Generalitat de Catalunya Departament d'Ensenyament	**INS Puig Castellar** **Departamento**: Informática **Ciclo Formativo**: CFGS Administración de Sistemas Informáticos y Redes **Mòdul Professional**: MP14 Proyecto	
	MEMORIA DEL PROYECTO	
	Alumno/s: Alejandro Mallen y Ángel de Caldas **Profesor:** Óscar Torrente	**Fecha**: 29/05/2020

Cluster Rancher con Raspberry Pi

Gandalf

(Proyecto de desarrollo)

CFGS Administración de Sistemas Informáticos y Redes

Angel de Caldas y Alejandro Mallen
2º de ASIX
2020-2021

1 PARTE I. Gestión del proyecto

1.1 Introducción

Año tras año las tecnologías van aumentando su capacidad de computación, el tamaño va reduciéndose y su precio con el. Cada vez más es más asequible o está más al alcance de todos, ya que el mercado tecnológico es muy competitivo.

Por eso poco a poco van apareciendo nuevas tecnologías en el mercado, ordenadores más pequeños, como pueden ser los teléfonos móviles o smartphones.

Una de estas tecnologías son los microcontroladores o ordenadores de placa reducida, estos se tratan de ordenadores cada vez más pequeños y van mejorando su capacidad computacional casi acercándose a las máquinas tradicionales como los portátiles o los ordenadores de sobremesa.

El objetivo principal de nuestro trabajo es poder demostrar que con una baja cantidad de recursos podemos crear un ordenador haciendo un cluster de máquinas Raspberry Pi 4. Además enseñaremos como hacer un sistema de orquestación de contenedores con Rancher. Este cluster estará formado por varias máquinas, por eso es muy importante tener en cuenta la estructura que diferencia un cluster con Rancher y uno con Kubernetes y que cosas tiene uno que no tiene el otro.

Una idea más a largo plazo o como visión de futuro es aprovechar este cluster de máquinas para instalar un servidor web, en el cual podemos poner diferentes webs con subdominios distintos en los cuales poner contenido diferentes. La ventaja de esta idea es que el cluster será autosuficiente es decir será un servidor de alta disponibilidad, si algo falla esa máquina será borrada y Rancher pondrá en marcha una nueva.

Esta idea final es muy interesante ya que de cara al usuario final podríamos subir diferentes webs. Una función que es muy enriquecedora ya que tendríamos disponibles las herramientas necesarias para poder administrar estas máquinas y por lo tanto asegurar la disponibilidad de los datos y la seguridad de estos.

1.2 Objetivos y justificación

1.2.1 Objetivo general

Poder crear un cluster con pocos recursos usando placas Raspberry Pi que permitan crear un espacio capaz de administrar de forma fácil los servicios expuestos en los contenedores. Mayoritariamente servicios web, como por ejemplo una página web wordpress. Al ser low cost los servicios podrán estar al alcance de cualquier usuario.

1.2.2 Objetivos específicos

- Crear un cluster con 2 placas Raspberry Pi de 4GB de RAM de modelo 4B y máquinas virtuales.
- Crear un cluster con Rancher que sea capaz de desplegar pods personalizados, conocido como “Deployments”.
- Gestionar de forma automática el cluster.
- El cluster ha de ser accesible desde internet independientemente de donde se esté ejecutando.
- Monitorizar los nodos con Prometheus y Grafana.
- Exponer servicios.

1.3 Estrategia y planificación del proyecto

1.3.1 Planificación inicial

Año tras año las tecnologías aumentan su capacidad computacional y su tamaño cada vez disminuye más. Y el precio de este también va siendo más asequible para el alcance de gran parte de la población, ya que el mercado tecnológico es muy competitivo.

Poco a poco entran nuevas tecnologías al mercado, computadoras cada vez más pequeñas como por ejemplo los móviles. Y una de estas tecnologías es Raspberry Pi, una computadora muy reducida que año tras año va mejorando y su capacidad computacional casi se acerca a la de una máquina tradicional de sobremesa.

El objetivo de nuestro trabajo de síntesis es poder demostrar que con pocos recursos podemos crear un ordenador/servidor creando un cluster de 4 Raspberry Pi.

Vamos a crear un servidor web low cost, pero la pregunta es ¿Cómo lo haremos? Utilizaremos kubernetes para poder hacer el cluster. El cluster será de 4 Raspberry Pi donde estarán conectadas entre sí donde una será el master.

El cluster tendrá la capacidad de poder crear contenedores donde se sitúan los servidores web. En estos podremos subir contenido diferente. Para visualizar la monitorización de estas máquinas usaremos Kibana o Grafana. Para administrar las Raspberry y las máquinas virtuales usaremos Ansible. El servidor web será Nginx. Hay un sin fin de puertas que nos abre el software libre y todas las herramientas que usaremos las hemos visto o mencionado en clase.

Esta idea permitirá poder subir diferentes webs a los usuarios finales y nosotros como administradores tendremos las herramientas necesarias para garantizar la seguridad a los usuarios.

1.3.1.1 Puntos de control

Los puntos de control fueron marcados por el profesorado para poder ir mostrando cómo progresaba el proyecto. Los objetivos marcados en cada punto son los siguientes:

- Punto 1: en el primer punto decidimos en que se iba a basar nuestro proyecto, crear un diagrama de gantt para poder tener una organización temporal clara a la hora de conseguir las tareas del proyecto y comenzar a investigar Kubernetes y a la misma vez a poner en marcha esa primera planificación inicial.
- Punto 2: en esta fase uno de los objetivos es tener la memoria y procesos que hemos realizado a medias, y comenzar a tener desarrollado el progreso del proyecto y solucionar problemas relacionados con ello o buscar alternativas. Uno de los primeros objetivos es si seguir el proyecto con Kubernetes era viable y dependiendo de esto cambiar o buscar una alternativa, en este caso k3s.
- Punto 3: el objetivo principal es tener terminada satisfactoriamente tanto la memoria como el cluster.

1.3.2 Planificación final

Una vez decidida la arquitectura de nuestro cluster, que fuese de alta disponibilidad, investigamos qué software escogeremos para cada componente. En cuanto al orquestador de contenedores el escogido fue Rancher, también conocido como k3s. El balanceador de cargas que distribuye las peticiones a los servidores maestros es HAProxy. Para almacenar los estados de los componentes del cluster es MySQL en la base de datos externa. Como administrador externo del cluster es kubectl, y el dashboard que proporciona Kubernetes.

Empezamos configurando la base de datos, para que aceptase conexiones del exterior y no localhost solamente. Después creamos un usuario y una base de datos donde los maestros se conectarán.

El balanceador de cargas fue el siguiente. Teníamos que redireccionar las peticiones de los puertos 6443 (Rancher), 80, 433 y 8080 a los dos servidores que harían como maestros.

Cuando la base de datos externa y el balanceador de carga eran funcionales pudimos añadir e instalar Rancher en los servidores maestros indicando el balanceador de cargas como punto de salida y la MySQL como base de datos externa.

Si los dos servidores maestros están funcionando podemos añadir los dos trabajadores indicando el token del cluster y el balanceador de cargas como punto de salida.

Llegados a este punto el cluster ya es funcional y podemos crear tantos pods como queramos. Pero para facilitar la administración y la monitorización del cluster añadimos una máquina extra con kubectl y el dashboard de kubernetes.

Para poder acceder al cluster usamos un DNS dinámico para linkear la IP del router con el dominio gandalf 9.ddns.net. El siguiente paso fue redireccionar los puertos 2201 para el dashboard y el 2200 para el balanceador de cargas.

1.3.2.1 Visión de futuro

En los puntos de control es interesante ir controlando y supervisando si los diferentes requisitos o objetivos principales han sido cumplidos satisfactoriamente. Una vez vamos cumpliendo los diferentes requisitos ir ampliando cada vez más el proyecto, en cuanto complejidad, requisitos o funcionalidades.

Teniendo en cuenta esto, uno de los primeros requisitos era realizar el proyecto con Kubernetes, pero fue prácticamente imposible debido a los problemas que nos ocasionó. En un futuro sería interesante realizarlo con esta tecnología ya que es mucho más complejo y más profesional para crear un entorno de producción.

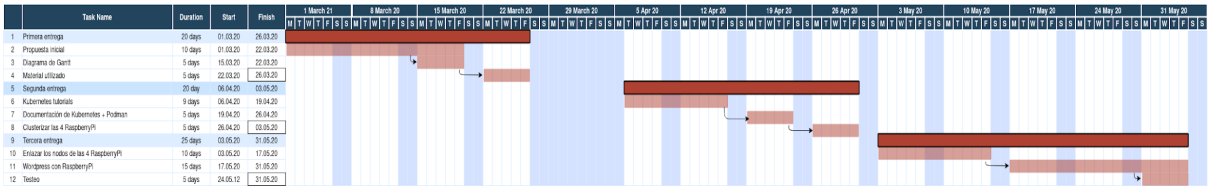
En cuanto al proyecto desarrollado con Rancher tuvimos una serie de problemas de cara a la visión de futuro. Entre ellos monitorizar o convertir todo el proceso automatizado con Ansible y enlazar Prometheus con Grafana.

Conseguimos aplicar Prometheus en el entorno de prueba, pero no lo conseguimos hacer junto a Grafana. Algo que también falló fue la creación del playbook que desplegaría un cluster Rancher. No lo logramos debido a complicaciones con la base de datos.

1.4 Metodología de trabajo

Al inicio del proyecto utilizamos los diagramas de gantt para poder facilitar el análisis y el diseño del proyecto y estructurarlo de una forma acorde. De esta forma pudimos hacer un estudio de la viabilidad del proyecto contemplando los posibles límites de tiempo a los que nos enfrentaremos en los siguientes 2 meses para poder hacer las tareas asignadas.

Este es el diagrama inicial que creamos para poder asignar las tareas:



Una vez empezado el proyecto hemos usado la metodología Scrum. Esta es muy útil para equipos pequeños y es una de las estructuras más sencilla y usadas.
El objetivo era poder mejorar la comunicación entre el equipo y el cuerpo docente, pudiendo hacer un seguimiento cronológico de las tareas realizadas en el proyecto para poder avanzar.

Escoger una metodología acorde a las necesidades del proyecto es muy importante, ya que está optimiza la evolución y el desarrollo del proyecto. El diagrama gantt nos guió al principio para poder tener una guía de las tareas y el tiempo en cuando las teníamos que realizar aproximadamente. En cuanto a la metodología Scrum permitió mantener una comunicación fluida con los miembros del equipo y el profesorado, y poder resolver las dificultades presentadas durante el proyecto o encontrar alternativas.

1.5 Presupuesto

El presupuesto es algo variable, depende de si los diferentes componentes o nodos que interactúan entre ellos están siempre encendidos o si simplemente queremos aislar los “agentes” de los “master”. Con esto lo que me vengo a referir es si queremos que esos componentes o “roles” dentro de la infraestructura de nuestro cluster queremos que sean un dispositivo hardware (Raspberry Pi 4) o en el caso contrario una máquina Virtual, VPN, etc.

Como planificación inicial o primeras pruebas las realizamos dentro de los microcontroladores Raspberry Pi 4, esto es debido a que un esquema con Kubernetes y un cluster no precisa más. Más adelante cambiamos nuestra planificación inicial y cambiamos hacer el cluster con K3S debido a varios factores.

Una vez puesto en contexto todos estos factores que afectan el presupuesto, deberíamos tener en cuenta tanto componentes lógicos (agentes, master, etc), como dispositivos hardware (SD, cables, Raspberry, etc).

En cuanto a dispositivos hardware que interactúan en el proyecto y que por lo tanto juegan un gran papel son:

Componente:	**Cantidad:**	**Precio/unid:**
Raspberry Pi 4 (4GB de RAM)	2	72,90 €
Cables Micro HDMI	2	6,49 €
Cables Ethernet	2	0,60 €
Switch	1	Material del departamento
Fuente de alimentación	2	8.95 €
Pendrive con adaptador microSD	1	6,92 €
Micro SD de 32 GB	2	11,05 €

Estos componentes están desglosados por la cantidad, ya que en el caso de querer aumentar aspectos como potencia simplemente tendríamos que añadir más y aplicar la configuración a nivel de software.

Los dispositivos software que interactúan en el proyecto son muy importantes ya que para tener una infraestructura mínima con k3s necesitamos dos master y como mínimo 2 agentes, en nuestro caso hemos puesto estos cuatro roles o componentes en una misma red aunque lo correcto es separar los diferentes tipos, es decir los master en máquinas AWS (en nuestro caso máquinas Virtuales), de los agentes en RaspberryPi 4:

Componente:	**Precio:**	**Características:**
Master 1	69.0432 $ al mes	t2.large con 8 de RAM
Master 2	69.0432 $ al mes	t2.large con 8 de RAM
Balanceador de cargas	33.408 $ al mes	t2.medium con 4 de RAM
Base de datos	33.408 $ al mes	t2.medium con 4 de RAM
Worker 1	33.408 $ al mes	t2.medium con 4 de RAM
Worker 2	33.408 $ al mes	t2.medium con 4 de RAM

Los precios son calculados a partir de los detalles del producto de la página de Amazon.

Una vez más los componentes dependen de la configuración o la potencia que queremos que tenga nuestro k3s, en este caso hemos utilizado una configuración estándar con lo necesario para poder utilizarlo. Eso no quiere decir que podamos añadir menos agentes o menos máster, sino que a partir de esta configuración estandard de la infraestructura de nuestro k3s hemos hecho un presupuesto pero en caso de querer que sea más seguro, es decir tener más master para añadir más servicios o querer tener más raspberry Pi 4 para tener más velocidad simplemente tendríamos que añadirlos a la idea base y configurarlo, en este aspecto el precio es bastante flexible ya que se adaptara a los módulos o la potencia que queremos que tenga nuestro cluster, en nuestro caso base el presupuesto rondaría en unos 270€.

