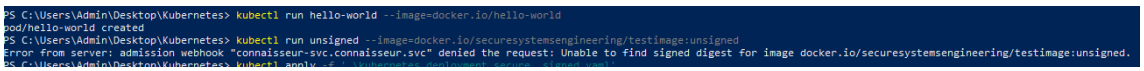

Imagen 67. Connaisseur prueba de funcionamiento.

Connaisseur está preconfigurado para la obtención de imágenes de su repositorio y el de imágenes oficiales de Docker. Deberíamos modificar el fichero **values.yaml** del deployment de Connaisseur para agregar el patrón del repositorio de código deseado, en nuestro caso **ghcr.io/devteam-15/*:***, para que permita validar las imágenes.

```
138
139  ### IMAGE POLICY ###
140  # the image policy ties validators and images together whereby always only the most specific rule (pattern)
141  # is applied. specify if and how images should be validated by which validator via the validator name.
142  policy:
143    - pattern: "*:*"
144    - pattern: "docker.io/library/*:*"
145      validator: dockerhub-basics
146      with:
147        trust_root: docker-official
148    - pattern: "k8s.gcr.io/*:*"
149      validator: allow
150    - pattern: "docker.io/securesystemsengineering/*:*"
151      validator: dockerhub-basics
152      with:
153        trust_root: securesystemsengineering-official
154
```

Imagen 68. Fichero de configuración Connaisseur

8. Conclusiones y trabajos futuros

La realización de este me ha permitido hacer una incursión en el uso de la tecnología de contenedores, de la que era completamente desconocedor.
El contenido que se ha intentado abarcar este trabajo era relativamente largo como para profundizar en muchas de las tareas que se han realizado, y algunas de ellas han sido traspasadas a la lista de trabajos futuros (debido a la falta de tiempo para realizarlas.)
A nivel general, debo reconocer que el resultado obtenido ha sido un poco mejor de lo esperado, ya que una mala planificación, ha conllevado que este proyecto no haya finalizado un poco más completo.

El uso de la metodología nos ha ayudado a que muchas de las tareas de seguridad en la parte de desarrollo se hayan realizado según avanzaba el proyecto, y no se ha tenido que esperar al final para realizarlas.

La conclusión, extraídas de las tareas, es la importancia de la aplicación de medidas la seguridad en la tecnología, y como los sistemas son vulnerables a ataque de hackers. También mencionar la importancia de la formación

multidisciplinar ya que las nuevas tecnologías eliminan la separación existente entre el mundo del desarrollo y la parte de sistemas u operaciones.

Como trabajos futuros a realizar sobre este proyecto, sería importante continuar con las tareas pendientes que no se finalizaron por no disponer de tiempo suficiente como la comprobación del rendimiento del equipo una vez que se establecen los límites de recursos a los PODS o comprobar el funcionamiento de connasieur con repositorios de imágenes diferentes a los predefinidos.
Adquirir conocimientos necesarios para poder realizar otras tareas que quedaron pendientes por no disponer de conocimientos suficientes para realizarlas correctamente, como la aplicación correcta de una segmentación de red.

La realización de este proyecto ha dejado muchas inquietudes abiertas por los que seguir mejorando el trabajo desarrollado, algunos ellos son los siguientes:

- Configuración de reglas de aplicaciones DAST para realización de pruebas
- Con respecto a la integración continua, me ha parecido muy interesante la parte de automatización de tareas, y que no se ha utilizado por desconocimiento (GitActions)
- Aplicar configuraciones de red en Policy Networks.
- Migrar a sistema Operativo de Linux ya que la mayoría de las herramientas externas son para ese sistema operativo.
- Introducción al mundo de Kubernetes en Cloud.

