



kubernetes



# Infraestructura IT integrada en Kubernetes



SAL\_kubernITes

(SAL\_kubernITes – SAL\_kITs)

- Autor: Saúl Altoubah León
- Grupo: ASIA/B
- Curso: 2021 - 2022





Esta obra está sujeta a una licencia de [Reconocimiento-NoComercial-SinObraDerivada 4.0 Internacional \(CC BY-NC-ND 4.0\)](https://creativecommons.org/licenses/by-nc-nd/4.0/)

Copyright © 2021 **Saúl Altoubah León**

Reservados todos los derechos. Está prohibido la reproducción total o parcial de esta obra por cualquier medio o procedimiento, comprendidos la impresión, la reprografía, el microfilme, el tratamiento informático o cualquier otro sistema, así como la distribución de ejemplares mediante alquiler y préstamo, sin la autorización escrita del autor o de los límites que autorice la Ley de Propiedad Intelectual.

### Centro educativo

| Código   | Centro                       | Concello | Año académico |
|----------|------------------------------|----------|---------------|
| 15005397 | I.E.S. Fernando Wirtz Suárez | A Coruña | 2021/2022     |

### Ciclo formativo

| Código de la familia profesional | Familia profesional          | Código de ciclo formativo | Grado  | Régimen |
|----------------------------------|------------------------------|---------------------------|--|---------|
| FP16                             | Informática y Comunicaciones | CSIFC01                   | Administración de Sistemas Informáticos en Red | Adultos |

### Módulo profesional y unidades formativas de menor duración (\*)

| Código MP/UF | Nombre   |
|--------------|--|
| MP0373       | Proyecto de Administración de Sistemas Informáticos En Red<br>Equivalencia en créditos ECTS: 5<br>Código: MP0379<br>Duración: 26 horas |

### Profesorado responsable

| Tutora             | Equipo docente  |
|--------------------|---|
| Calvo Castro, Ruth | Aira Martín, Sandra<br>Corbelle Mejuto, Manuel<br>Martínez Mejuto, Teresita<br>Pérez Mariño, Sofía<br>Romarís Caamaño, Nuria<br>Vázquez Abelleira, Arturo |



# Índice

|  |    |
|--|----|
| • <b>1. Introducción</b> .....         | 2  |
| • <b>2. Objetivo</b> .....             | 6  |
| • <b>3. Descripción</b> .....          | 7  |
| • 3.1. Alcance .....                   | 7  |
| • 3.2. Medios a utilizar .....         | 8  |
| • <b>4. Planificación</b> .....        | 18 |
| • <b>5. Ejecución</b> .....            | 19 |
| • 5.1. Diseño .....                    | 21 |
| • 5.2. Implementación.....             | 26 |
| • 5.3. Administración.....             | 28 |
| • 5.3.1. Monitorización .....          | 28 |
| • 5.3.2. Seguridad .....               | 38 |
| • 5.3.3. Almacenamiento de datos ..... | 50 |
| • 5.3.4. Redes .....                   | 52 |
| • <b>6. Presupuesto</b> .....          | 54 |
| • <b>Conclusiones finales</b> .....    | 57 |
| • <b>Bibliografía</b> .....            | 58 |



# Índice de figuras

|   |    |
|---|----|
| • Figura 1 – “Servidor de <i>PRIMERGY</i> cumple 25 años” .....                         | 2  |
| • Figura 2 – “Es el fin de la Infraestructura como servicio” .....                      | 3  |
| • Figura 3 – “Dashboard de <i>Netdata</i> ” .....                                       | 10 |
| • Figura 4 – “GUI de <i>Apache Guacamole</i> ” .....                                    | 13 |
| • Figura 5 – “Interfaz gráfica de <i>Lens</i> ” .....                                   | 14 |
| • Figura 6 – “Esquema de conexión VPN de <i>OpenVPN</i> de ejemplo” .....               | 15 |
| • Figura 7 – “Plano de control personal de <i>Notion</i> ” .....                        | 16 |
| • Figura 8 – “Plano de control personal de <i>GitHub</i> ” .....                        | 17 |
| • Figura 9 – “Esquema resumido de los dispositivos físicos en la infraestructura” ..... | 21 |
| • Figura 10 – “Arquitectura de <i>Kubernetes</i> ” .....                                | 23 |
| • Figura 11 – “Arquitectura de <i>Minikube</i> ” .....                                  | 23 |
| • Figura 12 – “Esquema general de contenedores con sus respectivas herramientas” .....  | 24 |
| • Figura 13 – “Esquema de monitoreo de la infraestructura TI general” .....             | 29 |
| • Figura 14 – “Dashboard de <i>Kubernetes</i> ” .....                                   | 31 |
| • Figura 15 – “Interfaz gráfica del clúster en <i>Lens</i> ” .....                      | 32 |

# Índice de figuras

|   |    |
|---|----|
| • Figura 16 – “Helm Chart de Netdata en Lens” .....   | 32 |
| • Figura 17 – “Interfaz local de monitorización de Netdata” ...   | 33 |
| • Figura 18 – “Netdata Cloud, pestaña de nodos conectados” .  | 34 |
| • Figura 19 – “Netdata Cloud, panel de supervisión general del servidor SAL_kubernITes” .....   | 34 |
| • Figura 20 – “Netdata Cloud, panel de alertas” .....   | 35 |
| • Figura 21 – “Advertencia de Netdata Cloud recibida por email” .....   | 35 |
| • Figura 22 – “Terminal de SAL_kubernITes, resultado del registro del pod ‘counter’” .....  | 37 |
| • Figura 23 – “Terminal en SAL_kubernITes, ejecución del comando “cat” empleando el filtro “grep” que muestra los usuarios con UID 0” ..... | 35 |
| • Figura 24 – “Terminal de SAL_kubernITes, resultado al ejecutar el comando “ulimit -l -u -v -s -m”” .....                                  | 45 |
| • Figura 25 – “Ejemplo de creación y conversión docker-compose.yaml a “deployments” en .yaml para servidor OpenVPN en Minikube” .....       | 49 |

# Índice de figuras

|  |    |
|--|----|
| • Figura 26 – “Planificación de redes en <i>Kubernetes</i> entre dos<br><i>nodos</i> ” ..... | 53 |
|--|----|



# Índice de tablas

|   |    |
|---|----|
| • Tabla 1 – “Requerimientos mínimos para VM servidor <i>SAL_KubernITes</i> ” .....        | 24 |
| • Tabla 2 – “Requerimientos mínimos para VM cliente <i>SAL_kITs</i> ” .....               | 24 |
| • Tabla 3 – “Lista de comandos para la instalación de CLI de <i>Docker</i> ” .....        | 25 |
| • Tabla 4 – “Lista de comandos para la instalación del clúster de <i>Minikube</i> ” ..... | 26 |
| • Tabla 5 – “Directorios para auditar” .....  | 42 |
| • Tabla 6 – “Costes de recursos de hardware” .....  | 54 |
| • Tabla 7 – “Costes de recursos de software” .....  | 55 |
| • Tabla 8 – “Costes totales” .....  | 56 |



## ◦ 1. Introducción

Hace una década, ejecutar máquinas virtuales (*VM, Virtual Machines*) en la nube era algo de vanguardia. Esto hizo que la migración a la nube fuera bastante simple: las empresas solo podían cambiar las VM que ya ejecutaban en los servidores de sus instalaciones a los servidores de un proveedor de infraestructura como servicio (*IaaS, Infrastructure As A Service*). Las empresas, al liberarse de la carga del mantenimiento de los servidores físicos, ganaron flexibilidad y redujeron los costos, pero en la actualidad, nadie desarrolla nuevas aplicaciones basadas en las VMs. En cambio, recurren a un modelo que es más rentable, exige menos mantenimiento y es más escalable de lo que las VMs podían esperar ser: **la contenerización**. Este modelo, no las VMs, representan el futuro de la computación.

Los antecedentes recientes de la infraestructura TI son la historia de abstracción. En los 90's, se ejecutaban las aplicaciones en hardware en bastidores físicos:



Figura 1 - Servidor PRIMERGY de Fujitsu cumple 25 años

Cada nuevo desarrollo ha abstraído cada vez más aplicaciones del hardware. Por este motivo, la parte de infraestructura que las empresas tenían que manejar se redujo muchísimo, pero el modelo *IaaS* de ejecutar las VMs en la nube no es el último paso en esa progresión. Las VMs tienen algunas desventajas significativas:

- El hecho de que cada VM ejecute sistemas operativos múltiples inevitablemente crea ineficiencias. Incluso cuando se escala y dimensionan de manera adecuada, lo que no es seguro, las VM dejan mucha capacidad sin usar en los servidores.
- Con las VMs, las empresas siguen siendo responsables de llevar a cabo los molestos ejercicios de operaciones, como la recuperación de desastres, la alta disponibilidad y el escalado, así como la aplicación de parches y la seguridad.
- Las VMs no son muy flexibles y funcionan de manera diferente en distintos hiperescaladores, por lo que una VM impulsada por *Microsoft Azure* no se puede migrar a *AWS* ni a *Google Cloud*.



Figura 2 - Es el fin de la infraestructura como servicio

Las empresas que todavía tratan de migrar a la nube mediante migración de sus VM deberían pensárselo dos veces. Comprometerse ahora con un modelo ineficiente retrasará el progreso en el futuro. En cambio, las empresas deberían recurrir a la contenerización, aunque deban hacerse cambios significativos en sus

procesos. Los contenedores son el próximo paso en la tendencia de la abstracción. Varios contenedores pueden ejecutarse en un solo núcleo del sistema operativo, lo que significa que usan recursos de manera más eficiente que las VM. De hecho, en la infraestructura necesaria para una VM, podría ejecutarse una docena de contenedores. Sin embargo, los contenedores tienen sus desventajas. Si bien tienen más espacio eficiente que las VM, ocupan capacidad de infraestructura cuando están inactivos y generan costos innecesarios.

La transición a la contenerización requiere cambios importantes en los procesos y en la estructura de sus equipos de TI, así como elecciones justificadas sobre cómo llevarla a cabo.

El traspaso a la infraestructura de TI moderna es tanto una transformación de personas y procesos como de tecnología. La administración de la infraestructura de TI tradicional se basa, en gran medida, en las soluciones manuales de apuntar y hacer clic. Por el contrario, la administración de la infraestructura contenerizada es más como la ingeniería del software: los equipos de TI usan un código para describir el resultado final que quieren y sistemas automáticos para llevar a cabo la implementación.

Para aprovechar al máximo la flexibilidad y la eficacia que logra la infraestructura moderna, los equipos de TI deben pasar a lo que se conoce como una orientación de *DevOps*, que aporta prácticas de desarrollo de software ágiles a la administración de infraestructura. Por ejemplo, los equipos de TI de las empresas tradicionales tienden a estar aislados por función, pero *DevOps* toma un enfoque más ágil donde un equipo posee la totalidad de la aplicación. Es necesario adaptar esta nueva manera de trabajar para lograr el éxito de la transición de su infraestructura.

Es conveniente evitar soluciones de terceros de propiedad privada ya que muchas pilas de software de este estilo afirman que facilitan la transición a los contenedores. Sin embargo, al final, estas “abstracciones” solo pueden agregar pasos y costos adicionales; si bien pueden simplificar la transición inicial, a medida que se perfecciona, con el tiempo, tendrá necesidades que, probablemente, no puedan manejar. En cambio, nosotros deberíamos eliminar al intermediario desde el principio y utilizar las soluciones de *open source* (código abierto) con las comunidades activas, como *Docker* y *Kubernetes*. Es posible que la curva de aprendizaje sea más pronunciada, pero nos ahorrará tiempo a nuestros equipos de TI en un futuro.

No se tiene que hacer el traslado a los contenedores de una sola vez. Eso sería sumamente complejo. En cambio, podemos trasladar algunos servicios a los contenedores poco a poco hasta estar completamente contenerizados. Esta estrategia de adopción progresiva se conoce como “método estrangulador”, debido a que el código nuevo lentamente “estrangula” o reemplaza al código viejo. Esta estrategia lenta y constante también les da a nuestros equipos de TI tiempo para adaptarse a las nuevas maneras de trabajar.

Y en relación a este proyecto, ya que se está utilizando esta tecnología, ¿se podría decir que es el comienzo de un nuevo concepto de servicio cuyo nombre sería *CIaaS (Containerized Infrastructure As A Service)*? ¿*KaaS (Kubernetes As A Service)*? ¿*KIaaS (Kubernetes Infrastructure As A Service)*?...





## • 2. Objetivo

El presente documento contiene una guía para ejecutar cumplir con objetivo principal de este proyecto se basará en el diseño, implementación y administración de una infraestructura general de TI que abarcará todos los requerimientos necesarios para su utilización: **monitorización, seguridad, almacenamiento de datos, acceso remoto**, etc. Integrado en el sistema de contenedores mediante *Docker*, orquestado mediante *Kubernetes* y herramientas variadas para hacerlo posible y creíble para su creación.

Dicho en otras palabras, desarrollará una idea/concepto general de una infraestructura IT contenerizada o de contenerización que abarcan sus respectivas características conocidas, solo que, dotándola de:

- 1. **Flexibilidad**, para poder estar conectado desde cualquier equipo sea donde sea.
- 2. **Escalabilidad**, para una respuesta rápida sin encontrarse con inconveniencias como la saturación de procesos ya que los contenedores son ligeros y no se sobrecargan ya que son controlados por un único equipo.
- 3. **Alta disponibilidad**, para estar totalmente disponible sea el momento que sea.
- 4. **Compatibilidad**, para que se pueda implementar e implantar en diferentes arquitecturas (local o en la nube y, si se puede el caso, hiperconvergente).

La configuración deberá realizarse en diferentes máquinas con el sistema operativo recién instalado, si bien también se deben llevar a cabo periódicamente sobre cualquier máquina para comprobar todos y cada uno de los estados en el que se encuentran los contenedores implementados.

## ◦ 3. Descripción

Como breve descripción para este proyecto en relación a lo que se va a tratar, hay que tener en cuenta los siguientes rasgos o requisitos:

- Al ser una **infraestructura de tecnologías de la información o TI**, cabe recordar indicar **de qué o cómo está compuesta**.
- **De qué manera se ha llevado a cabo**.
- **Explicar y describir qué medios/recursos se han empleado para su creación**.
- **Etc.**

### ◦ 3.1. Alcance

Este documento se ha elaborado para proporcionar información específica con objeto de implementar tecnologías de contenerización sobre una empresa, o incluso con la oportunidad de ser de carácter particular, con el sistema operativo **Linux**, instalado, tanto en inglés como en español en su última versión.

El documento incluye:

- Herramientas que fueron utilizadas para la creación de dicho proyecto y hacerlo posible.
- Ejecución de las fases de diseño de la infraestructura acompañadas de esquemas y capturas de pantalla para idear una base del concepto de *“Infraestructura de contenerización o contenerizada”*.
- Tablas de presupuesto para realizar una valoración económica de la misma.