Es interesante este presupuesto en la “nube” pero finalmente en el proyecto no hemos hecho servir ninguna máquina en la nube.

1.6 Leyes y normativa

Nuestro proyecto se rige por la normativa europea RGPD. Que sitúa el nuevo marco legal en la Unión Europea, es decir, es un conjunto de reglas que hemos de cumplir. Imaginando que ofrecemos el servicio de poder crear y administrar servicios web en nuestro cluster a usuarios de internet donde para acceder los usuarios se registran en nuestra página web, hemos de tener en cuenta las siguiente normas:

- Los usuarios en los pods donde se encuentran los servicios web que están gestionando pueden guardar información personal o otro tipo de data. Esta información está en nuestros servidores pero no podemos acceder a ella. Toda la información es privada.
- Cuando un usuario deja de usar nuestros servicios y elimina sus pods donde se encontraba la información de sus servicios, tenemos la obligación de borrar toda la información y no podemos almacenarla bajo ningún concepto.
- En el caso de sufrir una brecha de seguridad donde la culpa es nuestra tenemos la obligación de avisar a los usuarios y la responsabilidad recae en nosotros.
- A la hora de hacer un login en la página web guardamos la información de los usuarios. Por lo tanto cualquier usuario que lo pida debe de poder recibir toda la información que tenemos almacenada sobre él. O si el usuario lo desea, puede tener la capacidad de modificar los datos que tenemos sobre él si hay alguno erróneo.
- El usuario puede ser capaz de indicarnos que sus datos no van a ser procesados con fines comerciales, por ejemplo para poder hacer campañas de marketing.

Como hemos comentando anteriormente, si hemos sufrido una brecha de seguridad hemos de avisar a los usuarios inmediatamente. Pero nosotros nos podemos respaldar en la ley si la brecha de seguridad no ha sido responsabilidad nuestra y ha

sido ocasionada por una gestión pobre de la seguridad de los servicios web de los usuarios, de esta manera los usuarios si sufren brechas donde la responsabilidad es suya, no podrán demandarnos.

No cumplir las normas de la RGPD que hemos resumido brevemente puede tener sanciones monetarias muy importantes. Evadir estos conceptos legales pueden suponer multas de 20 millones de euros o incluso llegar a pagar una multa del 4% de la facturación anual de la compañía. Estos últimos años súper compañías como Google han sufrido este tipo de sanciones.

2 PARTE II. Ejecución del proyecto

2.1 Análisis de requisitos

Gandalf cumple con las siguientes características:

- El cluster está expuesto a internet en un entorno DMZ sin exponer las demás redes que se encuentren en el domicilio.
- Poder administrarlo con una máquina externa al cluster tanto de forma visual como por línea de comandos.
- Poder monitorizar el estado físico de los nodos.
- Poder crear un cluster de forma automática

2.1.1 Requisitos funcionales

Uno de los primeros trabajos o tareas principales era planificar o pensar cuales iban a ser nuestros requisitos finales de cara al proyecto y dependiendo de esto estudiar si lo que íbamos a hacer iba a ser viable o no, es decir si finalmente íbamos a poder realizar satisfactoriamente el proyecto.

Uno de los primeros objetivos era conseguir que nuestro proyecto con Raspberry Pi 4 estuvieran conectadas entre sí por nodos llamados Worker y Master, con esto lo que queremos conseguir es ofrecer a clientes o a instituciones servicios web en los que tener una granja de servidores a bajo coste y con un alto rendimiento.

Otro de los requisitos funcionales, es que al estar ofreciendo un servicio, que un administrador de servicios sea capaz de monitorizar todo el rendimiento, funcionamiento o capacidad de estas máquinas, nodos y todos los elementos que forman parte de esta granja de servidores.

Y por último es que cada uno de los usuarios tenga la libertad de poder utilizar cada una de las máquinas de esta “granja” de la manera más segura posible, ya sea a niveles de seguridad en privacidad y en posibles ataques al cliente, además de que el usuario final tenga el 100% de confianza con el servicio que ofrecemos.

2.1.2 Requisitos no funcionales

Gandalf cumple con una serie de requisitos no funcionales, esto quiere decir que habrán cosas que el usuario no podrá utilizar pero de cierta manera estarán integradas e indirectamente se esperaran que estén funcionando.

La seguridad y privacidad de los datos del usuario debe de estar garantizada. La interfaz tiene que ser fácil de utilizar, es decir que cumpla con una serie de requisitos mínimos de usabilidad.

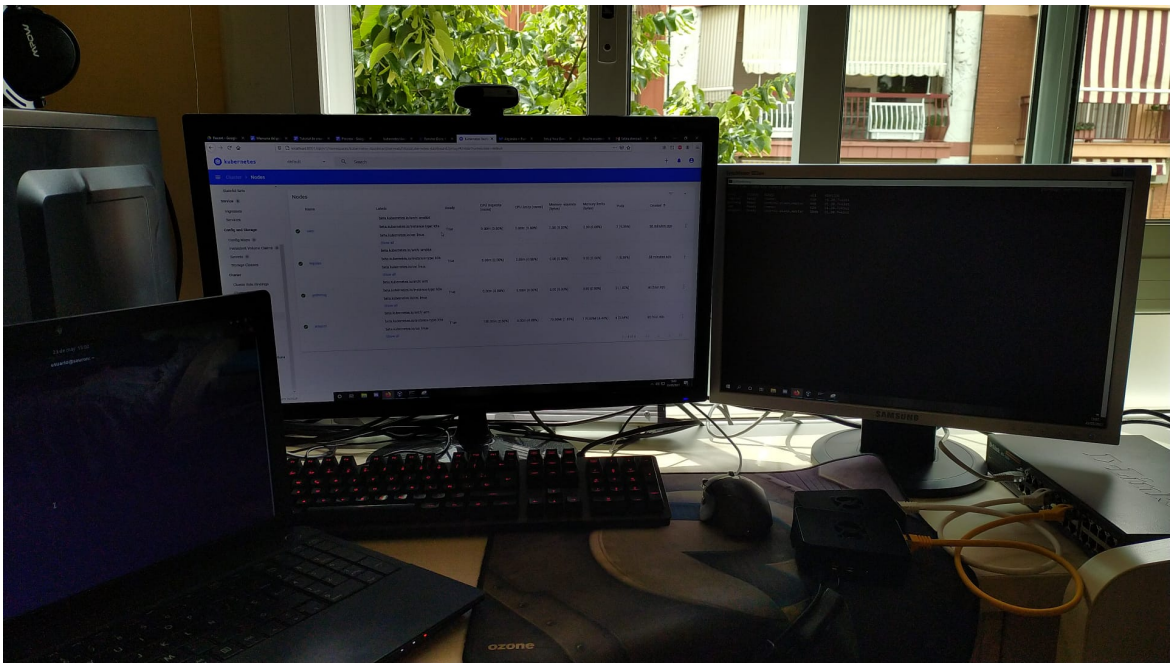
Y por último que sea escalable, es decir tiene que tener la posibilidad de poder aumentar la capacidad y carga de trabajo, además de poder aumentar el tamaño del sistema y la calidad.

2.2 Diseño

Un aspecto muy importante a la hora de realizar este proyecto es tener claro qué estructura va tener nuestro cluster o lo que es más importante, cuál es la estructura interna de una máquina con k3s. En los siguientes apartados hablaremos del diseño físico y lógico de nuestro proyecto.

2.2.1 Arquitectura física

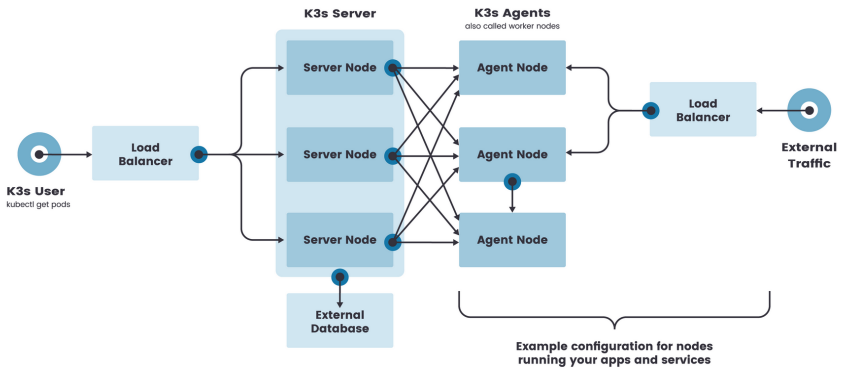
En la arquitectura física encontramos la máquina de escritorio donde se están ejecutando las máquinas virtuales del balanceador de cargas, la base de datos externa y un trabajador. Mientras que las raspberries están conectadas a la red doméstica gracias a un switch D-Link. Finalmente como administrador del cluster y donde podemos visionar el dashboard de k3s usamos un portátil donde además se ejecuta un nodo trabajador adicional.



2.2.2 Arquitectura lógica del cluster k3s

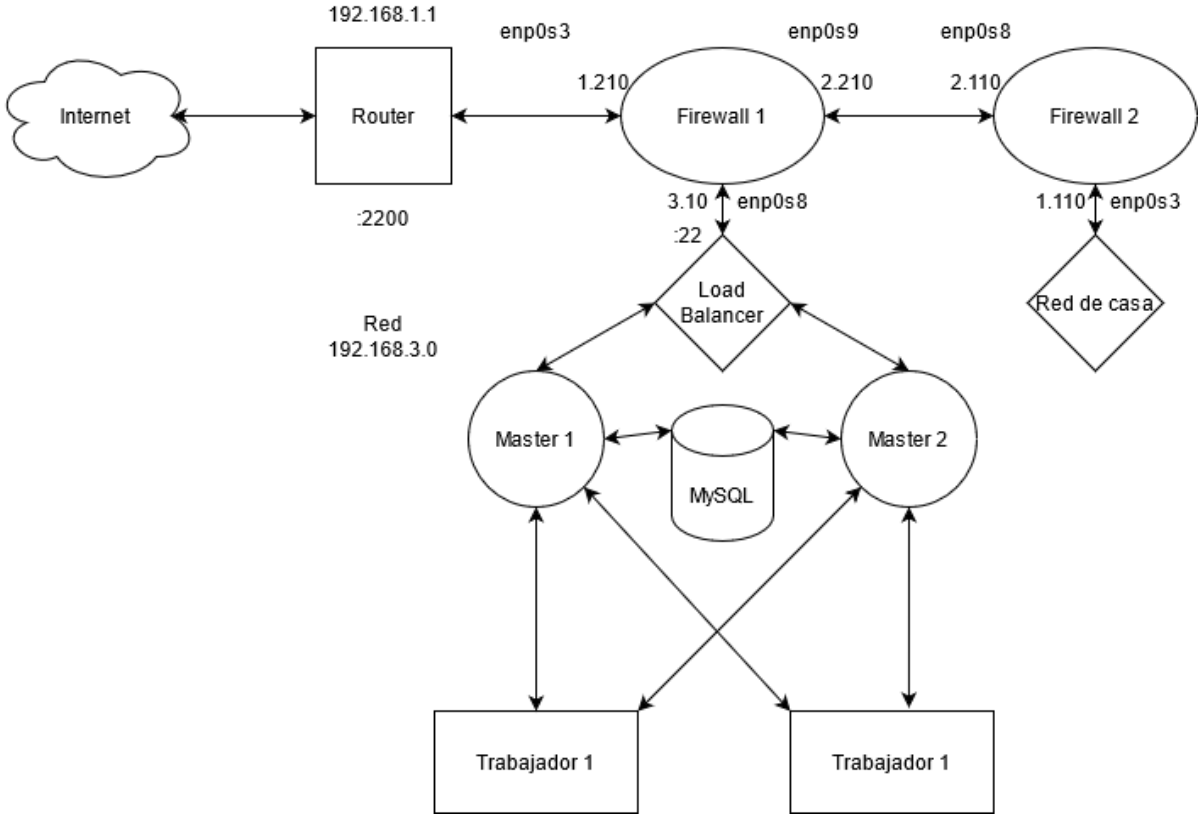
El cluster de alta disponibilidad está diseñado de la siguiente manera: en la primera línea tenemos un balanceador de cargas que distribuye tanto las peticiones a los servicios que puedan estar siendo ejecutados en los servidores agentes o las peticiones de kubectl desde la máquina dashboard (que más adelante hablaremos de ella), como get pods a uno de los servidores master. En los servidores maestro hay una base de datos externa donde conecta con ambos y estos podrán almacenar

el estado entre otros de los pods. Los servidores trabajadores están unidos a los servidores maestro donde se estarán ejecutando los pods de nuestros servicios.



2.2.3 Arquitectura lógica de redes DMZ

También se ha de tener en cuenta las siguientes especificaciones: la accesibilidad desde fuera de la red donde se ejecuta el clúster, y lo más importante, la seguridad. Por eso es importante la implementación de una DMZ.



Pero en nuestro proyecto por falta de máquinas físicas no podemos realizar este esquema físicamente con las raspberry pi ya que no podemos crear tantas redes. Al tener dos máquinas físicas con una máquina virtual en cada una con adaptador puente, al poner una red distinta a la anfitriona, por ejemplo 192.168.3.0. Se pueden ver entre ellas, pero ocurren errores de red. Como por ejemplo paquetes duplicados,

pérdida de paquetes y fallos de conexión totales. Por lo tanto la única manera de implementarla es creando las máquinas virtuales en la misma máquina física y poder crear redes internas.