9. Glosario

Concepto	Descripción
API REST	Servidor API es una parte del núcleo de Kubernetes, que permite a los usuarios finales comunicarse con los diferentes elementos del clúster. API REST permite llamar directamente a API sin utilizar la consola mediante un servicio.
ARP	Protocolo de resolución de direcciones.
CoreDNS	Sistema de nombres de dominio (DNS) utilizado por Kubernetes para que los nodos se comuniquen entre sí.
Cronjobs	Programador de tareas
IAM	Gestor de Identidad y acceso (Identity & Access Management)
K8s	Kubernetes
Kube	Kubernetes
kubeconfig	Fichero de configuración para la herramienta de línea de comandos.
kubectl	Herramienta de configuración por la línea de comandos en Kubernetes.
Tiller	Endpoint de extremo a extremo utilizado por el gestor HELM.
TTPS	tácticas, técnicas y procedimientos.
Volumen	Puede considerarse una especie de directorio que puede ser compartido entre los pods y que normalmente utilizamos para persistir información.

10. Bibliografía

Libro:

10.1. Rafael Troncoso, Kubernetes para profesionales | Desde cero al despliegue de aplicaciones seguras y resilientes, Primera edición, OxWORD, Móstoles (Madrid), 2022.

Web:

10.2. https://kubernetes.io/blog/2015/04/borg-predecessor-to-kubernetes/ [10/10/2022]

10.3. https://kubernetes.io/docs/concepts/security/security-checklist/ [11/10/2022]

10.4. https://learn.microsoft.com/es-es/azure/architecture/example-scenario/apps/devops-with-aks [11/10/2022]

10.5. https://minikube.sigs.k8s.io/docs/drivers/ [11/10/2022]

10.6. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/ [07/12/2022]

10.7. https://attack.mitre.org/matrices/enterprise/containers/ [05/11/2022]

10.8. https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html [05/11/2022]

10.9. https://attack.mitre.org/matrices/enterprise/containers/ [05/11/2022]

10.10. https://www.itu.int/es/mediacentre/backgrounders/Pages/icts-to-achieve-the-united-nations-sustainable-development-goals.aspx [11/12/2022]

10.11. https://www.cncf.io/blog/2022/09/22/how-to-think-about-securing-kubernetes-with-waf-and-dos-protection/ [18/12/2022]

10.12. https://github.com/sottlmarek/DevSecOps/blob/master/README.md [05/01/2023]

10.13. https://mudrii.medium.com/kubernetes-local-development-with-minikube-on-hyper-v-windows-10-75f52ad1ed42 [05/01/2023]

10.14. https://rkhal101.github.io/_posts/WAVS/ZAP/zap_browser_setup [05/01/2023]

10.15. https://kubernetes-security.de/en/admission_controler_noderestriction/ [05/01/2023]

10.16. https://techblog.cisco.com/blog/dark-side-of-kubernetes-admission-webhooks [05/01/2023]

10.17. https://cilium.io/ [05/01/2023]

10.18. https://ebpf.io/ [05/01/2023]

10.19. https://hackersvanguard.com/abuse-kubernetes-with-the-automountserviceaccounttoken/ [05/01/2023]

10.20. https://cheatsheetseries.owasp.org/cheatsheets/Secrets_Management_Cheat_Sheet.html [05/01/2023]

10.21. https://jagadesh4java.blogspot.com/2022/01/kubernetes-static-pods.html [05/01/2023]

10.22. https://github.com/ahmetb/kubernetes-network-policy-recipes/blob/master/06-allow-traffic-from-a-namespace.md [05/01/2023]

10.23. https://github.com/vmware-tanzu/velero-plugin-for-gcp [05/01/2023]

11. Anexos

A.1. Fichero YAML para la creación de un deployment.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kubernetesuocwebapp
  labels:
    app: kubernetesuocwebapp
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kubernetesuocwebapp
  template:
    metadata:
      labels:
        app: kubernetesuocwebapp
    spec:
      containers:
      - name: kubernetesuocwebapp
        imagePullPolicy: Always
        image: ghcr.io/devteam-15/kubernetesuocwebapp
        ports:
        - containerPort: 8090
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
          limits:
            cpu: 200m
            memory: 200Mi
      imagePullSecrets:
      - name : tfmregistry
```