## ◦ 3.2. Medios a utilizar

Para el desarrollo de este proyecto serán necesarias diversas herramientas y más adelante se describirá los medios a utilizar. Estas serán las herramientas a utilizar:

### VMware

**VMware** es un software de virtualización que ofrece la posibilidad de virtualizar diversos sistemas operativos para todo tipo de máquinas, tanto ordenadores personales como servidores de empresas.

Gracias a **VMware** dispondremos de un entorno de implementación de sistemas aislado. Especialmente, usaremos su nueva versión **Workstation Pro 16** que mejora la integridad gráfica y el rendimiento de las VMs en las cuales vamos a crear y diseñar nuestra infraestructura.

### Sistemas Operativos:

- Linux:
  - **Ubuntu 20.04 Server**, también conocido como “**Focal Fossa**” lanzado el 20 de abril de 2020, es una distribución de Linux cuya implementación está orientada a servidores. Es un sistema de código abierto, este sistema operativo será el indicado para nuestra infraestructura.
  - **Ubuntu 20.04 Desktop**, mismas características que su versión **Server**, pero es utilizado a nivel de equipos clientes.
  - **Linux Mint 20**, conocido como “**‘Ulyana’ XFCE Edition**” basada en **Ubuntu 20.04**, es una de los sistemas operativos alternativos que podemos implementar en nuestra infraestructura tanto a nivel de servidor y/o a nivel de cliente. Se caracteriza por ser liviano, veloz

y por disponer un escritorio gráfico llamado “*Cinnamon*” para aquellos usuarios familiarizados con el escritorio de *Windows*.

*\*Se había empleado como sistema operativo de pruebas antes de implementarlo en el proyecto para comprobaciones de rendimiento y validez para nuestra infraestructura contenerizada.*

### **Docker**

**Docker** es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización a nivel de sistema operativo en *Linux*. Esto evita la sobrecarga de realizar máquinas virtuales únicamente dedicadas a una aplicación.

Es sin duda una de las herramientas principales para el desarrollo del proyecto, puesto que es esencial para la contenerización de aplicaciones. Su gran comunidad de usuarios hace que sea un entorno de desarrollo muy amplio y con múltiples posibilidades.

### **Kubernetes**

**Kubernetes (k8s)** es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Google liberó el proyecto Kubernetes en el año 2014 y se basa en la ejecución de aplicaciones en producción a gran escala junto a las mejores ideas y prácticas de la comunidad.

Tiene varias características, se puede pensar en **Kubernetes** como:

- Una plataforma de contenedores
- Una plataforma de microservicios
- Una plataforma portable de la nube

Y mucho más...

**Kubernetes** ofrece un entorno de administración centrado en contenedores. Orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Opera a nivel de contenedor y no a nivel de hardware, ofrece algunas características que las **PaaS** (*Platforms As A Services*) también ofrecen, como **deployments** (despliegues), escalado, balanceo de cargas (**load balancing**), registros (**logs**) y monitoreo.

### Netdata

**Netdata** es una herramienta de monitorización que soluciona problemas de ralentización y anomalías en nuestra infraestructura con miles de métricas por segundo, visualizaciones significativas y unas perspicaces alarmas de salud (refiriéndose a el estado en que se encuentran de los procesos) sin ninguna configuración necesaria.

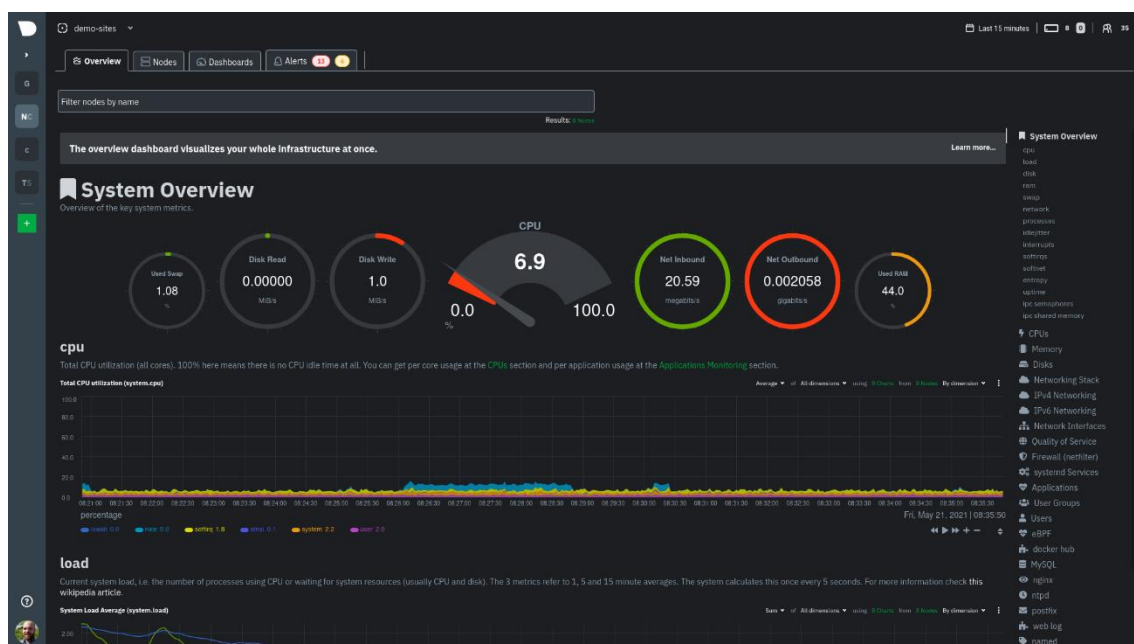


Figura 3 – Dashboard de Netdata

Es gratuito, sus agentes de monitorización trabajan con **Netdata Cloud** para ayudarnos a monitorizar y solucionar problemas en cada capa de nuestros sistemas para encontrar debilidades antes de que se vuelvan en cortes de servicios.

## Helm

**Helm** es una herramienta para gestionar aplicaciones de **Kubernetes**. **Helm** nos ayuda a “timonear” **Kubernetes** usando “cartas de navegación” (*charts*), conocidas como **Helm Charts**. La principal función de Helm es definir, instalar y actualizar aplicaciones complejas de Kubernetes. Helm es mantenido por la **CNCF, Cloud Native Computing Foundation** (Fundación de Informática Nativa en la Nube) en colaboración con **Microsoft, Google, Bitnami** y la comunidad de **Helm**.

Con **Helm Charts** es posible crear, versionar y publicar una aplicación **Kubernetes**. Cuando usamos los charts tenemos un asistente de optimización que facilita la administración e instalación de las aplicaciones **Kubernetes** y el proceso de empaquetamiento.

## Nginx

**Nginx** es un servidor web de código abierto que, desde su éxito inicial como servidor web ahora también es usado como proxy inverso, cache de HTTP y **load balancer**. Está diseñado para ofrecer un bajo uso de memoria y alta concurrencia (procesamiento de múltiples solicitudes que se ejecutan al mismo tiempo). En lugar de crear nuevos procesos para cada solicitud web, **Nginx** usa un enfoque asíncronico basado en eventos donde las solicitudes se manejan en un solo hilo. Un proceso maestro puede controlar múltiples procesos de trabajo. El proceso maestro mantiene los procesos de trabajo, y son estos lo que hacen el procesamiento real. Algunas características comunes que se ven en Nginx incluyen:

- Proxy inverso con caché (servidor intermediario que se asienta entre el cliente y el servidor maestro para el envío de tráfico directamente a los clientes)
- IPv6



- Balanceo de carga (*load balancing*)
- *Websockets* (para establecer una conexión bidireccional entre *frontend* (interfaz) y *backend* (servidor))
- Certificación *TLS/SSL*

### *Let's Encrypt*

*Let's Encrypt* es una autoridad de certificación (*CA, Certification Authority*) gratuita, automatizada y abierta para el beneficio del público. Es un servicio provisto por el *Internet Security Research Group (ISRG)*. Distribuyen certificados digitales gratuitamente a personas que necesitan poder habilitar el uso del protocolo *HTTPS (SSL/TLS)* en sitios web asegurando la privacidad y la seguridad a los usuarios. Básicamente, su objetivo es hacer posible la configuración de un servidor *HTTPS* y hacer que obtenga automáticamente un certificado confiado por el navegador, sin ninguna intervención humana. Esto se logra ejecutando un agente de manejo de certificados en un servidor web.

### *Cloudflare*

*Cloudflare* es una de las redes más grandes del mundo. Hoy en día, los sitios web y las aplicaciones de las empresas, organizaciones, etc. con presencia de Internet son más rápidos y seguros gracias a *Cloudflare*. Es una empresa que proporciona una red de entrega de contenido, servicios de seguridad de Internet y servicios de servidores de nombres de dominio distribuidos, localizados entre el visitante y el proveedor de alojamiento del usuario de *Cloudflare*, y que actúan como proxy inverso para sitios web.

## Apache Guacamole™

*Apache Guacamole* es una puerta de enlace de acceso remoto sin cliente (*clientless*). Soporta protocolos estándar como *VNC* (Linux), *RDP* (Windows) y *SSH*. Se le llama *clientless* porque no necesita de ningún plugin o de software cliente. Gracias a *HTML5*, una vez que Guacamole esté instalado en un servidor, todo lo necesario para acceder a nuestros escritorios es a través de un buscador web.

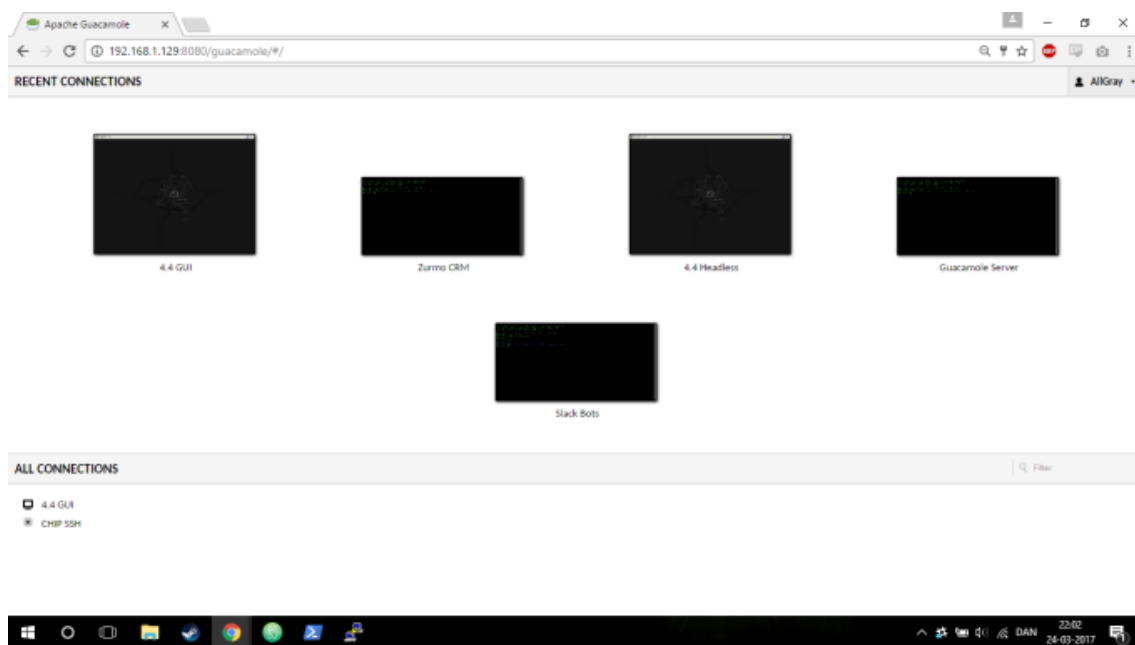


Figura 4 - GUI de Apache Guacamole

## Lens

*Lens* es un IDE de código abierto para administrar clústeres de Kubernetes multiplataforma. A través de *Lens*, podemos administrar fácilmente varios clústeres de *Kubernetes*. Algunos beneficios de usar **Lens** incluyen:

- Confianza en que nuestros clústeres están configurados correctamente.
- Mayor visibilidad, estadísticas en tiempo real, flujos de registros y capacidades prácticas de resolución de problemas (si disponemos de *Grafana*).

- La capacidad de trabajar con nuestros clústeres de forma rápida y sencilla, mejorando radicalmente la productividad y la velocidad del negocio.

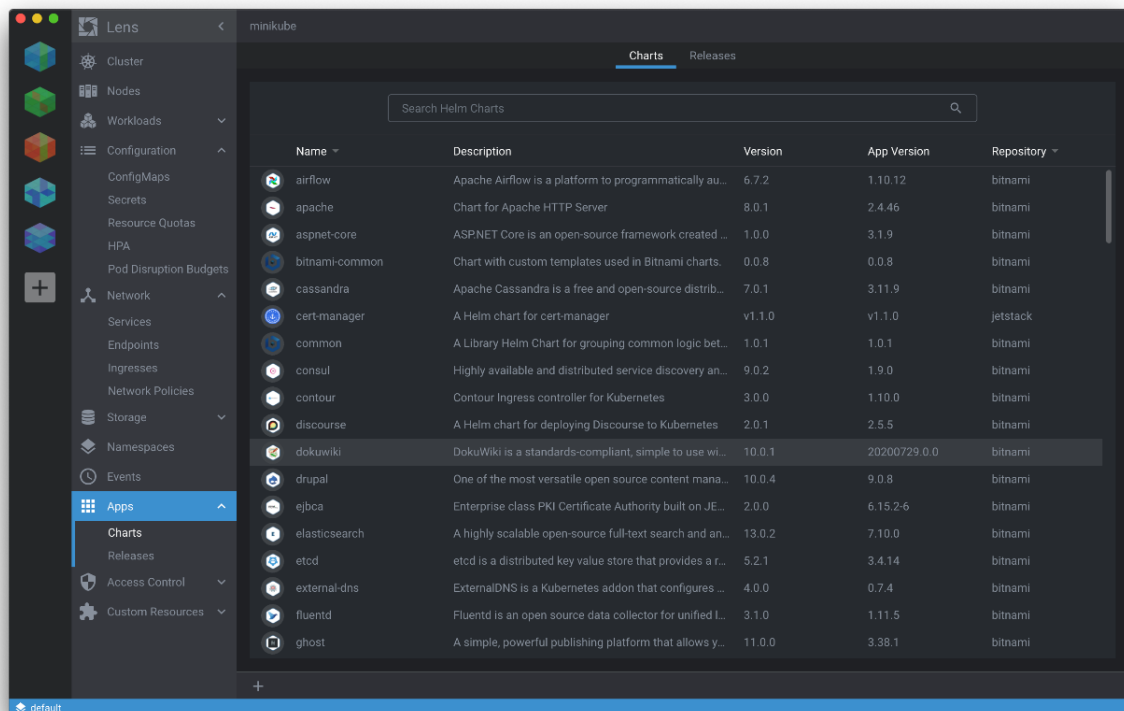


Figura 5 – Interfaz gráfica de Lens

## Freenom

**Freenom** es un proveedor de dominios públicos gratuitos del mundo, facilita la construcción de sitios web y sus contenidos, para cualquier empresa o persona de forma gratuita. Emplea un método de direccionamiento y enrutamiento llamado **AnyCast** en el que las peticiones entrantes pueden ser encaminadas a una variedad de diferentes localizaciones o nodos. **Freenom** garantiza la estabilidad y el rendimiento de todos los dominios que administra.