Este entorno es parecido a uno de producción, ya que a pesar de que estamos creando un entorno de prueba, no podemos omitir la seguridad e integridad de las máquinas. ¿Pero porque este entorno es más seguro? Vamos a empezar por la accesibilidad.

Cuando el router recibe una petición ssh al puerto 2200, la envía al firewall 1 y este hace forwarding de la tarjeta enp0s3 a la enp0s8 y le envía la petición ssh al load balancer por el puerto 22. De esta manera podemos acceder a la red de los servidores y exponer además otros puertos. Todo gracias a que en el router estamos utilizando un DNS dinámico que liga la IP pública con el dominio gandalf9.ddns.net. Entonces si la IP pública del router cambia no hay problema porque se liga al dominio.

Para evitar que podamos pasar de los servidores a la red de casa, hemos de indicar al firewall que todos los paquetes que vengan de la enp0s8 con destino a la red 192.168.1.0 sean dropeados. En una regla de filter forward.

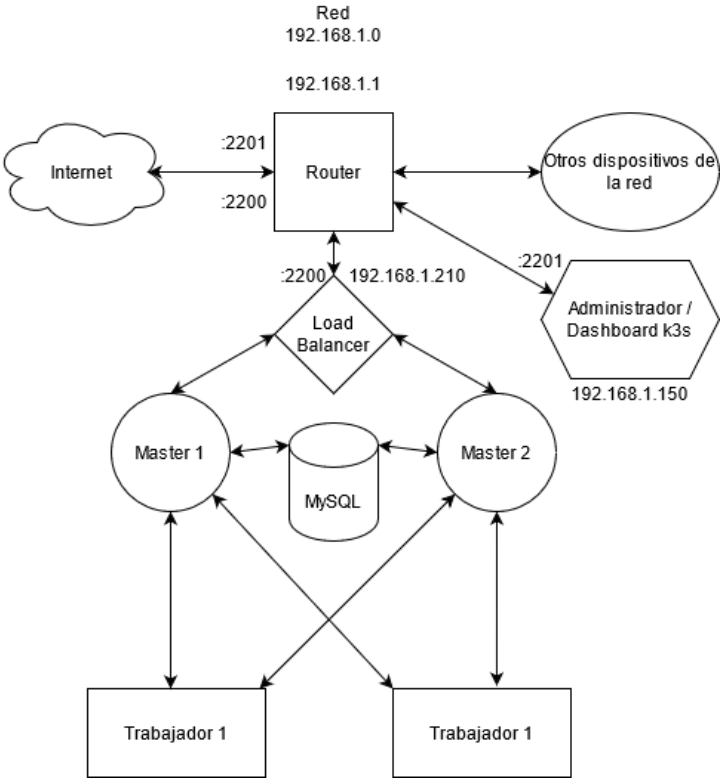
Para evitar que desde la red de casa puedan acceder a la red de los servidores hemos de dropear todos los paquetes que provengan de la tarjeta enp0s3 vayan a la tarjeta enp0s8 con una ip de destino del rango 192.168.3.0. Es una regla filter forward.

Para que la red 3.0 y la 1.0 tengan internet en el firewall 1 hemos de hacer masquerade a las IPs del rango 1.0 y 3.0 que salgan por la tarjeta enp0s3 hacia el router.

Con esta arquitectura de red si alguien con malas ideas accede a la red donde se encuentra el cluster, resultará prácticamente imposible acceder a la red de casa. Impidiendo que puedan acceder al ordenador personal, móviles, etc.

2.2.4 Arquitectura lógica de redes con Raspberry Pi

En el apartado anterior hemos explicado el problema que hemos tenido para poder implementar las raspberries pi con la DMZ. Por tanto si queremos implementar las raspberry físicamente hemos creado esta arquitectura de red.



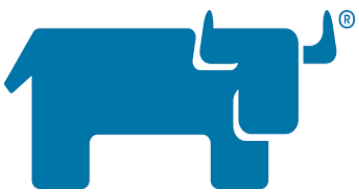
Se puede seguir accediendo al balanceador de cargas desde el exterior de la red mediante SSH conectando con el router, pero sin un intermediario, el firewall 1.
Además en este esquema implementamos el dashboard de k3s al que podemos acceder por el puerto :2201 vía ssh.

Todos los componentes del cluster están conectados en la misma red de casa. Por tanto si alguien accediese podría intentar escalar hacia otros host de la red como el ordenador personal, móviles, etc.

Finalmente implementamos esta topología lógica con las raspberries como servidores maestros. Debido a que era la única que podíamos implementar usando las raspberries.

2.3 Tecnologías

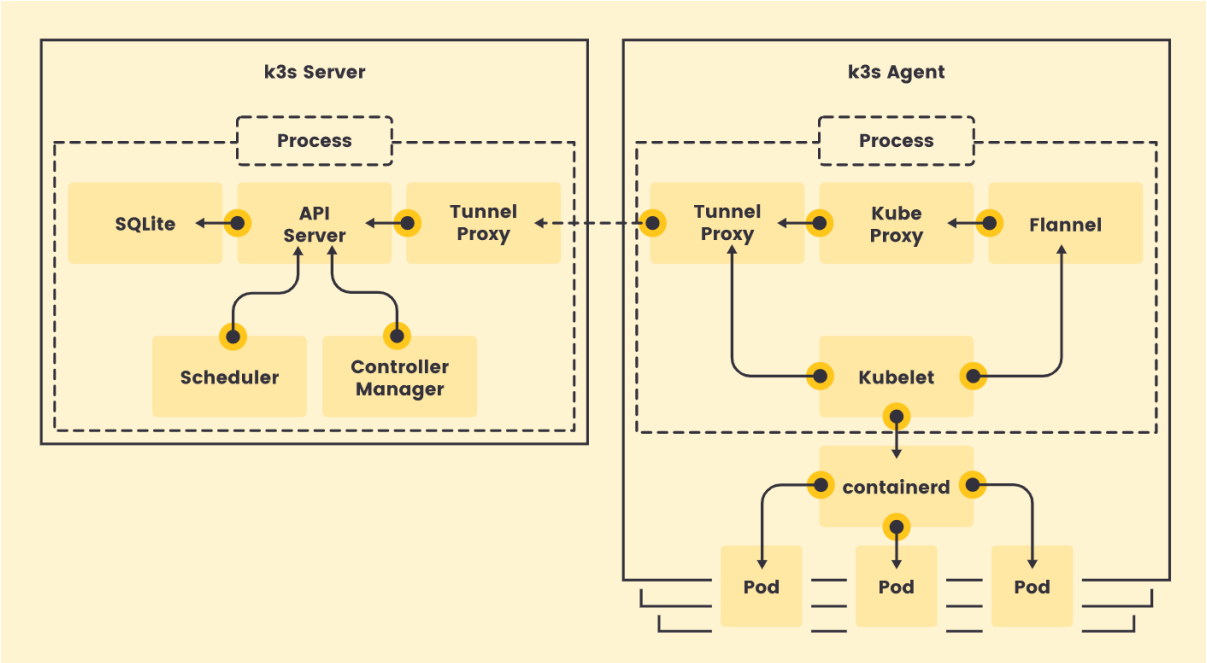
2.3.1 k3s



k3s es un binario de 40MB que contiene todo el software libre necesario para realizar a priori las mismas funciones que su hermano k8s a menor escala, es decir orquestar containers para poder automatizar el despliegue de servicios y poder administrarlos.

Este contiene los componentes de kubernetes: kube-apiserver, kube-controller-manager, kube-scheduler, kubelet, kube-proxy. También contiene dependencias externas que descarga y ejecuta en la instalación como containerd para la creación y administración de pods, flannel como CNI (capa de red), CoreDNS como resolución de nombres y traefik como ingress controller.

Esquema del funcionamiento lógico:



Como podemos ver en este esquema de cómo funciona k3s los pods se ejecutan en los trabajadores, pero ya veremos que no es del todo cierto porque en la creación del cluster hay ciertos pods que los masters necesitan encendidos para que funcione correctamente.

El kubelet se comunica con el servidor master a través del tunel proxy que crea kube-proxy para comprobar que los pods se están ejecutando en un nodo. Y por último vemos que containerd es el responsable de la creación de los pods. Aunque es posible hacer la instalación de k3s con docker.

Un ejemplo práctico de cómo estas tecnologías se coordinan es a la hora de el mantenimiento de los pods entre master-worker. La vida de los pods es reducida, por lo tanto cuando un pod “muere” en el nodo trabajador, el servidor maestro gracias al tunel proxy puede ver el estado de estos con kubelet. Cuando kubelet ve que el pod se ha “muerto”, rápidamente crea otro.

K3s es una herramienta de software libre increíble para empezar en este mundo para comprender como funciona a nivel interno y teórico un orquestador de contenedores.

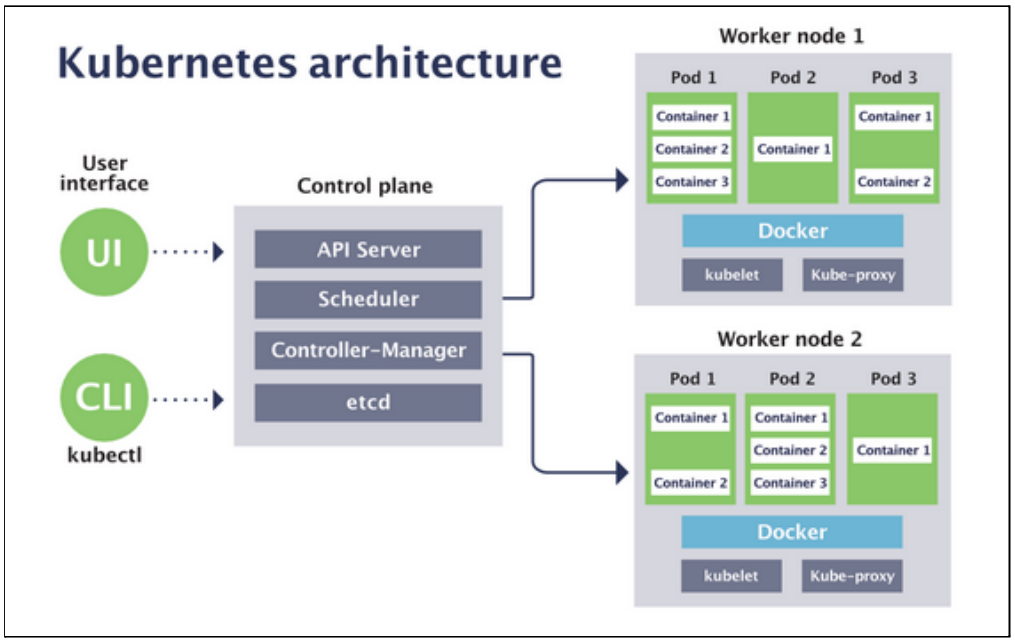
2.3.2 Kubernetes

Kubernetes es un poco diferente a rancher a nivel interno, porque a la práctica la función es la misma. Orquestar containers para poder automatizar el despliegue de servicios y poder administrarlos.

En k8s los nodos son mayoritariamente trabajadores, no hay distinción entre servidor maestro y trabajador tan clara como en k3s. Encontramos un nodo “maestro” que orquesta a los demás nodos, por lo tanto el único nodo distinto es este. El resto de nodos serán idénticos. Podrán administrar y mantener la carga de trabajo de los pods.

Kubernetes para poder funcionar usar los “hermanos kube” (kubectl, kubelet, etc) que en unos apartados más adelante hablaremos en profundidad sobre ellos, pero vamos a hacer una pequeña introducción. Kubeadm se encargará de la instalación y comprobar aspectos de los demás nodos. Kubelet permite al cluster saber si los contenedores se están ejecutando correctament en pods y kubectl permite ejecutar comandos en el cluster, comprobar los nodos, pods, etc.

Esquema del funcionamiento lógico:



Kubernetes para la creación y administración de los contenedores utiliza la tecnología Docker. Podemos ver el ahorro de software que hace k3s escogiendo containerd en vez de docker. Siendo docker la línea de comandos y containerd el demonio que procesa y maneja los containers.

Por defecto en la instalación kubernetes no tiene los componentes funcionales para poder crear un clúster funcional, sino que has de instalar manualmente una capa de red, un load balancer, almacenamiento persistente para los pods, etc.

2.3.3 Rancher vs kubernetes

Pese que en nuestra planificación inicial decidimos utilizar Kubernetes ya que es mucho más potente que k3s, más adelante debido a problemas que tuvimos valoramos la decisión de si era viable seguir con Kubernetes o K3s. En esta tabla comparamos un poco las diferentes ventajas y inconvenientes entre ellos:

Rancher (k3s)	Kubernetes (k8s)
Software de orquestación de contenedores creado a partir de kubernetes.	Software de orquestación de contenedores originalmente desarrollado por Google.
La complejidad de k3s es reducida.	La complejidad de k8s es elevada.
Poner en marcha k3s es sencillo, debido a que en la instalación implementa software adicional.	Poner en marcha k8s es complejo ya que has de instalar y configurar el software necesario manualmente.
K3s está enfocado y pensado para entornos de prueba.	K8s está enfocado a entornos de producción muy grandes.
Los recursos que necesita son reducidos.	Los recursos que necesita son más elevados.
Permite tener múltiples nodos maestro.	Permite tener múltiples nodos maestro.