## OpenVPN

**OpenVPN** es tanto un protocolo VPN como un software que utiliza técnicas VPN para asegurar conexiones punto a punto y de sitio a sitio. Actualmente, es uno de los protocolos VPN más populares entre los usuarios de VPN. La mayoría de los proveedores de VPN y expertos en seguridad realmente recomiendan adherirse a **OpenVPN** si se desea disfrutar de una experiencia en línea privada, de vigilancia y sin piratas informáticos.

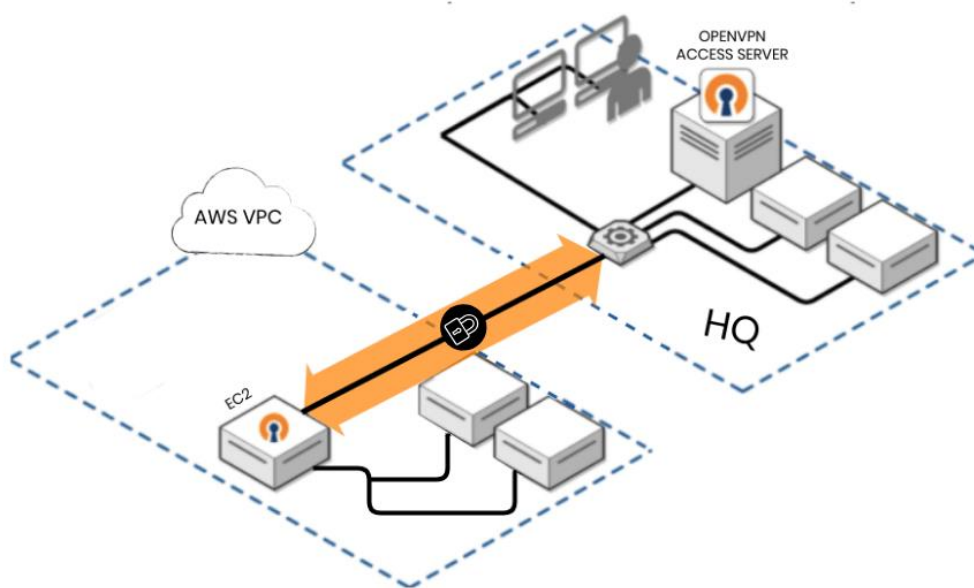


Figura 6 - Esquema de conexión VPN de OpenVPN de ejemplo

## Notion

**Notion** es un sistema de organización de información. Aglutina en una sola app las funciones de otras muchas que usamos frecuentemente, como editor de textos, toma de notas, gestión de tareas y proyectos, etc. Desde un punto de vista práctico, se puede decir que es una “navaja suiza de la productividad”. Nos permite almacenar de forma ordenada u práctica casi cualquier unidad de

información. **Notion** permite guardar dos tipos de estructuras de información básicamente:

- Páginas, un documento de texto como si estuviésemos usando un procesador de texto como **Word**, notas como **Evernote**, lista de tareas, recordatorios en un calendario, vídeos de **YouTube**...

Podemos personalizar cada hoja con una portada superior y un icono si lo deseamos. Además, podremos organizarlo de forma que podremos insertar o enlazar a otras páginas desde cualquier punto de la página, ordenar el contenido en columnas, cambiar el tipo de tipografía, establecer el ancho del documento, etc.

- Bases de datos, poder crear bases de datos totalmente flexibles, sobre la marcha, que nos permiten crear elementos dentro de ellas y a cada elemento asignarle unas propiedades.

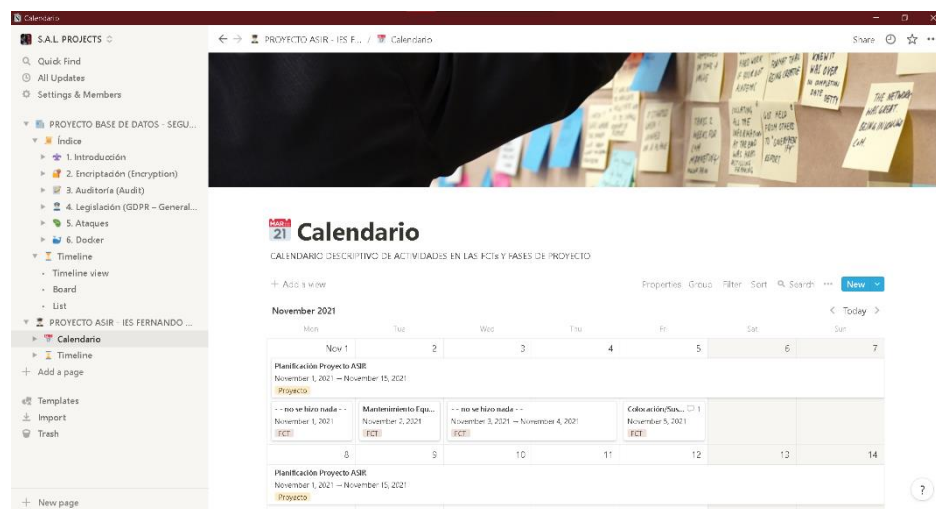
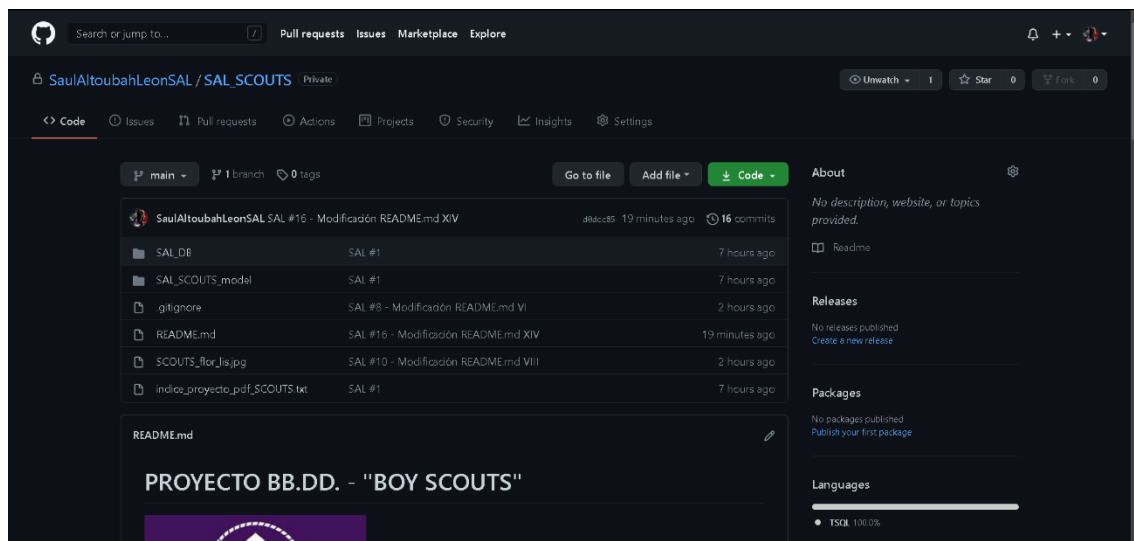


Figura 7 - Plano de control personal de Notion

## GitHub

**GitHub** es un servicio de hosting de repositorios almacenados en la nube. Esencialmente, hace que sea más fácil para individuos y equipos usar **Git** como la versión de control y colaboración. La interfaz de GitHub es sencilla de usar,

algunas personas lo usan para administrar cualquier tipo de proyectos como desarrollo de código, escribir libros...



*Figura 8 - Plano de control personal de GitHub*

Estas serán las herramientas utilizadas y alguna que otra que no se haya nombrado que posiblemente se le alude más adelante a lo largo de este documento. Para más información, avanzar sobre el siguiente punto · [5. Ejecución](#) en donde se explicarán con más detalle los recursos a emplear.

### **Equipo Personal**

Solamente queda indicar que se utilizará un PC portátil personal con 8 GB de RAM, CPU IntelCore i7-7700HQ de 4 núcleos y 8 hilos y un SSD externo de 500GB en donde se alojarán las VMs.



## ◦ 4. Planificación

En este apartado se usará la herramienta *Notion* para mostrar la temporalidad de las fases por las que se ha llevado a cabo el desarrollo y elaboración del proyecto y la herramienta *GitHub* para mostrar el repositorio público en donde se muestra el código y las capturas de su creación y desarrollo. Toda la información sobre el proyecto se encuentra en los enlaces **(MUY RECOMENDABLE ACCEDER A ELLOS)**:

- Notion: <https://aware-mirror-1cd.notion.site/PROYECTO-ASIR-IES-FERNANDO-WIRTZ-50b662a3fba94430b08b6ef92782dfbb>
- GitHub: [https://github.com/SaulAltoubahLeonSAL/SAL\\_kITs](https://github.com/SaulAltoubahLeonSAL/SAL_kITs)

## ◦ 5. Ejecución

Para diseñar una infraestructura de tecnologías de la información contenerizada hay que saber primero qué es.

Una infraestructura de tecnologías de la información o TI se trata de un conglomerado de elementos necesarios para operar y gestionar entornos de tecnologías de la información en empresas. Estos elementos incluyen el hardware, el software, los elementos de red, sistemas operativos y almacenamientos de datos. Todos ellos para ofrecer servicios y soluciones TI. Sus elementos se pueden clasificar de la siguiente manera:

- Hardware:
  - Servidores
  - Datacenters (centros de datos)
  - Equipos informáticos
  - Routers
  - Hubs/Switches
  - Etc.
- Software:
  - CMS, Content Management Systems (Sistemas Gestores de Contenido): *Wordpress, Moodle...*
  - Sistemas Operativos: *Windows, Linux* (Ubuntu, Debian...), *MacOS*.
  - *ERP, Enterprise Resource Planning* (Planificación de Recursos Empresariales): *Odoo, Holded, Sellenne...*
  - Etc.

Puede implementarse en un sistema de *cloud computing* (informática en la nube) o en instalaciones tradicionales:

- Tradicional: también conocida como *on-premise* (local), las empresas son las propietarias de todos los elementos (servidores, almacenamiento de datos, entre otros), a los cuales gestionan en sus propias instalaciones
- En la nube (*Cloud Computing*): la infraestructura en la nube hace referencia a los elementos y los recursos que se necesitan para el *cloud computing*. Existen 3 tipos: pública, privada e híbrida.
- Hiperconvergente: permite gestionar los recursos informáticos, de red y de almacenamiento desde una sola interfaz. Así podrá admitir cargas de trabajo más modernas con arquitecturas escalables en el sector a través de la combinación del almacenamiento de datos y la informática definidos por software.

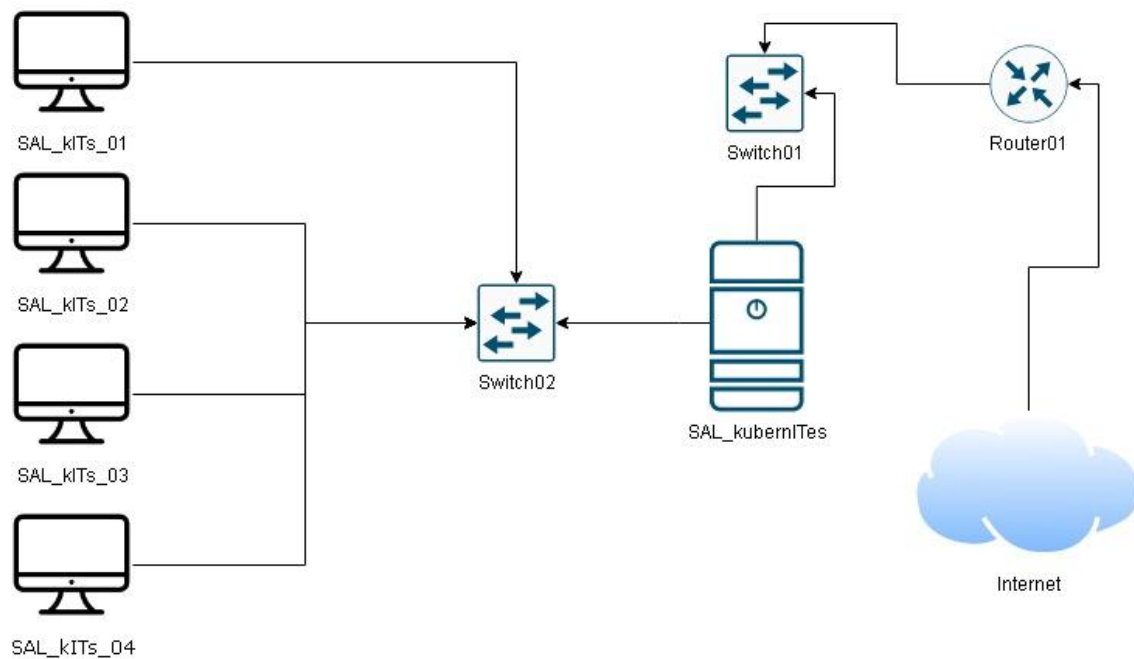
Y ahora, en relación a todo lo comentado en el punto · [1. Introducción](#), ¿cómo podemos definir una infraestructura IT contenerizada (*“integrada en Kubernetes”*)?

Podríamos decir que, se trataría de dicha infraestructura optimizada en escalabilidad y flexibilidad gracias al uso de contenedores para agilizar su funcionalidad a la hora de ofrecer servicios y soluciones TI que se adapta a cualquier implementación que se quiera realizar, tanto si se instala en servidores *bare-metal*, en nubes públicas, privadas o híbridas, o se preconfigura y se implanta en servidores dedicados hiperconvergentes; de esta manera dotándola de adaptabilidad a cualquier configuración que se nos proponga dependiendo de: *¿a quién se va a dirigir la empresa? ¿cuáles son sus objetivos a alcanzar? ¿y con qué estrategia se van a basar sus acciones de cara a sus clientes?*

## • 5.1. Diseño

### • Esquema físico:

En el esquema siguiente se muestra una nueva propuesta de infraestructura diseñada para este proyecto y se detalla cada uno de los componentes.