2.3.4 K3s

2.3.4.1 Coredns

CoreDNS es un servidor DNS escrito en Go que puede ser usado en diferentes entornos gracias a su flexibilidad. Por debajo de la versión 1.13 de kubernetes por defecto k3s usaba kube-dns pero ahora encontraremos a partir de la versión 1.14 CoreDNS.

Es importante que los componentes del cluster puedan resolver tanto nombres internos como externos. La configuración por defecto en la instalación de k3s es suficiente para que el cluster sea funcional.

2.3.4.2 Flannel

Por defecto K3S integra Flannel como CNI y usa las tarjetas VXLAN como backend predeterminado. La CNI define cómo se pueden ejecutar los complementos del cluster para configurar las interfaces de red de los pods.

Flannel crea la capa de red gracias a las tarjetas VXLAN que satisface las necesidades de k3s. Gracias a esta capa de red los pods se podrán comunicar con

los pods de otros nodos sin tener que habilitar el NAT, todos los nodos trabajadores pueden comunicarse con los nodos de otros trabajadores y la IP que un contenedor ve en el mismo es la misma IP que ven los otros contenedores.

La configuración por defecto en la instalación de k3s es suficiente para que el cluster sea funcional.

2.3.4.3 Traefik

Traefik es un controlador de ingresos moderno que funciona mediante HTTP como reverse proxy y balanceador de cargas para poder hacer deployments de servicios en el cluster de manera sencilla. Gracias a Traefik la complejidad de la red, poder hacer deploys y tener servicios corriendo será mucho más sencillo. Por defecto el ingress controller de Traefik usa los puertos de la máquina 80,443 y 8080.

Cuando recibimos una petición del exterior será capaz de comunicarnos con el destinatario de la petición. Sin el proxy inverso solo tendríamos unos nodos, y al recibir la petición no habría forma de saber cómo llegar al destino. Al fin y al cabo es un software que puede entender todas las reglas de entrada que hay y ejecutarlas.

2.3.4.4 MySQL

MySQL es una base de datos relacional de código abierto. Por defecto k3s usa etcd como base de datos integrada, pero nosotros hemos creado una arquitectura donde MySQL es nuestro sistema de base de datos externo.

El papel que juega MySQL en una estructura de k3s es que los componentes del cluster para ir almacenando su estado, necesitan de una base datos.

2.3.4.5 HAProxy

HAProxy es un balanceador de cargas software libre escrito en C que ofrece un balanceador de cargas de alta disponibilidad y proxy server para conexiones tanto TCP como HTTP, y distribuye estas conexiones hacia los servidores que le indiques en la configuración.

Tiene la reputación de ser muy rápido y eficiente, por eso lo hemos escogido como nuestro balanceador de cargas.

2.3.4.6 Nginx

Nginx aparte de ser un servidor web como todos sabemos, también puede ser utilizado como balanceador de cargas. Es un

eficiente balanceador de cargas HTTP que distribuye estas peticiones a los servidores que le indiquemos.

Está enfocado a los servicios webs, por eso la versión “estándar” no incluye conexiones TCP.

2.3.4.7 Etcd

Etcd es una base de datos consistente que provee una forma fiable de almacenar los datos a los que una máquina o un grupo de máquinas han de acceder. Puede tolerar fallos en las máquinas, incluso en los nodos maestros.

Es por esto que k3s implementa por defecto etcd como base de datos por defecto. Kubernetes guarda la información del cluster en la base de datos como “backup”. Por eso es importante de vez en cuando si usamos etcd hacer backups de esta.

2.3.4.8 Helm

Helm es una herramienta ideal para poder manejar el cluster de k3s con mayor facilidad. Ya que helm te facilita poder instalar y actualizar cualquier aplicación de nuestro cluster kubernetes. De esta forma podemos hacer deployments genéricos como por ejemplo un servidor web nginx con tan solo un comando.

Helm puede hacer esto gracias a los “charts”, que son ficheros que describen recursos de kubernetes. Estos “charts” están situados en directorios en particular, donde cada “chart” necesitará en el directorio un fichero.yaml que defina los campos.

2.3.4.9 Containerd

Containerd es un demonio software libre disponible para Linux que gestiona el ciclo de vida de los contenedores del host. Puede almacenar información dentro de los contenedores, ejecutar acciones dentro de ellos, supervisarlos entre otras muchas. Es la herramienta que utiliza k3s como creador de pods. Aunque ya hemos visto que podemos cambiar esta por docker. Docker y containerd son similares, ya que containerd es una parte de Docker Open Source project. Docker es el comando que usa el usuario para por ejemplo crear un contenedor. Containerd es quien descarga la imagen, controla la red y el almacenamiento y usa “runc” para poder arrancar el contenedor que se acaba de crear.

Además se implementa muy bien y con mucha facilidad con herramientas que usa k3s como kubelet para poder comprobar y administrar el estado de los pods en los servidores trabajadores.

2.3.4.10 Prometheus

Es una herramienta de software libre que sirve para monitorizar y crear alertas de eventos. Además crea gráficos en tiempo real y es una herramienta especializada en kubernetes y en hacer métricas sobre esta tecnología. Este va directamente a los componentes a buscar las métricas que les interesa.

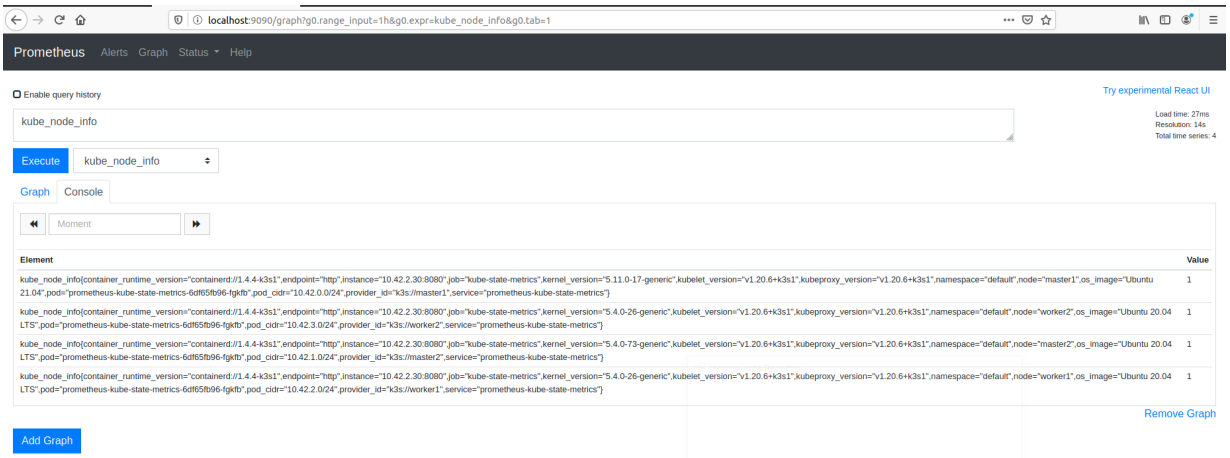
Esta herramienta es esencial para este tipo de proyecto ya que en cuanto contenedores se refiere, se crearán una gran cantidad de estos a la vez. Para monitorizar una gran cantidad de máquinas es interesante ya que podremos ver si se cumplen aspectos a nivel de rendimiento del sistema, alta disponibilidad, etc.

Es interesante tener una herramienta de este tipo ya que además de poder visualizar los datos de nuestros pod y nodos de una manera mucho más sencilla o visual y nos ofrece una información adicional de estos.
Estos datos son recopilados a tiempo real y están clasificados por métricas, por ejemplo: ver el estado de los nodos. En este caso dentro del dashboard de rancher podemos ver aspectos como RAM y CPU consumidas o la cantidad de deployments que hay en cada uno de esos nodos, pero con prometheus podemos ver muchas más métricas que con el dashboard normal no podemos ver.

Además Prometheus tiene un inconveniente y es que la forma que tiene de visualizar los datos es una gráfica por métrica, es decir si filtramos solo poder graficar o ver a tiempo real por aquello que hemos filtrado.

Una gran alternativa es utilizar Grafana para todo sea mas tipo dashboard, en el que se ven diferentes métricas con diferentes gráficos y el Prometheus utilizarlo para centralizar y almacenar esas métricas que a diferencia del dashboard de Rancher no muestra.

Fotografía del dashboard que implementa Prometheus desplegado en nuestro entorno de prueba:



2.3.4.11 Grafana

Es una herramienta de software libre que principalmente apareció como componente del Kibana.

Sirve para visualizar datos o hacer gráficos a tiempo real a partir de unas métricas con datos que se recogen de otros programas como pueden ser un ElasticSearch, el Sistema, Prometheus, etc.
Además Grafana presta una funcionalidad a Prometheus y es la de dar soporte a esta app, de esta manera se puede juntar las ventajas de estas dos aplicaciones que hacen gráficas, es decir aprovechar la potencia de Prometheus con la gran cantidad de métricas que tiene relacionadas con el Kubernetes y aprovechar la gran potencia que tiene Grafana para crear gráficos y mostrarlos.

Al fin y al cabo esta herramienta es una vía para poder visualizar los datos de una manera mucho más sencilla y amigable. Entre ellas tiene funcionalidades de crear dashboards, monitorear y recopilar datos de manera nativa de lugares como Prometheus o Elasticsearch.

2.3.5 Kubernetes

2.3.5.1 Docker

Es un proyecto de código abierto que sirve para automatizar el despliegue de aplicaciones dentro de contenedores, esto lo que hace es generar una capa de abstracción y para automatizar diferentes aplicaciones en múltiples sistemas operativos.

Este te permite diferentes funcionalidades como pueden ser crear, implementar, copiar, mover estos contenedores lo que hace que aplicaciones en la nube sean mucho más óptimas.

Esta tecnología funciona con el kernel de Linux y algunas funcionalidades de este, como pueden ser cgroups o pueden ser los namespaces, de esta manera se consigue que cada uno de los procesos o contenedores se puedan ejecutar de manera independiente.

Además esta tecnología utiliza un modelo de implementación basado en imágenes, esto hace que sea más fácil el compartir, aplicaciones, servicios.
Al principio Docker utilizaba o estaba basado en los contenedores propios de Linux, LXC. Estos contenedores

2.3.5.2 Metal-lb

A la hora de crear o implementar un cluster de Kubernetes fuera de servicios como AWS o similares, este necesita una forma de exponer el servicio fuera del cluster. Esto es lo que hace el MetalLB implementar un balanceo de carga para los clústeres de Kubernetes con protocolos de enrutamiento estándar.
Un buen ejemplo es lo que hace un proveedor de un entorno en la nube, estos crean y asignan IP y la responsabilidad forma parte de ellos, en este caso la responsabilidad es del Metal-lb, este asigna una IP de un grupo de direcciones que tiene configurado. Es decir lo que hace es repartir los recursos o el trabajo entre las máquinas que hay disponibles, una vez se le asigna la IP externa este redirige el tráfico desde la IP externa del cluster, para ello utiliza protocolos de red como por ejemplo ARP o BGP.
Cuando instalamos Metal-LB en kubernetes se crean dos pods, controller y speaker. Controller es el encargado de asignar IP y el speaker es el demonio que hace que el controller pueda comunicarse y sea alcanzado.

2.3.5.3 Weave Net

En kubernetes Weave Net crea una red virtual que conecta los contenedores Docker y permite que se descubran entre ellos automáticamente. Gracias a esto los servicios que hagamos en el cluster podrán ser expuestos al mundo exterior independientemente de en qué nodo se encuentren.

Esto es posible gracias a que Weave Net crea una capa de red de nivel 2 utilizando las características físicas de nuestro Linux y un demonio configura la red y administrar el enrutamiento entre las diferentes máquinas.

Weave Net implementa un micro servidor DNS en cada nodo. Esto nos permite conectar con los contenedores simplemente por el nombre, esto permite al balanceador de cargas acceder a diferentes contenedores con el mismo nombre.

Además también hay que añadir que se implementa muy bien con docker, ya que gracias a este Weave Net no requerirá de un almacenamiento externo ya que se incluirá como un complemento de red de docker.

Es conocido por ser rápido, ya que usa un algoritmo para detectar en cada momento cuál es el camino entre los dos hosts más rápido y escogerlo.

2.3.5.4 Software externo al cluster

2.3.5.4.1 Ansible

Ansible es una herramienta de software libre que facilita la configuración y administración de los sistemas de nuestra red. Es un software de orquestación, gestiona a los sistemas gracias a SSH y no requiere software adicional para poder realizar las gestiones. Los módulos trabajan sobre JSON y la salida estándar puede encontrarse en cualquier lenguaje. En cuanto a la configuración nativamente utiliza YAML.

Ansible para saber en qué sistemas ha de configurar o administrar dispone del inventario. Aquí indicamos el grupo y las IPs que pertenecen a estos. Una vez tenemos el inventario organizado dispone de playbooks. Donde se despliega la configuración que vas a realizar en algún grupo del inventario.

2.3.5.4.2 Nftables

Es un programa que actúa como firewall, es decir filtrar paquetes y clasificarlos, es una combinación de componentes del núcleo de Linux. Integra las funciones de filtrar paquetes, NAT y otras formas de manejar paquetes de red.

2.3.5.4.3 IPFire

Es un programa de software libre que puede actuar como router y firewall. Añade un panel de control web para poder administrar las funciones del router y las reglas del firewall. Empezó siendo un fork de IPCop. Permite instalar addons para poder añadir servicios adicionales.

2.3.5.4.4 Dhcpcd

Es un cliente DHCP que actúa como demonio y es de código abierto. Se encarga de dar direcciones IP y opciones de configuración a ordenadores y estaciones de trabajo en red.