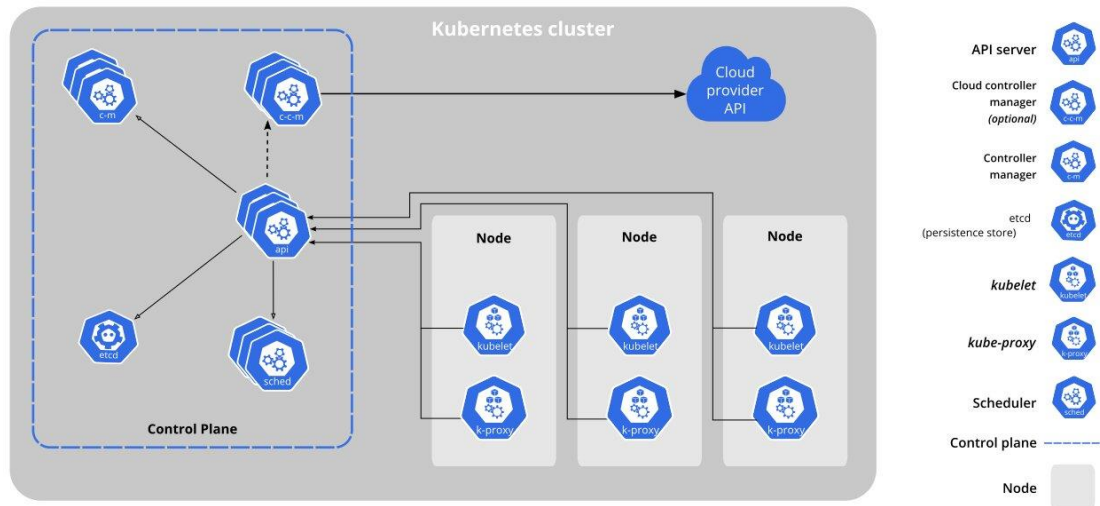


*Figura 9 – Esquema resumido de los dispositivos físicos en la infraestructura*

Como se puede observar en la figura, la infraestructura propuesta para este proyecto se trata de un ejemplo simple que muestra el conexionado de los equipos físicos incluyendo el enrutamiento al que deben seguir.

### · Esquema de contenerización:

En el esquema siguiente se muestra la arquitectura de Kubernetes recomendada que se iría a implementar en el equipo servidor:



**Figura 10 - Arquitectura de Kubernetes**

Esta arquitectura está formada por los siguientes componentes:

- **Kube-apiserver**: es el componente del plano de control de *Kubernetes* que expone la API de *Kubernetes*. El servidor API es el *frontend* para el plano de control de *Kubernetes*.
- **Etc**: es “la base de datos” de Kubernetes usado como almacenamiento de todas las claves, componentes y datos de todo nuestro clúster.
- **Kube-scheduler**: es el componente “programador” del plano de control que vigila a los *Pods* sin nodo asignado y les asigna uno para ponerlos en funcionamiento.
- **Kube-controller-manager**: es el componente del plano de control que ejecuta los procesos del controlador. Lógicamente, cada controlador está en un proceso separado, pero para reducir su complejidad, están todos compilados en un único binario y un único proceso.
- **Kubelet**: un agente que arranca cada nodo en el clúster. Se asegura que los contenedores se están ejecutando en un *Pod*.

- **Kube-proxy**: es el proxy de red que se ejecuta en cada nodo en nuestro clúster, implementando parte del concepto de los *Servicios* de *Kubernetes*.
- **Pod**: es el objeto más pequeño y básico desplegable en *Kubernetes*, es una unidad básica de ejecución que representa procesos corriendo en nuestro clúster. Tiene la función de encapsular un contenedor (o en algunos casos varios contenedores), recursos de almacenamiento, una dirección IP única que es válida solo dentro de nuestro clúster y las opciones que definen como el contenedor debe ejecutarse.
- **ReplicaSet**: es una regla definida mediante un *Deployment* en *Kubernetes* y usado para garantizar la disponibilidad de un número específico de *Pods* idénticos.
- **Deployment**: es un conjunto de órdenes declaradas que se le dan a nuestro clúster de *Kubernetes* todos los *ReplicaSets* y los *Pods* se alinean a lo que diga un *Deployment*.

Estos son los componentes más conocidos de *Kubernetes*, pero en este caso vamos a crear una versión simplificada y ligera utilizando *Minikube*. *Minikube* utiliza en un solo clúster el nodo master y el nodo *worker*:

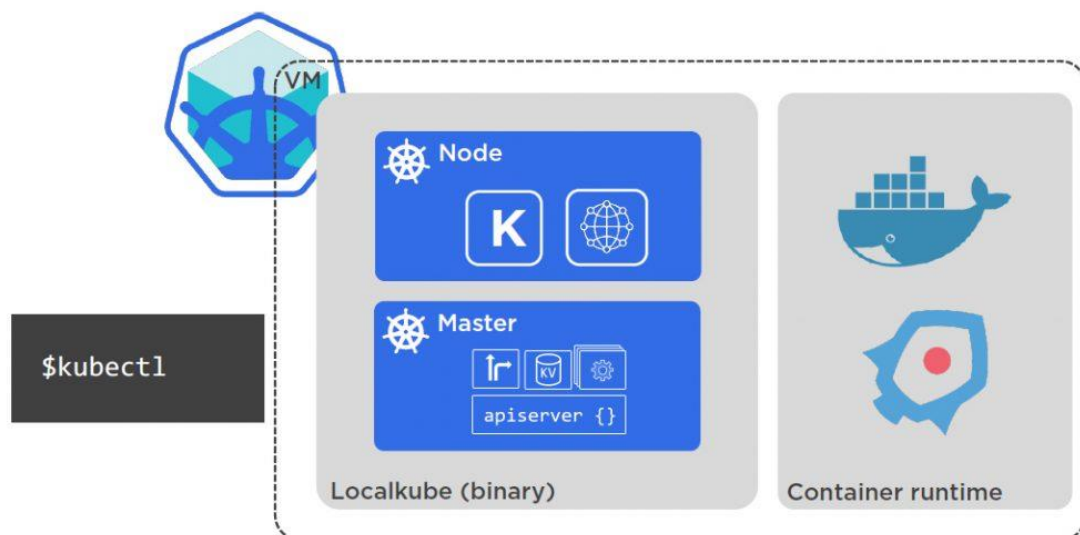


Figura 11 - Arquitectura de Minikube



Como podemos observar, este clúster de pruebas se va a ejecutar dentro de una VM en la que se le instalarán los paquetes de instalación de *Docker* y de *Minikube*.

Y finalmente esta será el esquema general de contenedores a seguir para la infraestructura:

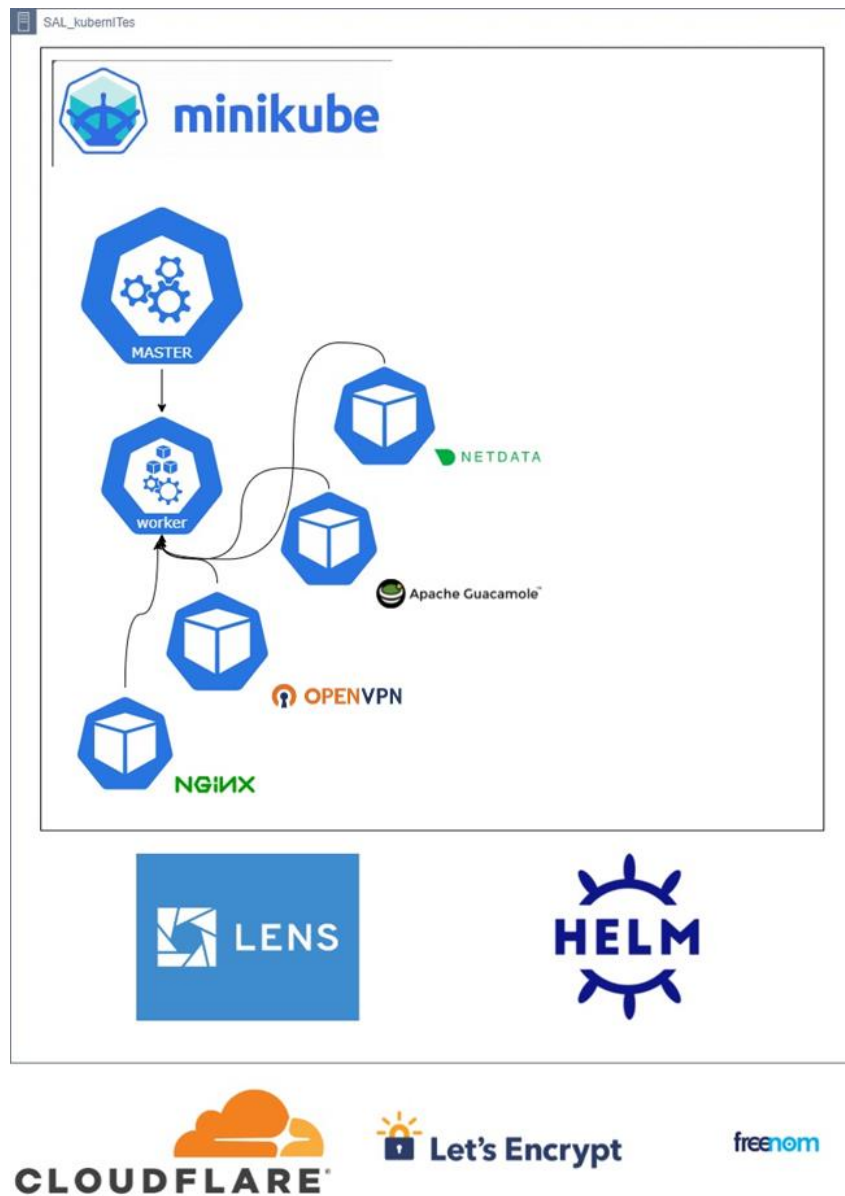


Figura 12 - Esquema general de contenedores con sus respectivas herramientas

Los requisitos de las VMs que se han seguido para la elaboración del proyecto son los siguientes:

| Servidor SAL_kubernITes         |                             |
|---------------------------------|-----------------------------|
| • <u>Almacenamiento interno</u> | 200 GB                      |
| • <u>Memoria RAM</u>            | 2,5 GB (recomendables 8 GB) |
| • <u>Nº CPUs</u>                | 4 vCPUs                     |
| • <u>Tarjeta de red</u>         | NAT                         |

Tabla 1 - Requerimientos mínimos para VM servidor SAL\_kubernITes

| Cliente SAL_kITs                |                           |
|---------------------------------|---------------------------|
| • <u>Almacenamiento interno</u> | 100 GB                    |
| • <u>Memoria RAM</u>            | 1 GB (recomendables 2 GB) |
| • <u>Nº CPUs</u>                | 2 vCPUs                   |
| • <u>Tarjeta de red</u>         | NAT                       |

Tabla 2 - Requerimientos mínimos para VM cliente SAL\_kITs

\*Los requerimientos mínimos para la creación de las VMs disponen de esos datos debido a que, personalmente, no se disponía de suficiente potencial de hardware en relación con la memoria RAM y CPU y se ha tenido que recortar lo máximo posible para poder ejecutar el proyecto sin sufrir problemas de por medio. Espero que se tenga en cuenta.

## • 5.2. Implementación

### • Sistema operativo del servidor y clientes: SAL\_kubernITes & SAL\_kITs

El entorno utilizado para el desarrollo es una pequeña máquina virtual con un sistema operativo **Ubuntu 20.04 Server** contenido en ella que, a su vez, se ha instalado la versión Desktop en los clientes. Puede o no disponer de interfaz gráfica en caso de querer acceder a por CLI mediante SSH, pero le dotaremos de escritorio gráfico instalándole el escritorio “**Cinnamon**” o el escritorio por defecto que suele venir implantado en su versión **Desktop**. Si se quiere usar otra distribución de **Linux** como **Linux Mint 20** como se ha comentado anteriormente, cumple con su propósito igualmente.

### • Instalación de Docker:

Para la instalación de la CLI de **Docker**, se deben ejecutar la siguiente lista de comandos:

| Comandos   |
|--|
| 1. <code>sudo apt update &amp;&amp; sudo apt full-upgrade -y</code>  |
| 2. <code>sudo apt install apt-transport-https ca-certificates curl software-properties-common -y</code>          |
| 3. <code>curl -fsSL https://download.docker.com/linux/ubuntu/gpg   sudo add-key add -</code>                     |
| 4. <code>sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"</code> |
| 5. <code>apt-cache policy docker-ce</code>   |
| 6. <code>sudo apt install docker-ce -y</code>  |
| 7. <code>sudo usermod -aG docker \${USER}</code>   |
| 8. <code>su - \${USER}</code>  |
| 9. <code>reboot</code>   |
| 10. <code>id -nG</code>  |
| 11. <code>sudo service docker status</code>  |

Tabla 3 - Lista de comandos para la instalación de CLI de Docker

### · Instalación de Minikube:

El siguiente paso es instalar el clúster de *Minikube*. No es necesario configurar nada ya que solamente es activar el clúster y ya se puede trabajar con él. Los comandos a ejecutar son:

| Comandos  |
|---|
| <pre>1. curl -LO    https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 2. sudo install minikube-linux-amd64 /usr/local/bin/minikube 3. minikube start</pre> |

Tabla 4 - Lista de comandos para la instalación del clúster de Minikube

Se recomienda ejecutar estos comandos como administrador, no como *root*. Alternativamente, *Minikube* puede descargar la versión de *kubectl* y deberíamos ser capaces de usarlo tal que así: ***minikube kubectl - get pod -A***

Para agilizar a la hora de ejecutar el comando *kubectl* se recomienda crear un alias: ***alias kubectl="minikube kubectl --"***

### · Instalación de herramientas Nginx, OpenVPN...:

Como siguientes pasos en la implantación de la instalación de las herramientas en nuestra infraestructura, se recomienda ir al siguiente apartado · [4. Planificación](#) en donde se encuentra el enlace al repositorio público del proyecto.

## • 5.3. Administración

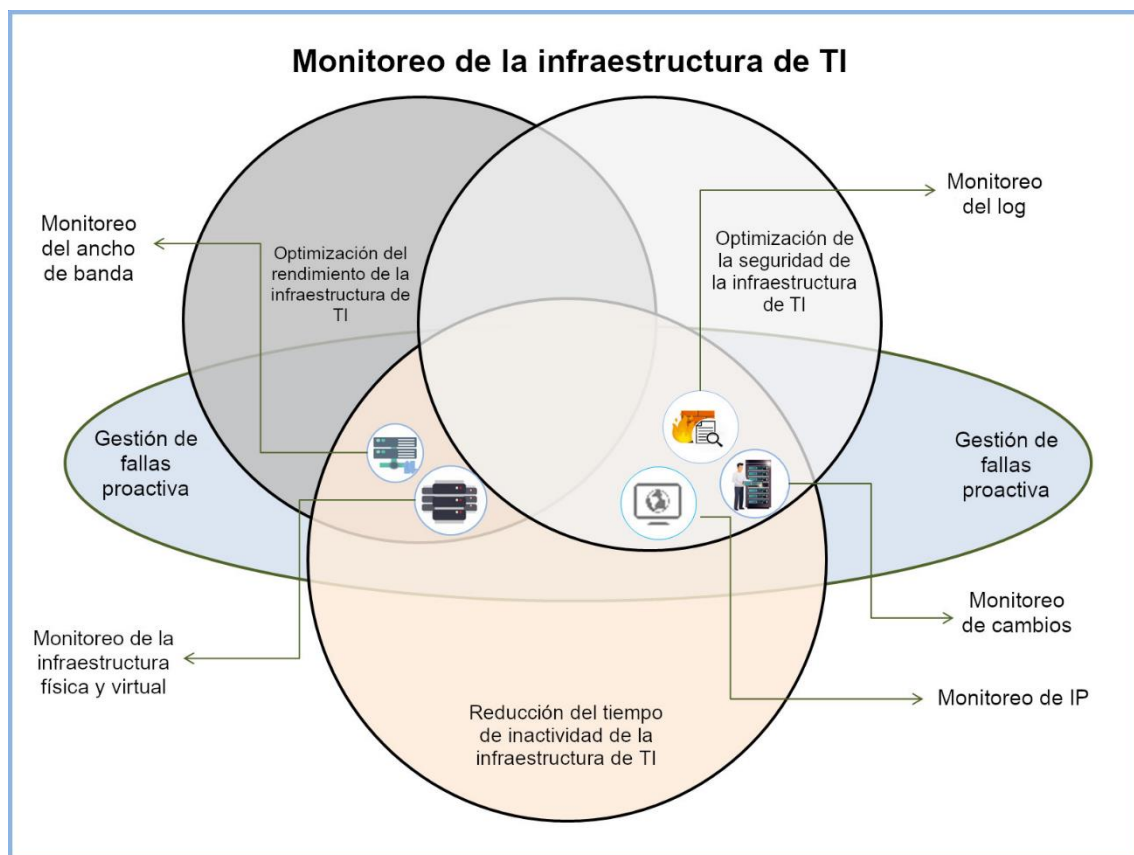
### • 5.3.1. Monitorización

El monitoreo de la infraestructura TI (*ITIM, Information Technologies Infrastructure Library*) es el proceso de seguimiento de los parámetros críticos en diversos dispositivos físicos y virtuales de la infraestructura TI de una organización. Se trata de una actividad de extremo a extremo destinada a garantizar el rendimiento óptimo de los elementos individuales de la infraestructura TI, eliminando el tiempo de inactividad y garantizando una gestión más rápida de los fallos y un monitoreo proactivo de la seguridad TI.

El monitorio de la infraestructura TI es una actividad complicada, ya que hay múltiples dispositivos y factores implicados, a cada uno de los cuales hay que dar la importancia que le corresponde. Algunos de los factores críticos que pueden dar lugar a la degradación del rendimiento de la TI incluyen:

- Excesiva utilización e inconsistencia en la salud y disponibilidad del hardware.
- Consumo de ancho de banda mal administrado entre los dispositivos individuales.
- Error de configuración durante las instalaciones o mejoras de la infraestructura TI.
- Falta de cumplimiento de las medidas esenciales de conformidad.
- Reglas y políticas ineficaces o anticuadas del firewall.

Elaborar una estrategia de monitoreo de la infraestructura TI claramente definida es esencial para garantizar que la infraestructura TI funcione de forma eficiente.



*Figura 13 – Esquema de monitoreo de la infraestructura TI general*

El monitoreo de la infraestructura TI puede dividirse en cinco componentes individuales, cada uno de los cuales tiene por objeto lograr la optimización del rendimiento, la optimización de la seguridad, la reducción del tiempo de inactividad o una combinación de los tres. Los cinco componentes son los siguientes:

- **Monitoreo de la infraestructura física y virtual**: el monitoreo de la infraestructura física y virtual se ocupa de garantizar el buen estado, la disponibilidad y el rendimiento óptimo de todos los dispositivos críticos de una red. Incluye el monitoreo de la red, monitoreo del servidor y el monitoreo de la salud y el rendimiento de los dispositivos virtuales. El

monitoreo de la infraestructura física y virtual no se limita a los dispositivos, sino que se extiende al monitoreo de los diversos procesos y servicios que se ejecutan en esos dispositivos. El monitoreo de la salud y la disponibilidad es un aspecto importante que ayuda a reducir el tiempo de inactividad y a optimizar el rendimiento de toda la infraestructura TI, lo que da lugar a una estrategia de gestión de la infraestructura TI altamente eficiente.

- **Monitoreo del ancho de banda:** el seguimiento del consumo de ancho de banda es otro aspecto importante del monitoreo de la infraestructura TI que ayuda a optimizar la disponibilidad y el rendimiento de los dispositivos en una infraestructura TI. El seguimiento del ancho de banda debe realizarse tanto a nivel de infraestructura como a nivel global y de red. La planificación proactiva del consumo de ancho de banda, junto con el monitoreo activo y en tiempo real de los patrones de tráfico de la red, es clave para garantizar que su infraestructura TI no sucumba a las trampas de un uso mal gestionado del ancho de banda.
- **Monitoreo de cambios:** La planificación de un proceso de implementación y gestión de cambios claramente definido es fundamental para garantizar que el entorno de la infraestructura TI se mantenga bien protegido de los desastres debidos a los errores de configuración. El término “cambio” en este contexto se refiere no sólo a los cambios de configuración sino también a las alteraciones a nivel de seguridad (firewall, directivas, etc.). Cualquier cambio, independientemente de su naturaleza, tamaño, importancia e implicaciones, debe ser cuantificado, se le debe asignar una jerarquía de aprobación bien definida y debe ir precedido de una copia de seguridad de los cambios existentes. Todo el proceso de gestión de cambios también debe evaluarse a nivel de usuario para garantizar que sólo los administradores de TI con los privilegios apropiados puedan implementar los cambios.





debido al uso mínimo de hardware disponible, pero se encuentran fuera de peligro.

- Lens:

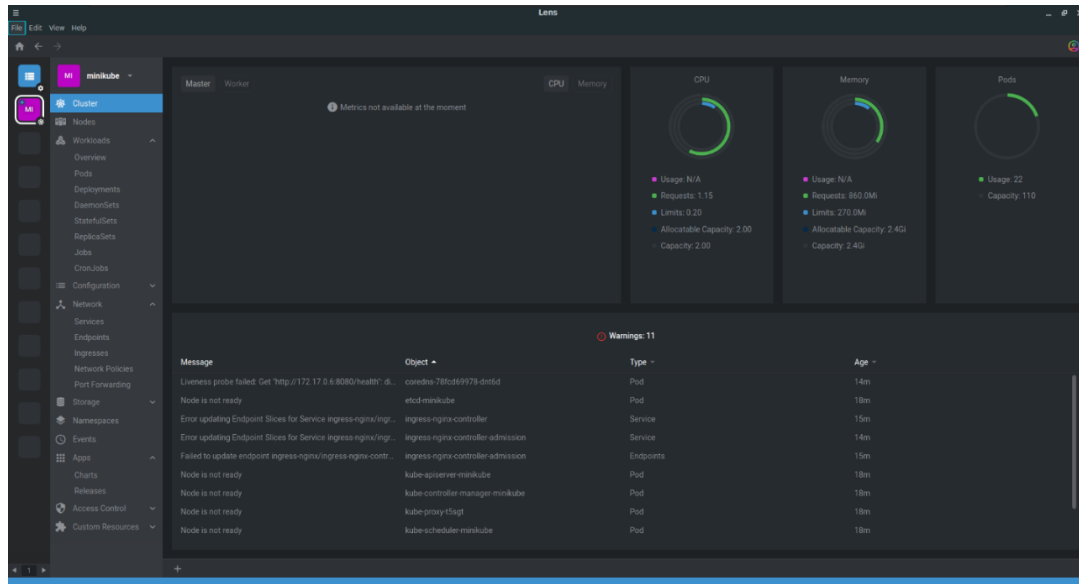


Figura 16 - Panel de control del clúster en Lens

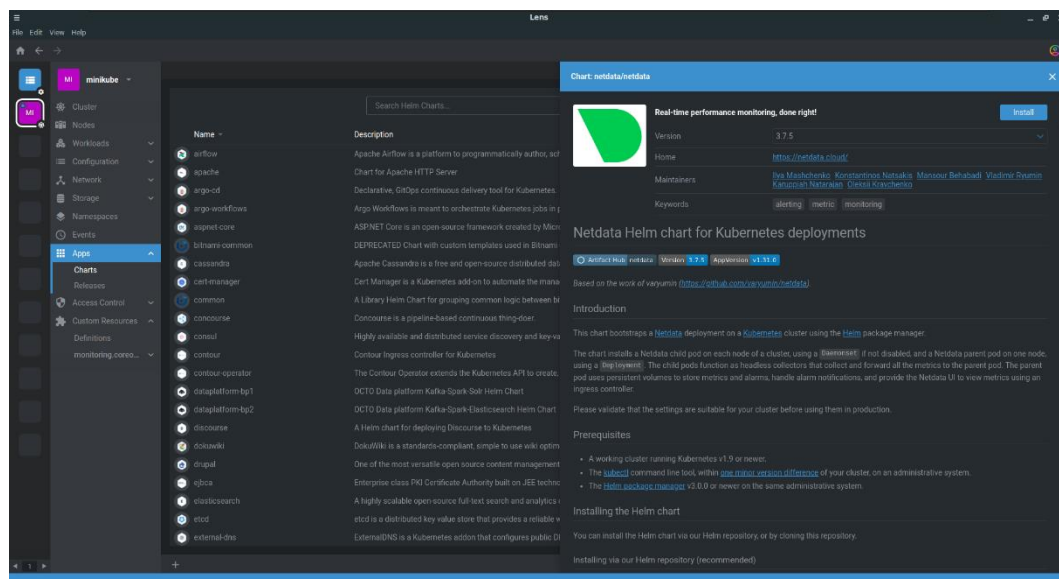


Figura 15 - Helm Chart de Netdata en Lens

Como se comentó anteriormente en el apartado “2.1. Medios a utilizar”, Lens es una aplicación que sirve para conectarse a cualquier tipo de clúster de *Kubernetes*, en este caso el nuestro, mediante una interfaz gráfica todo lo que disponemos en él. Podemos realizar las mismas acciones de gestión y

monitorización en el Dashboard de *Kubernetes* sólo que tendremos a nuestra disposición la instalación de *charts* de *Helm* en la pestaña “Apps > Charts”

- Netdata:

Como se explicó anteriormente en el apartado “2.1. Medios a utilizar” la herramienta de monitorización que se utilizará en este proyecto, en relación a las anteriores, será *Netdata*.

*Netdata* tiene la característica de monitorizar cualquier cosa: servidores (de bases de datos, web), aplicaciones en la nube, clústeres (*Kubernetes*, *AWS*, *Microsoft Azure*...), etc.



Figura 17 - Interfaz local de monitorización de Netdata

En esta captura observamos la monitorización del propio pod del servicio de *Netdata* que muestra los procesos de CPU, tráfico de red, etc. y se ha accedido localmente realizando un *port-forward* al servicio.

Otra manera monitorizar con *Netdata* es a través de su propia plataforma en la nube, *Netdata Cloud*, ingresando nuestros nodos tanto el propio equipo como el clúster entero o el mismo pod en sí:

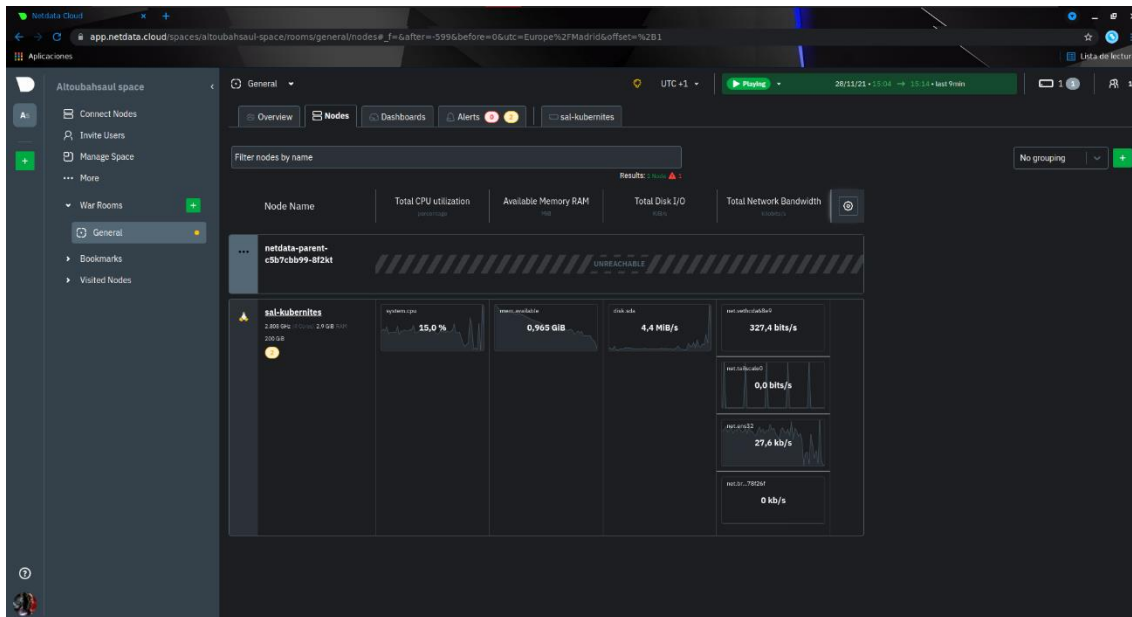


Figura 18 – Netdata Cloud, pestaña de nodos conectados

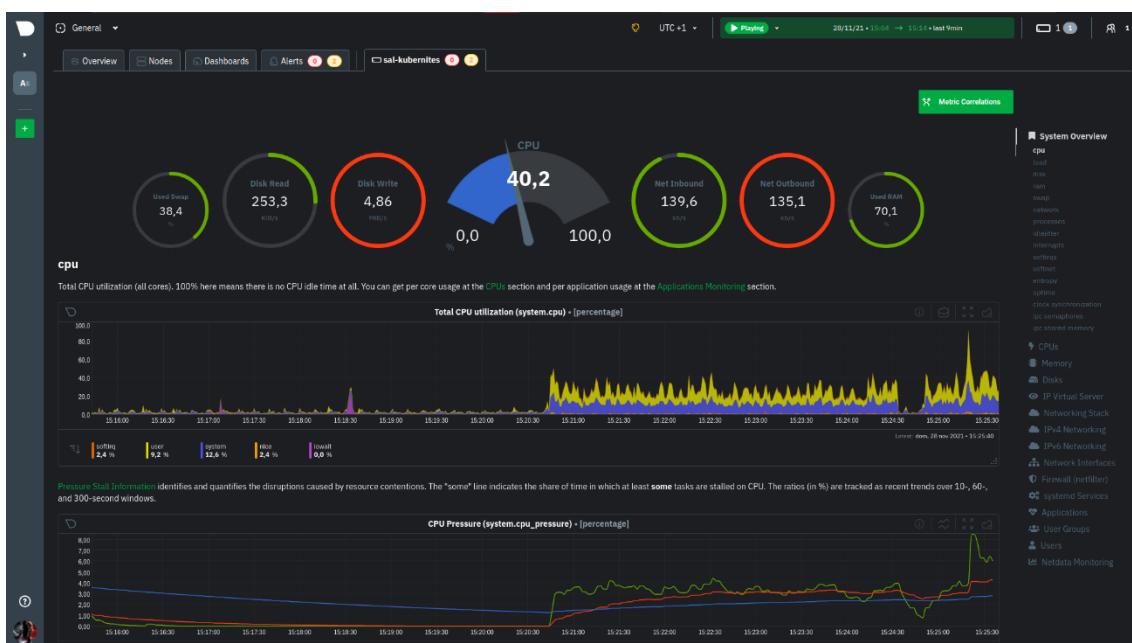


Figura 18 – Netdata Cloud, panel de supervisión general del servidor SAL\_kubernITes