2.3.5.5 Los hermanos “kube”

2.3.5.5.1 Kube-apiserver

El servidor API de Kubernetes valida los datos de los objetos de la API, este incluye pods, servicios, controladores y mucho más. El servidor API brinda servicios a las peticiones REST y nos proporciona la interfaz para ver el estado de cómo interactúan los componentes de nuestro cluster. Esto lo puede hacer gracias a que kube-apiserver expone la API de kubernetes.

2.3.5.5.2 Kube-proxy

Kube-proxy es un proxy de red que se ejecuta en cada nodo de nuestro cluster, sea maestro o trabajador. Este mantiene las reglas de red entre los nodos, hecho que hace que podamos comunicarnos con los pods de los servidores del cluster.

2.3.5.5.3 Kube-scheduler

Kube-scheduler está pendiente de los pods que huerfanos que se crean en el cluster, es decir que no tienen un nodo asignado y éste selecciona en qué nodo del cluster se ejecutará. Hay factores que influyen en la decisión, como si requiere componentes individuales o grupales, políticas de hardware o software y muchas más. No selecciona los nodos al azar.

2.3.5.5.4 Kube-controller-manager

Kube-controller-manager ejecuta los procesos del controlador. Cada controlador es un proceso separado y separados en diferentes tipos. Los más comunes son:

- Controlador de nodos: es el encargado de darse cuenta y responder cuando algún nodo del cluster ha caído.
- Controlador de trabajo: busca tareas únicas que hayan de ser ejecutadas y crea un pod para que puedan hacerlo hasta que acaben la tarea.
- Controlador de endpoints: se une a los servicios y algunos pods para complementarlos.
- Controlador de servicios y tokens: crea logins y tokens para el acceso a la API para poder crear nuevos espacios de nombre.

2.3.5.5.5 Kubelet

Kubelet está ejecutándose en cada nodo para comprobar que los contenedores se están ejecutando correctamente en un pod. En k3s vemos que kubelet se comunica desde los servidores master con los trabajadores gracias al tunel proxy que crea kube-proxy y va enviando al master la información que recopila en los servidores trabajadores. Cuando ve que un pod se ha caído, al momento lo vuelve a crear.

2.3.5.5.6 Kubectl

Kubectl es la herramienta de líneas de comandos de kubernetes. Esto nos permite tirar comandos hacia nuestro cluster kubernetes. Además nos permite crear aplicaciones, inspeccionar, manejar los recursos de nuestro cluster e incluso ver los logs del cluster. Es decir, con kubectl podemos administrar remotamente el cluster, es por eso que en nuestra infraestructura final tenemos una máquina donde tenemos el dashboard de k3s y el kubectl, para poder administrar remotamente nuestro cluster con una máquina que no forme parte de este.

2.3.5.5.7 Kubeadm

Kubeadm nos permite crear un cluster de Kubernetes con las características mínimas, ya que antes de instalarse kubeadm hará ciertos tests en las máquinas para ver si es viable la instalación de Kubernetes. Aparte de esta función también se encarga de otras cosas, como las actualizaciones del cluster. Kubeadm solo será instalado en la máquina controladora del cluster Kubernetes. En Rancher no encontramos kubeadm en la instalación.

2.4 Componentes

2.4.1 Raspberry Pi 4 + Model B

Las Raspberries son muy útiles para tener un sistema donde podrá soportar la gran parte de la carga computacional de nuestro proyecto, ya que cada máquina está dotada de 4GB de RAM. En estas pequeñas máquinas ejecutamos los servidores maestros. Si cupiese la posibilidad de tener un presupuesto más grande todas las máquinas que usamos en este proyecto serían Raspberry, abogando por la filosofía del proyecto. Intentar hacer un cluster funcional, low cost, con software libre y accesible para la mayoría de las personas.
Pero desgraciadamente el proyecto usa muchas máquinas (base de datos, 2 masters, 2 trabajadores, un balanceador de carga y un firewall. El dashboard perfectamente puede ser la máquina nativa) y el presupuesto podría superar los 500€. Es por eso que recomendamos que las Raspberries sean usadas para los servidores maestro o trabajadores.

2.4.2 K3s Master

En rancher el maestro es el encargado de orquestar y controlar a los trabajadores. Tiene los componentes necesarios para poder comunicarse y recibir información de los trabajadores, como por ejemplo si están encendidos los pods. O mover los pods creados por los “deployments” a un nodo trabajador. La metodología es clara, el maestro administra que todo funcione correctamente y el trabajador ejecuta la carga de trabajo de los pods.

En los nodos maestros aplicamos un “taint”, es decir, indicamos que no queremos que se ejecute ningún nodo que no sea imprescindible para el funcionamiento de este. Como antónimo podemos encontrar “affinity”, que efectivamente es lo contrario. Indicar que en este nodo queremos atraer a x pods. Aplicando el taint conseguimos que toda la carga de trabajo de servicios se ejecute en los trabajadores.

Imágen de las dos Raspberries Pi donde se ejecutan los nodos maestros conectadas al switch:

En la fotografía las dos Raspberries solo tienen conectado el cable de red.

2.4.3 K3s Worker

En rancher el trabajador es el encargado de aguantar la carga de trabajo de los pods y servicios. A la hora de hacer un “deployment” en el cluster automáticamente será movido a un nodo trabajador, gracias al maestro.

2.4.4 Load Balancer

Un load balancer o lo que es lo mismo un equilibrador de carga se refiere a distribuir de manera eficiente el tráfico de una red y es utilizada en muchos servidores o granjas de servidores.

En este caso la idea es utilizar este load balancer en nuestra granja de servidores para que este actúe como policía. Este tendrá la potestad de enrutar diferentes solicitudes de diferentes clientes (en este caso workers y agents), de esta manera conseguimos satisfacer a estos con la máxima velocidad y disponibilidad ya que cuando un trabajo esté sobrecargado simplemente este policía reparte la carga de trabajo, es decir subir o bajar el rendimiento a conciencia.

Un balanceador de carga tiene varias funcionalidades, entre ellas estas:

- Distribuye solicitudes de clientes y la carga de la red.
- Alta disponibilidad y solicitudes a servidores en línea.
- Flexibilidad de agregar o quitar servidores bajo demanda.

2.4.5 Base de datos externa

En nuestra arquitectura la base de datos es externa, a pesar de que la base de datos no se esté ejecutando en un nodo o en todos los nodos del cluster, es una parte muy importante del proyecto. Ya que esta guarda el estado de todos los componentes del cluster.

2.4.6 TSDBs

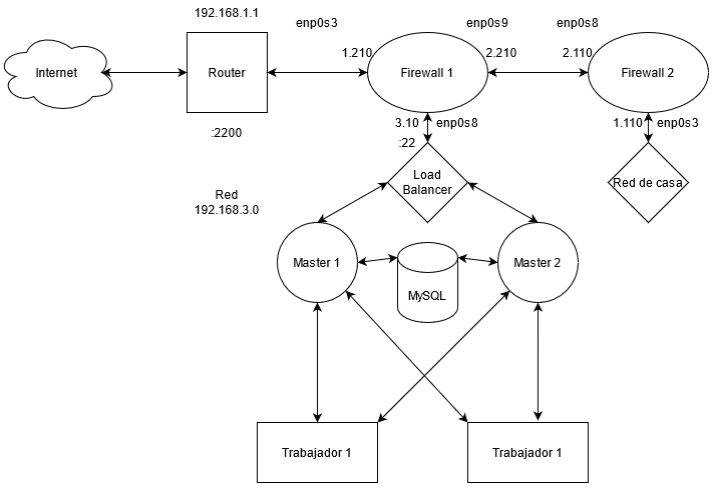
Son base de datos de series temporales en las que se pueden leer y escribir datos a alta velocidad. Es decir, podemos manejar grandes cantidades de información para mejorar la consulta y análisis de datos, como por ejemplo en gráficas.
Muchos de estos programas como Prometheus son útiles para mostrar muchos datos de estas granjas de servidores o en nuestro caso containers en los que podemos visualizar muchos datos con mayor frecuencia y hacer más fácil la seguridad y la monitorización de un sistema de producción como puede ser nuestro rancher.

Estos sistemas tienen un gran cantidad de beneficios, entre ellos:

- Alto rendimiento
- Rentabilidad
- Grandes cantidades de datos analizados de manera simultánea.
- Fácil de utilizar

2.4.7 DMZ

La finalidad de este proyecto es hacer un entorno de prueba lo más similar a uno de producción, para aprender cómo funciona Rancher y Kubernetes. Podemos aplicar los conocimientos que hemos adquirido en redes para poder hacer un entorno seguro y similar a un entorno de producción y hacer una DMZ. De esta manera las peticiones que lleguen a nuestro router por puertos concretos solo podrán acceder al cluster y no a la red de casa. Pero desde la red de casa puedes salir a internet.



Es posible gracias a la implementación de los firewall, ya que estos bloquean el tráfico que nosotros indicamos y permiten acceder desde redes internas a internet.

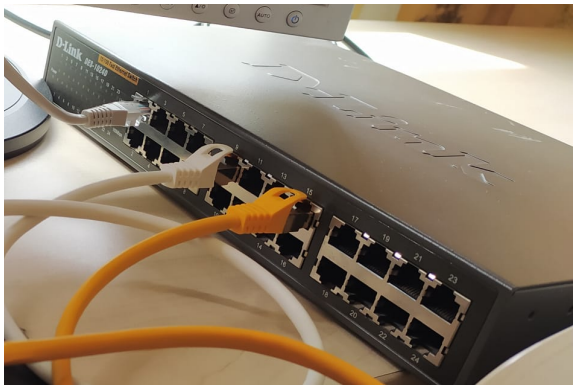
2.4.8 Orquestrador

En este proyecto tenemos muchas máquinas que administrar, la idea del orquestador es poder automatizar el proceso de administración gracias a los playbooks de ansible. Pero además hemos creado un playbook que genera un cluster k3s muy parecido al de nuestro proyecto final en minutos. Solo hemos de tener las máquinas con las IPs correctas y encendidas. La arquitectura lógica tendrá un balanceador de cargas que dirige hacia los dos servidores master, donde hay dos servidores trabajadores y una base de datos externa.

2.4.9 Switch

Con el switch podemos conectar de manera sencilla los componentes del cluster a la red. En este caso son dos Raspberries Pi. Pero el switch que tenemos tiene 24 puertos, menos 1 para conectar el cable que proviene del router, eso hace un resultado de 23 conexiones.

Fotografía del switch que se encuentra en nuestro domicilio:



El switch está conectado al router mediante un alargador de red PLC. Debido a que el router y el switch se encuentran en diferentes habitaciones:

2.4.10 Terminologías de containers en k3s

2.4.10.1 Pods

Un pod es un grupo de containers que funcionan conjuntamente y es la unidad más pequeña que kubernetes puede administrar. Los pods tienen una IP única que comparten con el resto de contenedores que hay dentro de este, al igual que los recursos hardware. Gracias a estos factores podemos tratar a todos los

contenedores de dentro del pod como uno. Pero no es obligatorio tener más de un contenedor dentro de un pod, podemos hacerlo cuando en ese contenedor solo hay un proceso ejecutándose y no requiere la ayuda de ningún otro proceso (bases de datos, etc).

2.4.10.2 Deployments

En los deployments podemos definir los parámetros de los pods que queremos lanzar, con qué contenido y cuantos pods se generarán a partir del lanzamiento. Por ejemplo, podemos hacer un deployment de una imágen nginx, que escuche en el puerto 80 y que se generen 10 pods iguales.

2.4.10.3 Servicios

Los pods no tienen una vida larga, por eso su contenido y entre ellos la IP no serán persistentes. En cuanto muere k3s crea otro pod nuevo reemplazando el antiguo. Es por eso que si queremos que un pod tenga un nombre y IP persistentes el servicio expone siempre la misma IP y nombre sobre estos pods, a pesar que los pods seguirán muriendo y resucitando continuamente.

3 PARTE III. Desarrollo del proyecto

3.1 Desarrollo

Las tareas relacionadas con el desarrollo serán llevadas a cabo con máquinas virtuales y Raspberry Pi. La idea es poder simular cada uno de los roles de una estructura con k3s dentro de cada una de estas máquinas, workers en máquinas virtuales y los master en las Raspberry Pi.

La idea a largo plazo o como visión de futuro es que pese al bajo presupuesto que tenemos que sea posible poder reemplazar esas máquinas por máquinas de AWS y poder monitorizar la granja de servidores. En el presupuesto hemos recreado la posibilidad de que en el caso de utilizar AWS tener diferentes máquinas y el precio de cada una de ellas, lo bueno de esto es que nos podríamos ahorrar o mejorar varias cosas como por ejemplo el balanceador de carga (servicio que ofrece AWS) y la accesibilidad a las diferentes instancias (ssh a las diferentes instancias).

3.2 Problemas encontrados en k8s

3.2.1 k8s docker

Al principio del curso durante la instalación de Kubernetes nos encontramos con el siguiente problema. Por defecto el SO en Raspberry Pi no permite al kernel limitar los recursos del sistema haciendo que el rendimiento de Docker aumentase.

Podemos ver los warnings ejecutando en la raspberrypi *sudo docker info*:

```
WARNING: No swap limit support
WARNING: No cpu cfs quota support
WARNING: No cpu cfs period support
```

Para poder solucionarlo hemos de hacer que docker cambie del grupo cgroups a systemd, para que systemd haga de cgroups admin y se asegure que el único admin de este es él y está siendo usado. Para hacer esto hemos de reemplazar el contenido de daemon.json por el siguiente contenido, podemos hacerlo ejecutando este comando:

```
cat <<EOF | sudo tee /etc/docker/daemon.json
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
```

}
EOF

Ahora hemos de indicar en los ficheros del kernel que cgroups pueda limitar las características físicas de las cuales se quejaba antes. Podemos hacerlo con este comando:

sudo sed -i '$ s/$/ cgroup_enable=cpuset cgroup_enable=memory cgroup_memory=1 swapaccount=1/' /boot/cmdline.txt

Gracias a sed añade las líneas que indicamos al final del documento separando con un espacio.