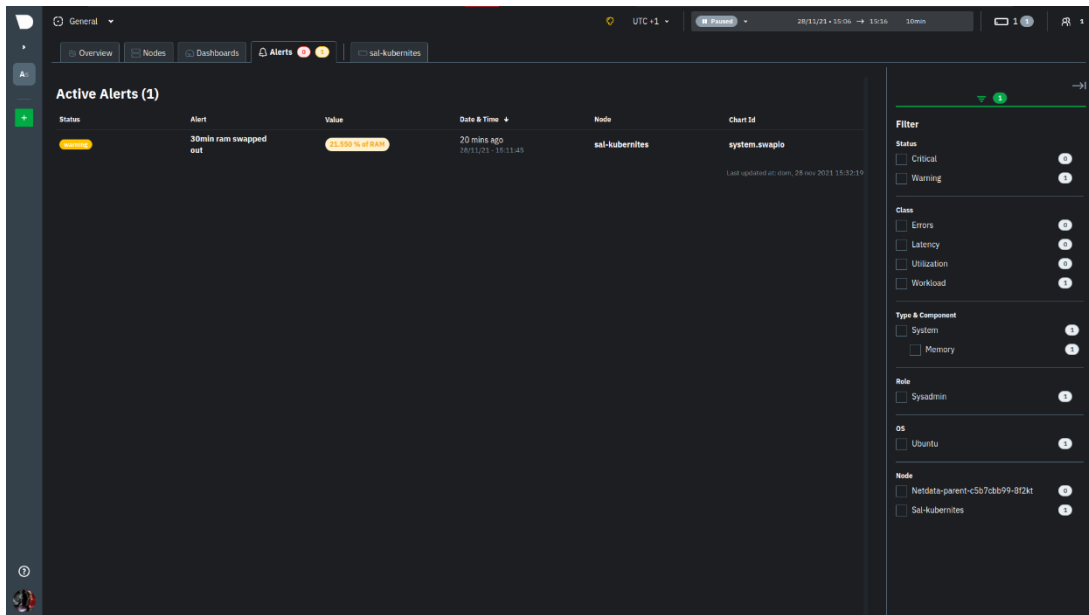


Figura 20 – Netdata Cloud, panel de alertas

En la pestaña de alertas, cuando un nodo en *Netdata* sufre algún tipo de anomalía como, por ejemplo, detención forzosa de servicios, inactividad de nodos, etc., recibiremos vía correo electrónico alertas de advertencias y/o errores por parte del panel de monitorización de *Netdata* indicándonos de manera resumida el mensaje y las acciones a realizar dependiendo del tipo de aviso que hemos recibido:

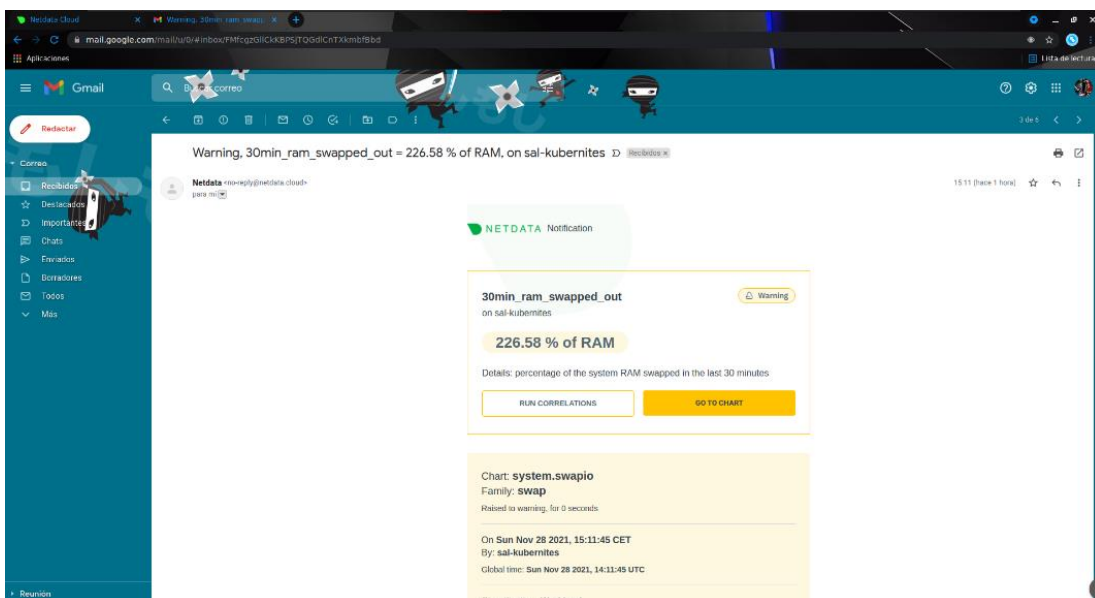


Figura 21 - Advertencia de Netdata recibida por email

- Logs de Kubernetes:

Los registros (**logs**) pueden ayudarnos a entender qué está ocurriendo dentro de nuestros recursos. Los **logs** son particularmente útiles para “depurar” problemas y monitorizar la actividad de nuestro clúster. La mayoría de las aplicaciones modernas tienen algún tipo de mecanismo de registro. Igualmente, el motor de contenedor está diseñado para soportar los registros. El método más fácil y mayor adoptado para los registros en aplicaciones contenerizadas es escribir en emisiones de datos de salida y de datos de error.

Sin embargo, la funcionalidad nativa provista por un motor de contenedores o un **runtime** (tiempo de ejecución) siempre no es suficiente para completar un remedio a los registros. Por ejemplo, puede que queramos acceder a los registros de nuestras aplicaciones si el contenedor “crashea”; un pod es corrupto; o un nodo muere. En un clúster, los registros deberían tener un almacenamiento separado y un ciclo de vida independiente de los nodos, pods o contenedores. Este concepto es llamado “registro a nivel de clúster”.

Las arquitecturas de registro a nivel de clúster requieren de un **backend** separado para almacenar, analizar y consultar registros. **Kubernetes** no provee una solución nativa de almacenamiento para los datos de registro. En vez de eso, hay muchas maneras de realización de registro con **Kubernetes**.

También los hay tipo “registro a nivel de nodo”, “retransmisión de un contenedor **sidecar**”...

```
sal@sal-kubernites:~$ ll *.yaml *.yaml
-rw-rw-r-- 1 sal sal 214 nov 28 22:23 counter-pod.yaml
-rw-rw-r-- 1 sal sal 469 nov 26 13:25 override.yaml
-rw-rw-r-- 1 sal sal 257 nov 28 01:10 sal-deployment.yaml
sal@sal-kubernites:~$ cat counter-pod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: contador
spec:
  containers:
  - name: contar
    image: busybox
    args: [/bin/bash -c,
          'i=0; while true; do echo "$(date)"; i=$((i+1)); sleep 1; done']
sal@sal-kubernites:~$ kubectl apply -f http://k8s.io/examples/debug/counter-pod.yaml
pod/counter created
sal@sal-kubernites:~$ kubectl logs couter
Error from server (NotFound): pods "couter" not found
sal@sal-kubernites:~$ kubectl logs counter
Error from server (BadRequest): container "count" in pod "counter" is waiting to start: ContainerCreating
sal@sal-kubernites:~$ kubectl logs counter
0: Sun Nov 28 21:38:21 UTC 2021
1: Sun Nov 28 21:38:22 UTC 2021
2: Sun Nov 28 21:38:23 UTC 2021
3: Sun Nov 28 21:38:24 UTC 2021
4: Sun Nov 28 21:38:26 UTC 2021
5: Sun Nov 28 21:38:27 UTC 2021
6: Sun Nov 28 21:38:28 UTC 2021
7: Sun Nov 28 21:38:29 UTC 2021
sal@sal-kubernites:~$ kubectl logs counter
0: Sun Nov 28 21:38:21 UTC 2021
1: Sun Nov 28 21:38:22 UTC 2021
2: Sun Nov 28 21:38:23 UTC 2021
3: Sun Nov 28 21:38:24 UTC 2021
4: Sun Nov 28 21:38:26 UTC 2021
5: Sun Nov 28 21:38:27 UTC 2021
6: Sun Nov 28 21:38:28 UTC 2021
7: Sun Nov 28 21:38:29 UTC 2021
8: Sun Nov 28 21:38:30 UTC 2021
9: Sun Nov 28 21:38:31 UTC 2021
10: Sun Nov 28 21:38:32 UTC 2021
```

Figura 22 – Terminal de sal\_kubernITes, resultado del registro del pod “counter”

### • 5.3.2. Seguridad

La seguridad de una infraestructura IT contenerizada se compone de las mismas medidas que se toman para proteger una infraestructura IT tradicional, añadiendo medidas adicionales para proteger el contenido de las máquinas virtuales. Estas medidas pueden incluir, por ejemplo, el cifrado de los datos almacenados, el uso de controles de acceso basados en roles para limitar el acceso a los servidores y equipos clientes, o el uso de herramientas de monitoreo para detectar y responder a amenazas en tiempo real. En 2017, aproximadamente el 82% de las empresas que utilizaban contenedores de TI en 2016 consideraron la seguridad como el factor clave a la hora de adoptar esta tecnología. Reiterada seguridad se basa en tres pilares fundamentales:

- Seguridad de los contenedores: los contenedores de TI se construyen sobre la seguridad de los datos y se utilizan para protegerlos, incluye medidas de autenticación, autorización, privacidad y rendición de cuentas.
- Seguridad de los entornos: los contenedores se ejecutan en entornos seguros que se construyen sobre la seguridad de los datos. Dichos entornos incluyen medidas de protección contra intrusiones, prevención de pérdida de datos detección de malware y protección de la privacidad.
- Seguridad de las aplicaciones: las aplicaciones se ejecutan en contenedores seguros y están protegidas contra intrusiones, pérdida de datos y malware.



En otras palabras, se compone por múltiples capas de seguridad, cada una diseñada para proteger una única pieza de la infraestructura. La capa más baja de seguridad (**seguridad lógica**) es el firmware del equipo, que se encarga de proteger el sistema operativo y el software de la infraestructura. La segunda capa de seguridad (**seguridad de red – protección interna**) es el firewall, que se encargan de proteger la red de la infraestructura. La tercera capa de seguridad (**seguridad de red – protección externa**) son los antivirus y los antispyware, que se encargan de proteger los equipos de la infraestructura de virus y malware. La cuarta capa (**seguridad de acceso lógico**) de seguridad son las contraseñas, que se encargan de proteger los archivos y los programas de la infraestructura. La quinta capa de seguridad (**seguridad de carácter externo**) son los sistemas de detección de intrusos, que se encargan de proteger la infraestructura de ataques externos. La sexta capa de seguridad (**seguridad de datos**) son los sistemas de copia de seguridad, que se encargan de proteger la información de la infraestructura en caso de una pérdida accidental. La séptima capa de seguridad (**seguridad de control de acceso lógico**) es el sistema de gestión de contraseñas, que se encargan de proteger las contraseñas de la infraestructura.

#### ~ Configuración inicial

Por defecto el sistema operativo base de los contenedores, crea ciertas configuraciones para facilitar el acceso al usuario, habilitando la mayor parte de funcionalidades y aumentando la velocidad de instalación del mismo. Estas configuraciones en muchas ocasiones pueden ser motivo de posibles brechas de seguridad.

Para evitar brechas innecesarias, se configurarán ciertos parámetros de manera correcta:

- **Creación de imágenes ajustadas (Docker)**: la base de la seguridad en contenedores se centra en saber qué sistema operativo se está ejecutando, qué contenido está incluido en el mismo y qué permisos posee el acceso. Del mismo modo hay que revisar, protocolos y servicios que son publicados por el contenedor. Para lograr un mayor control de todo esto, la premisa siempre será la creación de contenedores propios y firmados.
- **Creación de un usuario distinto a root**: cuando se habla de root, se refiere a la cuenta superusuario de Linux, aquella que posee todos los privilegios y permisos para realizar acciones sobre el sistema. Para la realización de tareas cotidianas dentro de un contenedor, es necesario configurar usuarios adicionales, que normalmente no vienen configurados ni habilitados, con menor privilegio que sean los encargados de realizar estas tareas.
- **Contraseña segura para root**: para ciertas acciones que afectan al sistema de archivos, se requiere tener acceso root. Sin embargo, se debe tener un conocimiento sobre las acciones que se realizan (que serían, en este caso, las auditorías), ya que una acción realizada de manera errónea podría ocasionar daños importantes en el sistema. Para evitar el uso de instrucciones con privilegios de superusuario la cuenta root tiene que estar dotada con una contraseña segura que evite que cualquier usuario malintencionado pueda comprometer de algún modo el sistema.
- **Usuarios UID 0**: en el fichero `/etc/passwd` existe un campo UID por cada usuario, que corresponde al identificador de cada usuario. Algunas distribuciones de Linux por defecto crean varios usuarios con UID 0 que corresponde al identificador de superusuario. Si existen varios superusuarios en el sistema la probabilidad de vulnerar el mismo es mayor, por este motivo se deben limitar los usuarios con UID 0

únicamente a root, siendo el único usuario habilitado para tener control total sobre el sistema.

A terminal window titled 'sal@sal-kubernites: ~' with a menu bar (Archivo, Editar, Ver, Buscar, Terminal, Ayuda). The command 'cat /etc/passwd | grep ":0:"' is entered and executed, resulting in the output 'root:x:0:0:root:/root:/bin/bash'. The prompt returns to 'sal@sal-kubernites:~\$'.

Figura 23 – Termina en SAL\_kubernITes, ejecución del comando "cat" empleando el filtro "grep" que muestra los usuarios con UID 0

- **Cuentas sin contraseñas:** en Linux existe la opción de configurar una cuenta de usuario sin contraseña, aunque ese usuario no pertenezca a los denominados “*sudoers*” (administradores). En el sistema no debe haber ningún usuario sin contraseña, esto supondría una vulnerabilidad, ya que cualquier usuario podría acceder a información sensible sin necesidad de estar autorizado para ello.

#### ~ Configuración a nivel de host:

La seguridad de los contenedores se asemeja a asegurar cualquier proceso en ejecución. Por tanto, se debe tener en consideración la seguridad de todas las capas antes de implementar cualquier solución de contenerización. Otro dato a tener en cuenta es la seguridad en todo el ciclo de vida de la aplicación y por ende del contenedor.

Los contenedores facilitan la tarea de promoción y migración de uno o varios servicios entre diferentes sistemas compatibles. Esta tarea, si bien facilita la versatilidad de los contenedores, también puede provocar problemas de seguridad si no se realiza entre sistemas seguros, por tanto, una de las principales medidas a adoptar para la implementación de un sistema de contenedores pasa por escoger correctamente y bastionar el equipo o equipos que contendrán esa tecnología. Para un sistema Linux, se deben tener en cuenta las siguientes pautas a nivel de host.

- Crear una partición separada para los contenedores
- Usar un Kernel de Linux actualizado
- No usar herramientas de desarrollo en producción
- Realizar un correcto bastionado del sistema adaptándolo a las necesidades del contenedor
- Borrar todos los servicios no esenciales en el sistema anfitrión
- Mantener el Daemon actualizado (tecnologías basadas en *Docker*)
- Permitir solo a los usuarios autorizados controlar el demonio
- Auditar el Daemon (para *Docker*) y los registros del sistema:

#### Directorios

- `/var/lib/docker`
- `/etc/docker`
- `docker-registry.service`
- `docker.service`
- `/var/run/docker.sock`
- `/etc/sysconfig/docker`
- `/etc/sysconfig/docker-network`
- `/etc/sysconfig/docker-registry`
- `/etc/sysconfig/docker-storage`
- `/etc/default/docker`
- `/etc/containers/policy.json`
- `/usr/share/containers/libpod.conf`
- `/etc/containers/registries.json`
- `/etc/containers/storage.json`
- `/etc/containers`

Tabla 5 - Directorios para auditar

## ~ Configuración de ficheros y permisos

En el proceso de diseño de una infraestructura de contenedores, es de suma importancia limitar y aislar recursos. El sistema debe poseer mecanismos que permitan el uso de grupos y roles, adecuando los permisos a las necesidades de cada usuario limitando, por tanto, usuarios como “*root*”.

Para realizar una buena configuración inicial se deberán revisar ciertos servicios y ficheros para que posean los permisos necesarios y no pertenezcan a usuarios no habilitados para tal fin. Para ello se deberán revisar los siguientes ficheros o servicios:

- Verificar que el archivo ***docker.service*** pertenece al usuario y grupo ***root***.
- Verificar que los permisos del archivo ***docker.service*** se encuentran configurados al menos como 644.
- Verificar que el archivo ***docker.socket*** pertenece al usuario y grupo ***root***.
- Verificar que los permisos del archivo ***docker.socket*** se encuentran configurados al menos como 644.
- Verificar que el archivo de entorno *Docker* (***/etc/sysconfig/docker*** ó ***/etc/default/docker***) pertenece al usuario y grupo ***root***.
- Si se usa ***systemd*** en el sistema se deben verificar los siguientes permisos:
  - Verificar que el archivo ***/etc/sysconfig/docker-network*** pertenece al usuario y grupo ***root***.
  - Verificar que los permisos del archivo ***/etc/sysconfig/docker-network*** se encuentran configurados al menos como 644.
  - Verificar que los permisos del archivo ***/etc/sysconfig/docker-registry*** se encuentran configurados al menos como 644.
- Verificar que el directorio ***/etc/docker*** pertenece al usuario y grupo ***root***.

- Verificar que los permisos del directorio */etc/docker* se encuentran configurados al menos 755.
- Verificar que el certificado del “*registry*” pertenece al usuario y grupo *root*.
- Verificar que los permisos del certificado del “*registry*” pertenece se encuentran configurados al menos como 444.
- Verificar que el archivo de clave del certificado del servidor *Docker* pertenece al usuario y grupo *root*.
- Verificar que los permisos del archivo de clave de certificado del servidor *Docker* se encuentran configurados al menos como 400.
- Verificar que el archivo de socket de *Docker* pertenece al usuario y grupo *docker*.
- Verificar que los permisos del archivo de socket de Docker se encuentran configurados al menos como 600.

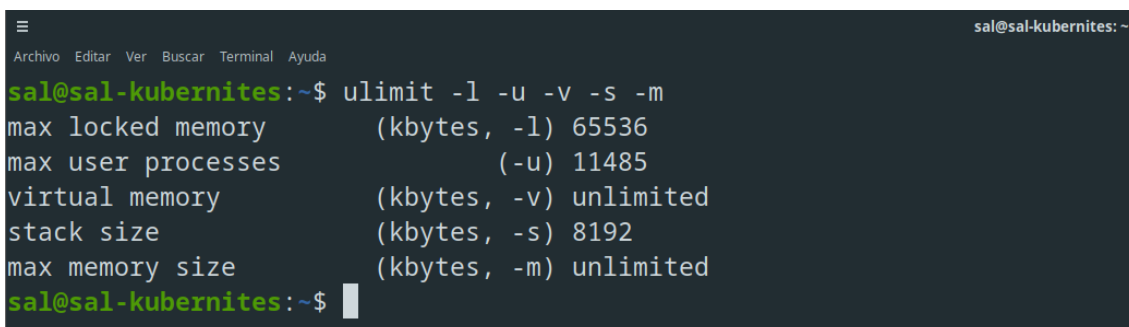
#### ~ Configuración a nivel de ejecución

La puesta en marcha de los contenedores motiva nuevas ramas de configuración como el nivel de ejecución de los procesos, la compartición de recursos con el host, los accesos administrativos y remotos, así como cualquier proceso que dependa de los contenedores actualmente en ejecución.

Para esta tarea se deben modificar ciertas configuraciones que limiten tanto la comunicación desde el anfitrión a los contenedores como los accesos a la configuración. Se describen a continuación ciertas configuraciones que deben tenerse en cuenta para la ejecución de los contenedores:

- Verificar las configuraciones de seguridad de *SELinux* (*CentOS*, *Fedora* y *Red Hat Enterprise Linux*).
- Verificar que los contenedores estén ejecutando un solo proceso principal.
- Restringir las “*Linux Kernel Capabilities*” dentro de los contenedores.

- Configurar correctamente los parámetros del kernel en el host.
- No usar contenedores con privilegios.
- No montar directorios sensibles del host en los contenedores.
- No ejecutar *SSH* dentro de los contenedores.
- No configurar puertos privilegiados dentro de los contenedores.
- Abrir sólo puertos necesarios en el contenedor.
- No usar el modo “*host network*” en un contenedor.
- Limitar el uso de memoria por contenedor.
- Configurar la prioridad de uso de CPU apropiadamente.
- Montar el sistema de ficheros raíz (/) de un contenedor como solo lectura (*RO*).
- Limitar el tráfico entrante al contenedor mediante una interfaz específica del anfitrión.
- Configurar la política de reinicio “*on-failure*” de un contenedor.
- No compartir *PID* de procesos del host con contenedores.
- No compartir *IPC* (*Inter-Process Communication*) del host con contenedores.
- No exponer directamente dispositivos del anfitrión en contenedores.
- Sobrescribir el “*ulimit*” (comando que establece o informa los límites de recursos de proceso del usuario) por defecto en tiempo de ejecución si fuera necesario.



```
sal@sal-kubernites:~$ ulimit -l -u -v -s -m
max locked memory      (kbytes, -l) 65536
max user processes      (-u) 11485
virtual memory          (kbytes, -v) unlimited
stack size              (kbytes, -s) 8192
max memory size         (kbytes, -m) unlimited
sal@sal-kubernites:~$
```

Figura 24 – Terminal de SAL\_kubernITes, resultado al ejecutar el comando “*ulimit -l -u -v -s -m*”

Por otra parte, en la zona de **Kubernetes**, la cosa cambia ya que, más allá de las funciones básicas que se han relatado anteriormente, no ofrece nada en lo que a seguridad de aplicación respecta, ni la protege de las vulnerabilidades de seguridad. Es aquí donde se deben utilizar programas adicionales o proveedores externos para garantizar que se implementen los sistemas de seguridad adecuados.

A continuación, estos son algunos factores a considerar cuando vayamos a proteger nuestros contenedores de **Kubernetes**:

- Configuraciones predeterminadas: uno de los principales factores a considerar cuando protejamos contenedores de Kubernetes son las configuraciones predeterminadas. Se deben comprobar todas las configuraciones predeterminadas de Kubernetes antes de uso con el fin de reducir el riesgo de que un ataque en un pod se propague a otros pods. Si bien **Kubernetes** tiene un marco específico como el control de acceso, generalmente la gran mayoría de estas funciones de control de acceso no están activadas de manera predeterminada. Es posible que estos tipos de controles tampoco estén configurados para reforzar las políticas con menor privilegio, proporcionando permisos totales a los usuarios que quizás no necesariamente necesiten esa información. Dejar posibles datos confidenciales expuestos de esa manera constituye un enorme riesgo, facilitando información confidencial a usuarios maliciosos.
- Tiempo de ejecución de un contenedor: un tiempo de ejecución del contenedor es una aplicación especial que ejecuta contenedores. Es importante comprender que **Kubernetes** no cuenta con protecciones frente a un ataque del tiempo de ejecución ni puede detectar intrusiones una vez que han tenido lugar. Si se detecta una filtración activa o una nueva vulnerabilidad en un contenedor en ejecución, todo el contenedor se debe eliminar y se debe volver a lanzar una versión no comprometida. La



información que solucionó la causa de origen de la incidencia de seguridad también se debe utilizar para volver a configurar el componente en el entorno.

- Imágenes: las imágenes también pueden hacer que los contenedores sean más vulnerables. Las imágenes indebidamente configuradas proporcionan un punto fácil acceso a los atacantes para penetrar en la red y las imágenes que contienen claves específicas de autenticación pueden ayudar a los cibercriminales en futuros ataques. La detección de código malicioso que está dentro de una imagen de contenedor requiere un análisis de vulnerabilidades en los registros y en la producción, lo cual no es una función de *Kubernetes*.
- Seguridad del host: intencionadamente, *Kubernetes* ejecuta contenedores en servidores asignados para ello. Dado que la herramienta de orquestación no tiene nada que ver con la seguridad de tales servidores, se deben utilizar otros procesos para supervisarlos en busca de problemas de seguridad. Muchas empresas se vuelcan en seguridad del host tradicionales en esta instancia para detectar *exploits* frente los recursos del sistema, pero en caso de que el host también esté comprometido, esto puede conllevar consecuencias devastadoras. Los sistemas del host se deben supervisar para buscar filtraciones y actividades sospechosas con el fin de combatir los ataques despiadados.
- Comunicaciones pod-pod: de manera predeterminada, *Kubernetes* no aplica la política de red a cada pod, lo que significa que los pods en el entorno de *Kubernetes* pueden comunicarse con otros. Es una ventaja para los contenedores y los pods que se comuniquen unos con otros en las implementaciones para un correcto funcionamiento. Sin embargo, esto se puede convertir rápidamente en un objetivo fácil para los ciberdelincuentes que solo tienen que infiltrarse en un contenedor para luego desplazarse lateralmente dentro del entorno. La asociación de una

política de red a un pod limita su comunicación a activos específicos, cumpliendo un rol similar a los controles y reglas de firewall.

- Protección de contenedores de *Kubernetes*: con una mayoría de organizaciones utilizándolo, garantizar la seguridad del contenedor de Kubernetes es crucial para mantener las redes y aplicaciones seguras de filtraciones y ataques maliciosos. Al integrar correctamente la seguridad en cada fase del ciclo de vida del contenedor de *Kubernetes*, las empresas pueden estar seguras de que están tomando todas las medidas adecuadas.

Y ahora una pregunta muy importante, ¿qué herramientas o métodos podríamos utilizar para solucionar estas inconveniencias? Pues esto es lo que haremos comenzando desde fuera a dentro, es decir, desde el usuario hasta el interior de la infraestructura.

Como ejemplo mínimo sobre esta infraestructura comenzaremos a crear el servidor VPN. Hay 3 opciones a crearlo:

- Pod *K8s* (siguiendo el esquema general de la infraestructura).
- Contenedor de *Docker* (creando un archivo ***.dockerfile*** o ***docker-compose.yaml*** mientras se usa).
- Instalación local

Teniendo en cuenta las opciones que disponemos, hay que saber que cada una de ellas tiene su funcionalidad y utilidad ya que posiblemente pueda afectar al rendimiento o, mejor dicho, a la estabilidad del servidor en sí porque, por ejemplo, los pods en *Kubernetes* llegan a fallar o a apagarse o a corromperse y salen en su lugar uno nuevo, pero los datos almacenados que estaban anteriormente se perdieron a causa de no disponer de un volumen que los almacene; aún así, no da lugar a que no sea una inconveniencia el hecho de no escoger esa opción. La segunda opción sería igual de válida que la anterior, pero no seguiría el esquema general. Y la tercera, sería una instalación común de

**OpenVPN** y configurar los archivos **.ovpn** para el cliente y el servidor y crear los certificados. En el caso de disponer un archivo **docker-compose.yaml**, junto con **Docker-compose** se genera el contenedor del servidor **OpenVPN**, se puede utilizar la herramienta **Kompose** para convertirlo en “despliegues” **.yaml** para crear los componentes necesarios en **Kubernetes**:

```
sal@sal-kubernites:~$ cat docker-compose.yaml
version: "2"
services:
  openvpn:
    cap_add:
      - NET_ADMIN
    image: kylemanna/openvpn
    container_name: openvpn
    ports:
      - "1194:1194/udp"
    restart: always
    volumes:
      - ./openvpn-data/conf:/etc/openvpn
sal@sal-kubernites:~$ kompose convert -f docker-compose.yaml
WARN Volume mount on the host "./openvpn-data/conf" isn't supported - ignoring path on the host
INFO Kubernetes file "openvpn-service.yaml" created
INFO Kubernetes file "openvpn-deployment.yaml" created
INFO Kubernetes file "openvpn-claim0-persistentvolumeclaim.yaml" created
sal@sal-kubernites:~$ ls -lih openvpn-*
3288743 -rw-r--r-- 1 sal sal 249 dic 1 22:11 openvpn-claim0-persistentvolumeclaim.yaml
3288742 -rw-r--r-- 1 sal sal 1,2K dic 1 22:11 openvpn-deployment.yaml
3288741 -rw-r--r-- 1 sal sal 420 dic 1 22:11 openvpn-service.yaml
sal@sal-kubernites:~$
```

Figura 25 – Ejemplo de creación y conversión **docker-compose.yaml** a “deployments” en **.yaml** para servidor **OpenVPN** en **Minikube**

Como siguiente paso quedaría comprobar permisos de los directorios mencionados anteriormente, gestionar permisos de usuarios, instalar antivirus, programas de detección de intrusos, etc.

### • 5.3.3. Almacenamiento de datos

El almacenamiento de datos en una infraestructura IT contenerizada se basa en el uso de *containers* (contenedores) que alojarán todos los elementos de tecnologías de la información. La infraestructura se puede escalar en función de las necesidades, añadiendo o eliminando contenedores. Cada contenedor tendrá asignado un número de identificación único que permitirá ubicarla en cualquier momento. Los contenedores estarán conectados entre sí a través de una red interna, de forma que se puedan intercambiar datos y recursos. La infraestructura se puede ampliar en cualquier momento añadiendo nuevos contenedores.