Pero para desactivar la memoria swap del sistema es necesario ejecutar el comando *sudo dphys-swapfile swapoff* manualmente. Nuestra solución es añadir al final del fichero /etc/profile (un rc.local en Ubuntu). De esta manera se ejecutará automáticamente en cada arranque del sistema.

Ahora reiniciamos la máquina y aplicaremos todos los cambios que hemos hecho y al hacer docker info no veremos los warnings más. Y podemos proseguir con la instalación de Kubernetes, ya que ahora Kubeadm podrá pasar el check previo a la instalación correctamente.

3.2.2 Metal-LB

Este problema con el balanceador de cargas no lo conseguimos solucionar. Tiene dos partes, este apartado de Metal-LB y el siguiente de Traefik. En este punto del proyecto teníamos Kubernetes instalado en una máquina con una capa de red de nivel 2 con Weave Net y el siguiente paso era escoger un balanceador de cargas.

Primero lo intentamos con Metal-LB ya nos fue recomendado en proyectos anteriores al nuestro. Pero después de seguir los pasos de la documentación para instalarlo vía Addon a la máquina:

```
pi@k8smaster:~ $ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.
9.6/manifests/namespace.yaml
namespace/metallb-system created
pi@k8smaster:~ $ kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/v0.
9.6/manifests/metallb.yaml
Warning: policy/v1beta1 PodSecurityPolicy is deprecated in v1.21+, unavailable in v1.25
+
podsecuritypolicy.policy/controller created
podsecuritypolicy.policy/speaker created
serviceaccount/controller created
serviceaccount/speaker created
clusterrole.rbac.authorization.k8s.io/metallb-system:controller created
clusterrole.rbac.authorization.k8s.io/metallb-system:speaker created
role.rbac.authorization.k8s.io/config-watcher created
role.rbac.authorization.k8s.io/pod-lister created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:controller created
clusterrolebinding.rbac.authorization.k8s.io/metallb-system:speaker created
rolebinding.rbac.authorization.k8s.io/config-watcher created
rolebinding.rbac.authorization.k8s.io/pod-lister created
daemonset.apps/speaker created
deployment.apps/controller created
```

Vemos que los pods no arrancan:

```
pi@k8smaster:~ $ kubectl -n metallb-system get pods
NAME                          READY   STATUS                       RESTARTS   AGE
controller-64f86798cc-fw9c7   0/1     Pending                      0          9m51s
speaker-hgcl9                 0/1     CreateContainerConfigError   0          9m51s
```

Creamos la configuración del metallb, el fichero se ha de llamar config, si utilizamos helm para que no haya problemas el fichero se ha de llamar metallb-config y hemos de cambiar el “name” por el nombre del fichero del archivo.

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: default
      protocol: layer2
      addresses:
      - 192.168.1.235-192.168.1.25
```

Aplicamos la configuración con el comando kubectl apply -f config.yaml pero el pod controller sigue sin arrancar:

```
pi@k8smaster:~ $ kubectl -n metallb-system get pods
NAME                          READY   STATUS    RESTARTS   AGE
controller-64f86798cc-56kbg   0/1     Pending   0          22m
```

También probamos a instalarlo con helm pero tampoco hubo resultados.

Para que funcione tanto controller como speaker han de estar iniciados, ya que el controller es quien asigna las IP y el speaker es el demonio que hace que el controller pueda comunicarse y sea alcanzado. Pero con la documentación de una capa de nivel 2 debería de ser suficiente para que los pods arrancase, pero no es el caso. A día de hoy no sabemos exactamente que podría ser.

3.2.3 Traefik

Traefik fue nuestra segunda opción para usar como load balancer en nuestro cluster, fue una sugerencia de nuestro tutor Óscar. La instalación que hicimos con de Traefik fue con helm:

```
pi@k8smaster:~ $ helm install traefik traefik/traefik
```

Pero tampoco pudimos hacer que el único pod que se generaba de Traefik ni el servicio funcionase. En este punto del proyecto decidimos abandonar Kubernetes y buscar otras opciones. Como Minikube o Rancher, finalmente decidimos escoger

Rancher (k3s) y conseguimos crear el cluster de alta disponibilidad con una base de datos externa.

3.3 Problemas encontrados en k3s

3.3.1 Instalación

Una vez ya teníamos clara la arquitectura lógica del proyecto que instalar k3s. Al principio nos guiaremos en un tutorial que era un poco antiguo y a la hora de instalar el k3s le daba valor a las variables de la instalación en el mismo comando. Pero en la versión actual muchas variables han de ser exportadas anteriormente a la instalación.

La instalación correcta del servidor maestro es así. Primero declaramos la ruta de la base de datos:

```
usuario@master:~$ export K3S_DATASTORE_ENDPOINT='mysql://k3s:root@tcp(192.168.1.200:3306)/k3s'
```

Y luego la instalación, donde si podemos definir parámetros como taint o ssl-tan.

```
usuario@master:~$ curl -sfL https://get.k3s.io | sh -s - server --node-taint CriticalAddonsOnly=true:NoExecute --tls-san 192.168.1.210
```

Por otro lado el servidor trabajador también ha de ser instalado con el mismo método. Primero exportamos la variable del TOKEN del cluster y luego la variable de nuestro balanceador de cargas que hace de endpoint. Y finalmente lo podemos instalar:

```
usuario@worker1:~$ export K3S_TOKEN=K1083c4a21117e76910ec9e390532159e2916f03fdd718e8475b556565323659315::server:97cf485cd0a9d998a6aebf2015704616
usuario@worker1:~$ export K3S_URL=https://192.168.1.210:6443
usuario@worker1:~$ curl -sfL https://get.k3s.io | sh -
```

De esta forma podemos hacer la instalación del cluster donde hay servidores maestros y trabajadores.

3.3.2 Nginx

Como hemos visto anteriormente, en nuestra arquitectura lógica en frente de los dos servidores maestros tenemos un balanceador de cargas que distribuye el tráfico a cualquiera de los maestros, al principio siguiendo una guía decidimos utilizar Nginx. Pero la versión standard de nginx es simplemente HTTP. Para poder implementar también el protocolo TCP hemos de instalar una versión de pago llamada Nginx Plus, si buscamos precios por internet podemos encontrar lo siguiente:

NGINX Plus, Single Instance	NGINX WAF	Additional NGINX products
Starting at $2500 per year	**$2000** per year	**Varies** based on instances, servers. Contact us for custom pricing.
24/7 Enterprise support with 30-minute SLA is available, for $5,000/year per instance. We also provide Basic support for $2,500/year and Professional support for $3,500/year for less-critical deployments.	NGINX Web Application Firewall (WAF) is an additional cost option for NGINX Plus. It is available for $2,000 per instance, per year, in addition to your NGINX Plus subscription.	https://www.nginx.com/products/pricing/

De cara al presupuesto se nos salía un poco, ya que no queríamos pagar por ningún tipo de software. Un poco la filosofía de este proyecto es utilizar la mayor parte del proyecto sea software libre, la solución a esto fue buscar una alternativa que nos aconsejaron llamado HAProxy. Este cumplia las espectativas y objetivos, es decir con esta herramienta es más que suficiente para realizar nuestro proyecto y funcionalidades que necesitamos.

3.3.3 HAProxy

Para crear una buena configuración del balanceador de cargas tuvimos bastantes problemas. Ya que al principio pensábamos que solo teníamos que dirigir las peticiones tcp del puerto 6443 (puerto por que el se comunica rancher) a las máquinas servidores, pero si solo hacíamos esto no funcionaba el cluster.

Para solucionarlo hemos de redirigir también las peticiones http del puerto 80, 8080 y las tcp del puerto 443. Ya que el ingress controller Traefik dentro del cluster escucha en la máquina host por esos puertos y ha de poder comunicarse con ellos.

3.3.4 Nodos master caídos

A la hora de implementar la arquitectura final del cluster en máquinas virtuales, al componerse por tantas máquinas virtuales y disponer de poca RAM en mi máquina nativa dotamos a las máquinas virtuales con poca memoria. Y a la hora de arrancar el cluster los nodos maestros cada periodo corto de tiempo se reiniciaba y se perdía el servicio con ambos servidores maestros.

Teníamos la teoría de que esto pasaba por los componentes físicos como la RAM.

Al crear el cluster con las Raspberries Pi, que contienen cada una 4GB de RAM. Y poder dotar a las máquinas virtuales con más RAM al usar también un portátil donde podemos crear más máquinas.

Dejamos 6 horas el cluster encendido mientras monitorizamos el estado de los nodos del cluster y en ningún momento había caídas de los nodos. Por lo que podemos decir que nuestra teoría era correcta y los nodos maestros necesitaban unas especificaciones físicas algo más elevadas que 1GB de RAM virtual. Con 4GB de RAM de la Raspberry Pi dejó de haber problemas.

3.3.4 Ansible MySQL

En la última etapa del proyecto decidimos crear un playbook que hiciese automáticamente la instalación y configuración de nuestro cluster. Para solo deber de asignar una IP fija a las máquinas y dejar que el playbook se encargue de todo.

Pero a la hora de crear la base de datos y crear el usuario tuvimos problemas con el usuario 'root'@'localhost', ya que no conseguimos asignarle una contraseña para poder entrar localmente sin usar el sudo.

Por lo tanto, para no perder más tiempo decidimos abandonar la idea de la creación del playbook. Siendo este uno de los pocos errores que no conseguimos solucionar.

4 GLOSARIO

Cluster: se refiere al conjunto de dos o más máquinas que se comunican entre sí y comparten servicios y están constantemente monitorizadas.
Pod: es un grupo de uno o más contenedores de tipo docker que tienen almacenamiento, red y con unas especificaciones de cómo ejecutarlos.
Container: es una alternativa a las máquinas virtuales tradicionales y permiten la virtualización de la infraestructura y a diferencia de estas solo habilitan las aplicaciones de software y utilizan el sistema operativo de su host para proporcionar uno propio.
Addon: se refiere a extensiones, plugins, programas que funcionan anexados a otros y que sirven para complementar las funcionalidades de un programa.
Hosting: es un servicio de alojamiento para sitios web.
Dominio: es un nombre fácil para recordar una dirección IP asociada.
Nodo: son los encargados de ejecutar aplicaciones en pods.
Ingress Controller: es un balanceador de carga especializado para entornos Kubernetes (se configuran con la Api).
Api: es un conjunto de protocolos que se utilizan para el desarrollo e integración de software a aplicaciones, además de poder conectar productos y servicios con otros.
Kube: es una plataforma open source para administrar cargas de trabajo y servicios en aplicaciones con contenedores.

5 CONCLUSIÓN

Empezamos este proyecto que nació por la curiosidad tanto de Raspberry Pi como de las tecnologías Kubernetes (y Rancher) que Óscar mencionó en clase de sistemas cuando estábamos estudiando contenedores.

Es digno de mencionar que nuestro antiguo compañero del Puig Antonio Tomás Franco, también hizo un proyecto similar. Donde parte de nuestro proyecto fue basado en el suyo, con la premisa principal de aumentar los conocimientos y las funcionalidades que él había logrado.

Tan pronto empezamos a investigar nos dimos cuenta de la cantidad de puertas que abre al mundo de la orquestación de contenedores esta tecnología. Pero Kubernetes al igual que fenomenal, es también extremadamente compleja. Durante el primer mes del proyecto se puede resumir a “intentar poner las funcionalidades básicas de Kubernetes en marcha”. Durante este primer mes el proyecto más que ser práctico, era pura investigación.

En el segundo mes rompimos el status quo del proyecto y decidimos abandonar Kubernetes por Rancher. El objetivo seguía siendo el mismo, pero esta vez con una tecnología más sencilla y enfocada para un entorno de prueba. Hasta que finalmente logramos los objetivos principales.

Una vez acabado el proyecto estamos satisfechos con el esfuerzo que hemos realizado este tramo final de curso. Asumimos un camino difícil y contra todo pronóstico no nos desviamos de él hasta conseguir llegar a la meta.

Podemos afirmar que hemos aprendido mucho sobre las tecnologías que hemos trabajado, tanto Rancher como Kubernetes. A pesar que este último prácticamente lo vimos solo a nivel teórico.

6 WEBGRAFÍA

Instalación de k8s:
https://opensource.com/article/20/6/kubernetes-raspberry-pi
https://alexellisuk.medium.com/walk-through-install-kubernetes-to-your-raspberry-pi-in-15-minutes-84a8492dc95a
https://limpygnome.com/2019/09/21/raspberry-pi-kubernetes-cluster/

Configuración de kubernetes con docker:
https://raspberrypi.stackexchange.com/questions/80193/uninstall-docker
https://medium.com/@tukai.anirban/docker-on-raspberry-pi-getting-started-c7b403205ecf

Cambiar de dhcpcd a systemd-networkd:
https://raspberrypi.stackexchange.com/questions/78787/howto-migrate-from-networking-to-systemd-networkd-with-dynamic-failover
https://www.raspberrypi.org/forums/viewtopic.php?t=270213

Kubeadm:
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/
https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/high-availability/

Kubernetes addons:
https://kubernetes.io/docs/concepts/cluster-administration/addons/
Weaver net kubernetes:
https://www.weave.works/docs/net/latest/kubernetes/kube-addon/
https://www.weave.works/blog/weave-net-kubernetes-integration/
https://github.com/weaveworks/weave/issues/3758
https://kubernetes.io/docs/tasks/administer-cluster/network-policy-provider/weave-network-policy/
https://www.weave.works/docs/net/latest/kubernetes/kube-addon/

Metal-lb kubernetes:
https://blog.inkubate.io/install-and-configure-metallb-as-a-load-balancer-for-kubernetes/
https://metallb.universe.tf/configuration/
https://metallb.universe.tf/installation/
https://mvallim.github.io/kubernetes-under-the-hood/documentation/kube-metallb.html
https://medium.com/@JockDaRock/kubernetes-metal-lb-for-on-prem-baremetal-cluster-in-10-minutes-c2eaeb3fe813
https://metallb.universe.tf/

https://metallb.universe.tf/concepts/
https://ubuntu.com/kubernetes/docs/metallb
https://devopslearning.medium.com/metallb-load-balancer-for-bare-metal-kubernetes-43686aa0724f

Traefik kubernetes:
https://doc.traefik.io/traefik/v1.7/user-guide/kubernetes/
https://medium.com/kubernetes-tutorials/deploying-traefik-as-ingress-controller-for-your-kubernetes-cluster-b03a0672ae0c

K3s vs minikube vs kubernetes:
https://brennerm.github.io/posts/minikube-vs-kind-vs-k3s.html
https://www.reddit.com/r/kubernetes/comments/be0415/k3s_minikube_or_microk8s/
https://github.com/kubernetes/minikube/issues/94
https://stackshare.io/stackups/k3s-vs-minikube

K3s:
https://blog.codybunch.com/2020/10/13/Kubernetes-Fix-Node-Password-Rejected/
https://k3s.io/
https://rancher.com/docs/k3s/latest/en/installation/
https://blog.alexellis.io/test-drive-k3s-on-raspberry-pi/
https://rancher.com/docs/rancher/v2.x/en/installation/resources/k8s-tutorials/ha-with-external-db/
https://rancher.com/docs/k3s/latest/en/quick-start/

K3s alta disponibilidad:
https://www.youtube.com/watch?v=O3s3YoPesKs
https://rancher.com/docs/rancher/v2.x/en/installation/install-rancher-on-k8s/
https://www.youtube.com/watch?v=UoOcLXfa8EU&t=413s

HAProxy:
https://serversforhackers.com/c/using-ssl-certificates-with-haproxy
https://tecadmin.net/how-to-setup-haproxy-load-balancing-on-ubuntu-linuxmint/
https://itnext.io/create-a-highly-available-kubernetes-cluster-using-keepalived-and-haproxy-37769d0a65ba
https://github.com/k3s-io/k3s/issues/1409
https://github.com/rancher/k3d/issues/110

https://kubesphere.io/blogs/install-kubernetes-using-kubekey/
https://docs.nginx.com/nginx/admin-guide/load-balancer/http-load-balancer/
https://docs.nginx.com/nginx/admin-guide/load-balancer/
https://www.haproxy.com/blog/loadbalancing-faq/

https://www.haproxy.com/blog/layer-4-load-balancing-tunnel-mode/
https://www.haproxy.com/blog/layer-4-load-balancing-direct-server-return-mode/
https://www.haproxy.com/blog/layer-4-load-balancing-nat-mode/
https://github.com/citrix/citrix-k8s-ingress-controller/blob/master/docs/deployment-topologies.md

https://levelup.gitconnected.com/a-guide-to-k3s-ingress-using-traefik-with-nodeport-6eb29add0b4b

https://rancher.com/docs/rancher/v2.x/en/installation/resources/advanced/rke-add-on/layer-4-lb/
https://www.digitalocean.com/community/tutorials/an-introduction-to-haproxy-and-load-balancing-concepts

k3s dashboard:
https://github.com/kubernetes/dashboard
https://rancher.com/docs/k3s/latest/en/installation/kube-dashboard/
https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/

Prometheus y grafana:
https://www.civo.com/learn/monitoring-k3s-with-the-prometheus-operator-and-custom-email-alerts

DNS dinámico:
https://www.noip.com

CNI:
https://www.henrydu.com/2020/11/16/k3s-cni-flannel/

Nginx:
http://nginx.org/en/docs/http/load_balancing.html

Load Balancing:
https://www.nginx.com/resources/glossary/load-balancing/
https://docs.aws.amazon.com/es_es/elasticloadbalancing/latest/application/introduction.html
https://en.wikipedia.org/wiki/Load_balancing_(computing)

TSBD:
https://www.alibabacloud.com/product/hitsdb
https://en.wikipedia.org/wiki/Time_series_database
https://www.influxdata.com/time-series-database/

Ansible:
https://es.wikipedia.org/wiki/Ansible_(software)#Orquestaci%C3%B3n[10]%E2%80%8B

Nodos:
https://kubernetes.io/docs/tutorials/kubernetes-basics/explore/explore-intro/

Amazon:
https://aws.amazon.com/es/ec2/instance-types/t2/

7 ANEXOS

7.1 Proceso de creación del cluster

7.1.1 Configuración de MySQL

Hemos de configurar MySQL para que acepte conexiones a la base de datos desde el exterior. Para poder hacer eso hemos de editar el fichero /etc/mysql/mysql.conf.d/mysqld.cnf:

Ahora hemos de crear la base de datos que usará k3s para guardar los estados de los componentes del cluster. Es recomendable crearla con un collate latin1_swedish_ci, ya que k3s por defecto crea las bases de datos con este, así que para evitar problemas lo hacemos:
CREATE DATABASE k3s CHARACTER SET latin1 COLLATE latin1_swedish_ci;

Ahora hemos de crear un usuario que tenga permisos de superuser en la base de datos k3s, que es el que usará rancher para escribir en la base de datos:

CREATE USER 'k3s'@'%' IDENTIFIED BY 'root';

*GRANT ALL PRIVILEGES ON *.* TO 'k3s'@'%';*

Con el % indicamos que este usuario se puede conectar desde cualquier IP, y no solo desde localhost.

7.1.2 Configuración de HAProxy

Es importante que haproxy a parte de escuchar en el puerto 6443 (rancher), también escuche en los puertos 80 y 443 sino no funcionará. Ya que el ingress controller (traefik por defecto) necesita estos puertos.

La configuración de nuestro fichero es la siguiente:

```
defaults
        log     global
        mode    tcp
        option  tcplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
```

```
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http

frontend kube-server-6443
        bind *:6443
        mode tcp
        default_backend kube-server-6443

backend kube-server-6443
        mode tcp
        option tcp-check
        balance roundrobin
        server master1 192.168.1.220:6443 check
        server master2 192.168.1.221:6443 check

frontend kube-server-http
        bind *:80
        mode http
        default_backend kube-server-http

backend kube-server-http
        mode http
        balance roundrobin
        server master1 192.168.1.220:80 check
        server master2 192.168.1.221:80 check

frontend kube-server-http8080
        bind *:8080
        mode http
        default_backend kube-server-http8080

backend kube-server-http8080
        mode http
        balance roundrobin
        server master1 192.168.1.220:8080 check
        server master2 192.168.1.221:8080 check


frontend kube-server-https
        bind *:443
        mode tcp
        default_backend kube-server-https

backend kube-server-https
        mode tcp
        balance roundrobin
        option ssl-hello-chk
```

server master1 192.168.1.220:443 check
server master2 192.168.1.221:443 check

La sintaxis de haproxy como vemos es simple. Se divide en un apartado de "frontend" donde indicamos el puerto que haproxy va a escuchar y si este puerto será HTTP o TCP. Luego una parte llamada backend que está ligada a la de frontend gracias al parámetro default donde indicamos el nombre del backend. Aquí indicamos el algoritmo de balanceo de carga, roundrobin. Y después los servidores disponibles junto al puerto donde puedan ir dirigidas las peticiones del frontend.

7.1.3 Masters

En los servidores masters solo hemos de indicar la ruta a la base de datos. Mysql en este caso. Después indicamos el login (user-password), por motivos de seguridad has de ingresar la contraseña hasheada y no como en este ejemplo. Y después de la @ el método de la conexión y la IP de la máquina donde está la base de datos, el puerto y por último el nombre de la base de datos.

export K3S_DATASTORE_ENDPOINT='mysql://k3s:root@tcp(192.168.18.135:3306)/k3s'

Luego instalamos k3s indicando que vamos a hacer un taint, es decir alejar a todos los pods de que se ejecuten en los servidores máster salvo los pods necesarios para el funcionamiento de estos y por último la IP de nuestro balanceador de cargas.

curl -sfL https://get.k3s.io | sh -s - server --node-taint CriticalAddonsOnly=true:NoExecute --tls-san 192.168.18.210

Hemos de ingresar el parámetro K3S_DATASTORE_ENDPOINT como variable porque sino nos encontramos con un fallo de sintaxis.

Miramos los nodos una vez hemos ingresado ambos servidores:

```
usuario@master:~$ sudo k3s kubectl get node
NAME      STATUS   ROLES                  AGE     VERSION
master2   Ready    control-plane,master   17m     v1.20.6+k3s1
master    Ready    control-plane,master   9m27s   v1.20.6+k3s1
```

Comprobamos que pods se están ejecutando sin todavía ingresar ningún servidor esclavo y con el taint.

```
usuario@master2:~$ sudo k3s kubectl get pod --all-namespaces
NAMESPACE     NAME                                     READY   STATUS    RESTARTS   AGE
kube-system   metrics-server-86cbb8457f-kf6pg          1/1     Running   6          17m
kube-system   coredns-854c77959c-c55ld                 1/1     Running   0          17m
kube-system   local-path-provisioner-5ff76fc89d-vqhfx  1/1     Running   7          17m
kube-system   helm-install-traefik-b4gl9               0/1     Pending   0          17m
```

Traefik todavía no se ha podido instalar porque necesitamos añadir algún worker al cluster.

Es importante que cada máquina tenga un hostname distinto, sino tendremos problemas con los nombres de los nodos a la hora de la instalación si no añadimos un parámetro especial, para que los nodos puedan llamarse igual. Nosotros simplemente los cambiamos a master y master2.

7.1.4 Workers

Para ingresar el worker al cluster hemos de conseguir el token en cualquier máquina maestro:

```
usuario@master:~$ sudo cat /var/lib/rancher/k3s/server/node-token
K1083c4a21117e76910ec9e390532159e2916f03fdd718e8475b556565323659315::server:97cf485cd0a9d998a6aebf2015704616
```

En la instalación hemos de ingresar los parámetros como en los masters como variables para evitar errores de sintaxis. En la primera ingresamos el valor del token de los nodos y en K3S_URL indicamos la dirección de nuestro balanceador de cargas y el puerto 6443 que es el que usa rancher para comunicarse como hemos indicado antes:

```
usuario@worker1:~$ export K3S_TOKEN=K1083c4a21117e76910ec9e390532159e2916f03fdd718e8475b556565323659315::server:97cf485c
d0a9d998a6aebf2015704616
usuario@worker1:~$ export K3S_URL=https://192.168.1.210:6443
usuario@worker1:~$ curl -sfL https://get.k3s.io | sh -
```

Esperamos que arranque k3s-agent en el worker y miramos los nodos en uno de los maestros:

```
usuario@master:~$ sudo k3s kubectl get node
NAME      STATUS   ROLES                  AGE     VERSION
master2   Ready    control-plane,master   35m     v1.20.6+k3s1
master    Ready    control-plane,master   28m     v1.20.6+k3s1
worker1   Ready    <none>                 4m54s   v1.20.6+k3s1
```

Pero al añadir el worker vemos que se han generado más pods:

```
usuario@master:~$ sudo k3s kubectl get pod --all-namespaces --output=wide
NAMESPACE     NAME                                      READY   STATUS      RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS GATES
kube-system   metrics-server-86cbb8457f-kf6pg           1/1     Running     6          40m     10.42.0.4   master2   <none>           <none>
kube-system   coredns-854c77959c-c551d                  1/1     Running     0          40m     10.42.0.3   master2   <none>           <none>
kube-system   local-path-provisioner-5ff76fc89d-vqhfx   1/1     Running     7          40m     10.42.0.2   master2   <none>           <none>
kube-system   helm-install-traefik-b4gl9                0/1     Completed   0          40m     10.42.2.2   worker1   <none>           <none>
kube-system   svclb-traefik-s25px                       2/2     Running     0          5m31s   10.42.0.5   master2   <none>           <none>
kube-system   svclb-traefik-x4vhm                       2/2     Running     0          5m31s   10.42.2.3   worker1   <none>           <none>
kube-system   svclb-traefik-p44vq                       2/2     Running     0          5m31s   10.42.1.2   master    <none>           <none>
kube-system   traefik-6f9cbd9bd4-c5m86                  1/1     Running     0          5m32s   10.42.1.3   master    <none>           <none>
```

Y traefik se ha instalado finalmente y ya está siendo ejecutado.

7.1.5 Kubeconfig y kubernetes dashboard

Kubernetes ofrece la posibilidad de poder administrar el cluster desde una máquina remota. Pero para eso hemos de instalar kubectl y el dashboard. Para ello seguiremos los pasos de la documentación oficial de rancher.

Hemos de crear una nueva máquina, importante que sea **desktop**, donde tenemos que tener instalado kubectl.

Instalación kubectl:

```
usuario@kubedashboard:~$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/s
table.txt)/bin/linux/amd64/kubectl"
```