Los administradores dispondrán de una interfaz web o CLI para gestionar la infraestructura. A través de esta interfaz podrán realizar todas las operaciones necesarias como, por ejemplo:

- Crear, eliminar y reorganizar contenedores
- Gestionar el acceso a la infraestructura
- Monitorizar el estado de los elementos de TI alojados en cada contenedor.

Cabe insistir que los pods de *Kubernetes* son efímeros y la información guardada en ellos no es persistente (al igual que un contenedor en *Docker*), pero es evidentemente que necesitamos que nuestros contenedores y aplicaciones tengan la posibilidad de que su información no se pierda. La solución es añadir volúmenes (almacenamiento persistente) a los pods para que los puedan utilizar los contenedores. Los volúmenes son considerados otro recurso de *Kubernetes*.

En la definición en sí de un pod, además de especificar los contenedores que lo van a formar, también podemos indicar los volúmenes que tendrá. Siendo así que, la definición de cada contenedor tendremos que indicar los puntos de montajes de los diferentes volúmenes.

Reiterando información, los datos que se almacenan en los son efímeros, por lo que, si queremos mantener información de forma persistente, debemos almacenar esa información de algún modo. Esta persistencia de la información, en *Kubernetes*, se consigue mediante volúmenes. Éstos se montan en los pods cuando son creados. Algunos tipos de volúmenes son:

- ***emptyDir***: un volumen es creado primero cuando se asigna un pod a un nodo, y existe mientras el pod está corriendo en el nodo. Como su nombre lo indica, un volumen *emptyDir* está inicialmente vacío. Todos los contenedores en el pod pueden leer y escribir archivos en el volumen *emptyDir*, aunque ese volumen se puede montar en la misma o diferente ruta en cada contenedor. Cuando un pod es removido del nodo por alguna razón, los datos en *emptyDir* se borran permanentemente.
- ***hostPath***: se caracteriza por almacenar la información intra-nodo, es decir, almacena la información en una ruta dentro del nodo en que está corriendo el pod, por lo que no se puede emplear en un clúster. Este tipo de almacenamiento se puede utilizar para hacer pruebas en clústeres de 1 nodo, como *Minikube*, el cual estamos utilizándolo en este proyecto.
- ***local***: representa un dispositivo de almacenamiento local como un disco, una partición o un directorio. Los volúmenes locales solo se pueden usar como un PV (*Persistent Volume*) creado estáticamente. El aprovisionamiento dinámico no está soportado. Comparados con volúmenes *hostPath*, los volúmenes *local* se usan de manera duradera y portátil sin programar pods manualmente a los nodos. Si un nodo deja de estar sano, entonces el volumen local se vuelve inaccesible al pod y el volumen no se puede ejecutar.

### • 5.3.4. Redes

Planificar los servicios de red de una infraestructura IT normal tiene sus pros y sus contras a la hora de configurar el enrutamiento, los puertos, que reglas seguir, etc., pero ahora viene la parte de *networking* con **Kubernetes** y aparecen nuevos conceptos y normas que hay que entender. Existen varios tipos de comunicaciones de red en un clúster de **Kubernetes**: *container-container*, pod-pod, pod-servicio, tráfico externo – servicio. Y a su vez unas reglas que tenemos que entender:

- Los pods del clúster de **Kubernetes**, da igual en qué nodo corran, pueden comunicarse con otros pods de diferentes nodos sin hacer *NAT*.
- Elementos como los *daemons* del sistema (*kubelet*) pueden comunicarse con todos los pods en ese nodo.
- Todos los contenedores pueden comunicarse entre sí directamente sin *NAT*.
- Todos los nodos pueden comunicarse con todos los contenedores y viceversa sin *NAT*.
- La IP que un contenedor ve, es la misma para sí y para el resto de componentes.

Se puede decir, que cuando se trabaja con pods de Kubernetes, se intenta alcanzar el modelo de red de la virtualización con máquinas virtuales, ya que cada pod tiene una IP asignada de forma dinámica. Con lo que podemos usar esto para poder asignar puertos, descubrir servicios, equilibrio de carga...y poder llevar una aplicación que ya existía en una máquina virtual a **Kubernetes**.

Estas direcciones IP de las que hablamos que se asignan en los pods, residen en el mismo *Namespace*. Y los contenedores que corren en esos pods, que están en el mismo, son capaces de llegar a los puertos de escucha de cada contenedor dentro de ese *Namespace*. Lo que hay que tener claro, es que los contenedores dentro de cada pod tienen que gestionar el uso de los puertos de escucha. Y a nivel de host, es posible gestionar los puertos de escucha de los pods y reenviar a los puertos de escucha de los pods. Siendo algo totalmente transparente para el pod que está en ejecución.

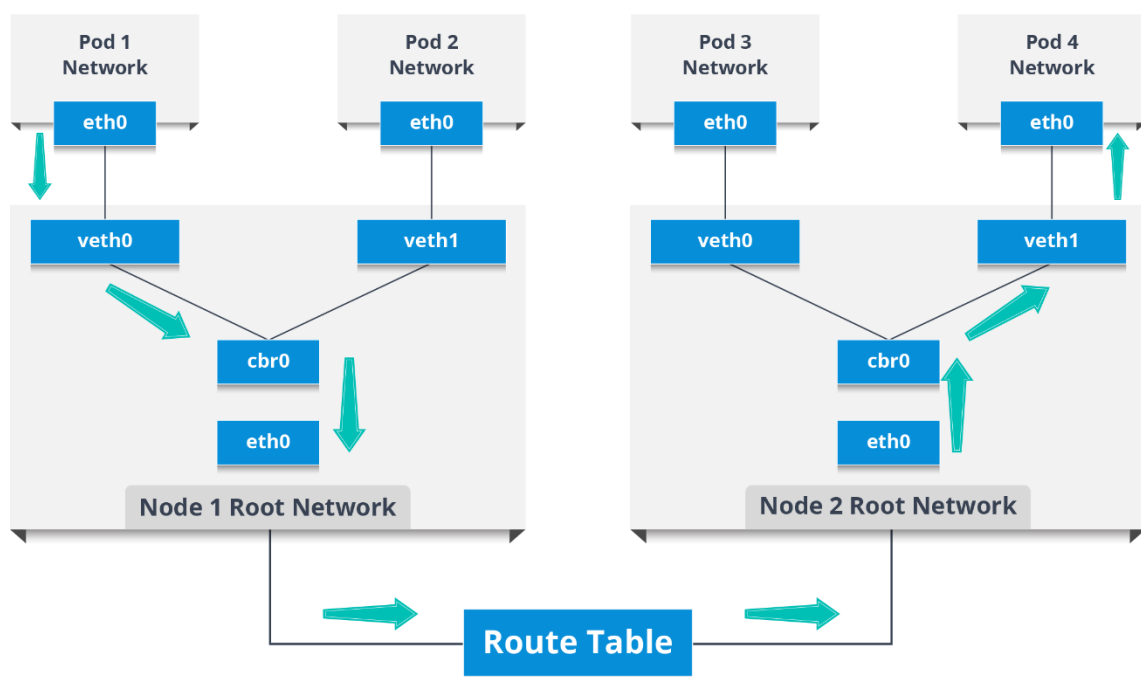


Figura 26 – Planificación de redes en Kubernetes entre dos nodos

La gestión de redes de *Kubernetes* utiliza *iptables* para controlar las conexiones de red entre pod y nodos, manipulando muchas de las redes y reglas de entrada de puertos. De esta manera, los clientes no necesitan mantener un trayecto de direcciones IP para conectarse a los servicios de *Kubernetes*. Incluso, el mapeo de puertos es bastante simplificada (y la mayor parte eliminada) desde que cada pod (repetimos) tiene su propia dirección IP y sus contenedores pueden escuchar en su puerto nativo.

## ◦ 6. Presupuesto

Para la planificación del presupuesto de este proyecto se llevará a cabo una estimación de los costes a invertir, en el caso de llevarlo a cabo a la realidad, con tal de estudiar si el proyecto es viable económicamente. Para ello intervienen los siguientes aspectos recomendados:

~ Costes Materiales: Hardware

| <u>Identificación</u>                           | <u>Estimación</u><br><u>precio</u> | <u>Descripción</u>  |
|---|------------------------------------|---|
| Servidor físico                                 | 2800 €                             | Ordenador Red Workstation Server Pro Ryzen 9 5950X, 64GB RAM, GTZ 1660 SUPER, 1TB SSD + 1 TB HDD                              |
| Despliegue de servidores virtuales              | -                                  | Dependerá si se va a implementar on-premise o en la nube (pública, privada o híbrida)   |
| Infraestructura de red (servicio e instalación) | 50 € /mes                          | El precio cambiará según qué ISP ( <i>Internet Service Provider</i> ) contratemos: <i>Movistar, Vodafone, Orange, Digi...</i> |
| PCs personales y periféricos                    | 1128 €                             | PC genérico (4 GB RAM, 250 GB SSD, 4 equipos) = 270 € x 4 = 1080 €<br>Periféricos (Ratón, teclado...) = 12 € x 4 = 48 €       |

Tabla 6 - Costes de recursos de hardware



~ Costes Materiales: Software

| <u>Identificación</u>                   | <u>Estimación precio</u>                       |
|---|--|
| <i>Docker</i>                           | 0 €  |
| <i>Kubernetes</i>                       | 0 €  |
| <i>Netdata</i>                          | 0 €  |
| <i>Lens</i>                             | 0 €  |
| <i>OpenVPN</i>                          | 0 €  |
| <i>Apache Guacamole</i>                 | 0 €  |
| <i>GitHub</i>                           | 0 €  |
| <i>VMware Workstation Pro 16</i>        | 171,65 €                                       |
| <i>Helm</i>                             | 0 €  |
| <i>Let's Encrypt (certificados SSL)</i> | 0 €  |
| <i>Cloudflare</i>                       | 200 €/mes (si es la tarifa “Bussiness”)        |
| <i>Freenom</i>                          | 0 € (en caso de escoger un dominio web gratis) |

Tabla 7 - Costes de recursos de software

~ Costes Materiales totales

| <u>Costes</u>                                       | <u>Precio total</u>                           |
|---|---|
| <i>Hardware</i>                                     | 2800 € + (50 € x 12) + 1128 € = <b>4528 €</b> |
| <i>Software</i>                                     | 171,65 € + (200€ x 12) = <b>2571,65 €</b>     |
| <b>TOTAL</b> *(si los costes son pasados 12 meses): | <b>4528 € + 2571,65 € = 7099,65 €</b>         |

*Tabla 8 - Costes totales*

Como se observa en la tabla anterior, el coste total aproximado del proyecto es de **7099,65 €** (si los costes son pasados 12 meses).

## • Conclusiones finales

Una vez llegado a este punto, se puede concluir que, en mayor o menor medida, se ha intentado cumplir y respetar todos los apartados del proyecto. Esto se debe a la **muy baja disponibilidad de recursos personal** en relación a la exposición de esta idea/concepto de infraestructura IT “integrada en *Kubernetes*” (contenerizada) causado por la **alta demanda de recursos de hardware (memoria RAM y CPU, exactamente)** a la hora de realizar cada implementación e implantación con el fin, vuelvo a reiterar, en ejecutar y llevar a cabo este proyecto para que sea **posible y creíble**.

- Se ha conseguido buscar una vía que pueda simular el uso de la orquestación con Kubernetes -> *Minikube*.
- El modelo de infraestructura no ha sido necesario realizar ninguna configuración al ser todo automático.
- Las funcionalidades del orquestador fueron implantadas de manera básica y se ha procurado seguir el esquema general.
- Hubo complicaciones de rendimiento a causa de las razones anteriormente redactadas que implicó el retraso en la ejecución de elaboración de esta memoria de proyecto.
- Se ha procurado seguir las medidas de seguridad para una mejor protección de la infraestructura.

Como párrafo final, insisto que se tenga en cuenta este documento ya que se ha intentado alcanzar las propuestas oportunas en cuanto a la propuesta de proyecto con el fin de no defraudar al equipo docente en la corrección de este proyecto y a los compañeros de prácticas que han estado animando y apoyando personalmente. Muchas gracias a todos.

## ◦ Bibliografía

1. “Es el fin de la infraestructura como servicio tal como la conocemos: esto es lo que sigue” - <https://www.rackspace.com/es/solve/its-end-infrastructure-service-we-know-it>
2. “VMware Workstation Pro 16” - <https://www.vmware.com/products/workstation-pro/workstation-pro-evaluation.html>
3. “Linux Ubuntu 20.04 Server” - [https://www.stackscale.com/es/blog/ubuntu-20-04-its/#Ubuntu\\_Server\\_2004](https://www.stackscale.com/es/blog/ubuntu-20-04-its/#Ubuntu_Server_2004)
4. “Linux Ubuntu 20.04 Desktop” - [https://www.stackscale.com/es/blog/ubuntu-20-04-its/#Ubuntu\\_Desktop\\_2004](https://www.stackscale.com/es/blog/ubuntu-20-04-its/#Ubuntu_Desktop_2004)
5. “Linux Mint 20” - <https://blog.redigit.es/que-es-linux-mint-y-como-instalarlo/>
6. “Docker” - <https://www.docker.com/>
7. “Kubernetes” - <https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/>
8. “Netdata” - <https://www.netdata.cloud/>
9. “Helm” - <https://helm.sh/https://www.aplyca.com/es/blog/helm-gestor-de-aplicaciones-para-kubernetes>
10. “Nginx” - <https://kinsta.com/es/base-de-conocimiento/que-es-nginx/>
11. “Let’s Encrypt” - <https://letsencrypt.org/>
12. “Cloudflare” - <https://www.cloudflare.com/es-es/learning/what-is-cloudflare/>
13. “Apache Guacamole” - <https://guacamole.apache.org/>
14. “Lens” - <https://k8slens.dev/>

15. “Freenom” - <https://www.freenom.com/es/aboutfreenom.html>
16. “OpenVPN” - <https://es.wikipedia.org/wiki/OpenVPN>
17. “Notion” - <https://notionapp.es/descargar-notion/>
18. “GitHub” - <https://github.com/>
19. “Contenerización en arquitecturas virtuales v1.0 – CCN, Centro Criptológico Nacional” - <https://www.ccn-cert.cni.es/pdf/guias/series-ccn-stic/guias-de-acceso-publico-ccn-stic/5828-ccn-stic-667-contenerizacion-en-arquitecturas-virtuales/file.html>
20. “Monitoreo de infraestructura TI, *ManageEngine*” - <https://www.manageengine.com/latam/it-operations-management/monitoreo-de-infraestructura-ti.html>
21. ¿Qué es la seguridad de contenedores de Kubernetes? - [https://www.trendmicro.com/es\\_es/what-is/container-security/kubernetes.html](https://www.trendmicro.com/es_es/what-is/container-security/kubernetes.html)
22. How Kubernetes Networking Works: The Basics - <https://blog.neuvector.com/article/kubernetes-networking>