```
usuario@kubedashboard:~$ curl -LO "https://dl.k8s.io/$(curl -L -s https://dl.k8s.io/release/stable.tx
t)/bin/linux/amd64/kubectl.sha256"
```

Comprobamos:

```
usuario@kubedashboard:~$ echo "$(<kubectl.sha256) kubectl" | sha256sum --check
```

```
usuario@kubedashboard:~$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Una vez instalado hemos de copiar la configuración de algún master y copiarla en este directorio de la nueva máquina. Vamos al master:

```
usuario@master1:~$ sudo cat /etc/rancher/k3s/k3s.yaml
```

Y lo copiamos en:

```
usuario@kubedashboard:~/.kube$ nano config
```

Y ahora hemos de indicar en server la IP de nuestro lb haproxy:

```
 server: https://192.168.18.210:6443
```

Y ahora desde dashboard podemos ver los nodos con kubectl:

```
usuario@kubedashboard:~$ kubectl get nodes
NAME      STATUS   ROLES                  AGE   VERSION
worker2   Ready    <none>                 20m   v1.20.6+k3s1
worker1   Ready    <none>                 20m   v1.20.6+k3s1
master2   Ready    control-plane,master   31m   v1.20.6+k3s1
master1   Ready    control-plane,master   32m   v1.20.6+k3s1
```

Instalamos el dashboard con este comando:
kubectl apply -f
https://raw.githubusercontent.com/kubernetes/dashboard/v2.2.0/aio/deploy/recommended.yaml

```
usuario@kubedashboard:~$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/dashboard/v2.2
.0/aio/deploy/recommended.yaml
```

Ahora creamos el user admin:

```
usuario@kubedashboard:~$ nano dashboard.admin-user.yml
```

Con el siguiente contenido:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
```

Ahora creamos el rol, en un fichero llamado dashboard.admin-user-role.yml:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
```

- kind: ServiceAccount
* name: admin-user*
* namespace: kubernetes-dashboard*

Lanzamos la configuración:
kubectl create -f dashboard.admin-user.yml -f dashboard.admin-user-role.yml

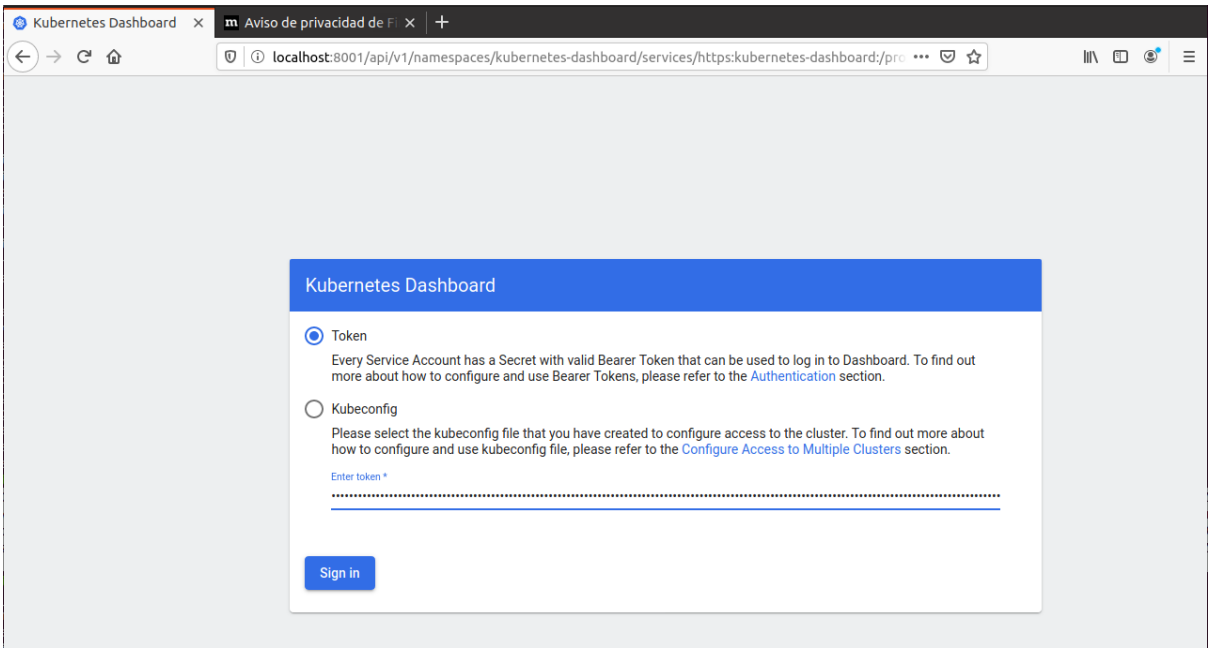
Conseguimos el token para poder acceder al dashboard de nuestro cluster:
kubectl -n kubernetes-dashboard describe secret admin-user-token | grep '^token'

```
usuario@kubedashboard:~$ kubectl -n kubernetes-dashboard describe secret admin-user-token | grep '^to
ken'
token:      eyJhbGciOiJSUzI1NiIsImtpZCI6Ijl6YThjeWxjbjBxS3F2T3ZOU3NYbWhmblJFaFBaQjF5Mnp0VXBXOVBaQ00if
Q.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2U
iOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11
c2VyLXRva2VuLW54bjJkIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNlLWFjY291bnQubmFtZSI6ImFkbWluL
XVzZXIiLCJrdWJlcm5ldGVzLmlvL3NlcnZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiJhMzFiNWRiYS03ZTJkLTRlOD
AtOWU3Ny1mMmE2ODNiODJlMzkiLCJzdWIiOiJzeXN0ZW06c2VydmljZWFjY291bnQ6a3ViZXJuZXRlcy1kYXNoYm9hcmQ6YWRtaW4
tdXNlciJ9.ZhMnwVlr7AjP79mCH2U3FPvZ-2JDuIXtRUMH4l-h_zejN3WU2N2IEX5wx3vuQLk9sZJcVfEZOirooNj5FDRuJv7rzx8
-BVl0FQzCW_WohKCLXoeYZiS7Sdme_lF5k_wewnzT9cjb3Ik_zGONam-ixNP_ZsCWGZlU9Fe4GVlwj-KzogkjyBBuwu2XPx4ashlI
vfxsaF8ML-BSbzio3tpYe6palARjKOcIQEXUauRvBKe9LJscQ5tM5FcDQaVJLE8wZ8s-kcVCydt2Hp-InTa7iRM4qhK-r7-HEOw3A
jxxz0sKmdX25N6T1yqo1-c7G0yi4qV-9845gF5FdZQUBJwJGQ
```

Abrimos un tunel hacia el cluster:

```
usuario@kubedashboard:~$ kubectl proxy
```

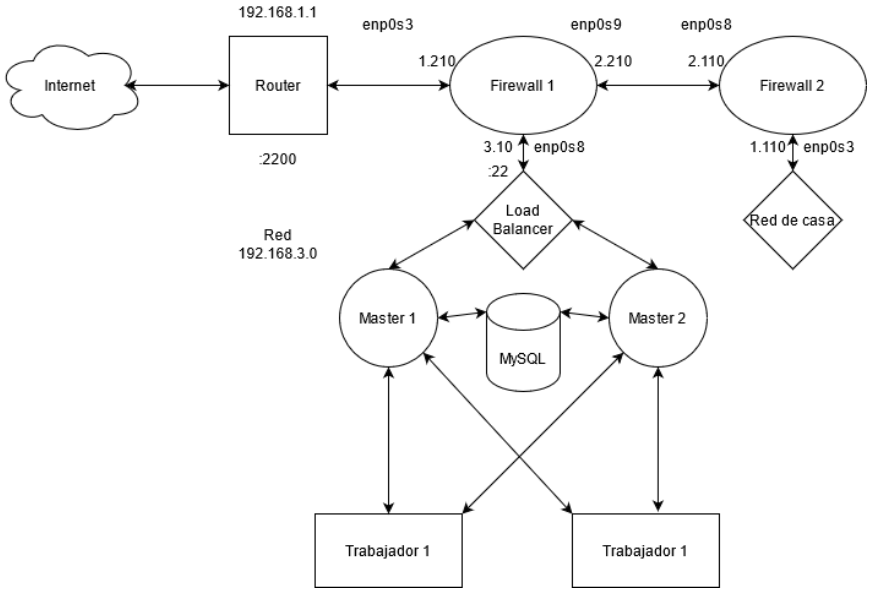
Ahora accedemos a localhost:8001 y copiamos el token y lo ponemos como password:



Link para acceder al dashboard: http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

7.2 DMZ

Para la configuración de las reglas del firewall hemos escogido nftables.
Tomando como referencia este esquema de red:



Reglas del firewall 1:

Creamos la tabla de nat:
sudo nft add table ip nat
sudo nft add chain ip nat prerouting { type nat hook prerouting priority 100\; }
sudo nft add chain ip nat postrouting { type nat hook postrouting priority 100\; }

La regla para que las peticiones ssh al puerto 2200 que entren por la tarjeta enp0s3 sean redirigidas al balanceador de cargas al puerto 22:
sudo nft add rule ip nat prerouting iifname enp0s3 tcp dport 2200 dnat 192.168.3.210:22

Las reglas para que las red de los servidores y la red de casa tengan acceso a internet. Indicando al firewall que haga masquerade de las conexiones que salgan por la tarjeta enp0s3 provenientes de la red 192.168.1.0 o 192.168.3.0. De esta manera cuando la petición salga tendrá la IP del destino del servidor y cuando reciba la respuesta el firewall será capaz de saber a que IP de las redes internas pertenecía esta respuesta.

La regla para impedir que de la red interna donde están los servidores se pueda acceder a la red de casa:
sudo nft add rule inet filter forward iifname enp0s8 ip daddr 192.168.1.0 drop

Reglas del firewall 2:

Creamos la tabla:

sudo nft add table inet filter
sudo nft add chain inet filter input { type filter hook input priority 0 \; policy accept \; }

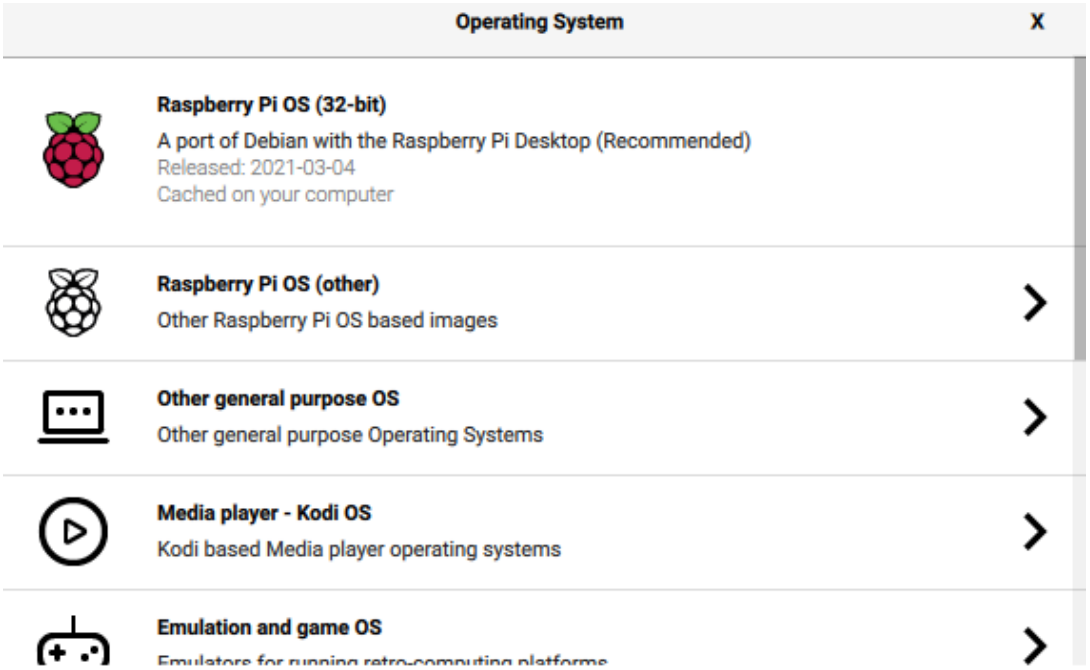
sudo nft add chain inet filter output { type filter hook output priority 0 \; policy accept
sudo nft add chain inet filter forward { type filter hook forward priority 0 \; policy accept

Regla para que desde la red de casa no se pueda acceder a la red de servidores:
sudo nft add rule inet filter forward iifname enp0s3 oifname enp0s8 ip daddr 192.168.3.0 drop

7.3 Raspberry pi para que haga de Master

Para la instalación de Raspbian en la SD utilizamos Raspberry Pi Imager. Contiene varios tipos de SO a parte de Raspbian, como Ubuntu y otros sistemas linux.

Aquí podemos ver la lista de los SO que te deja seleccionar:



Escogimos Raspberry Pi OS.

Lo primero una vez iniciamos la raspberry es cambiarle la IP e instalar el servicio ssh para poder acceder desde la máquina nativa y no tener que usar un teclado y ratón adicional.

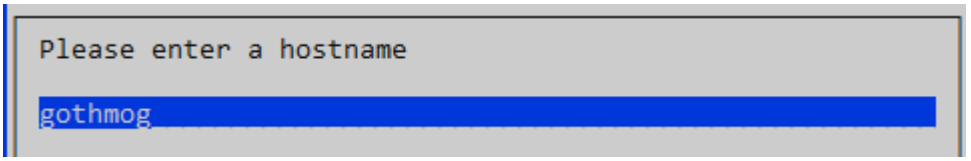
Importante al instalar el paquete ssh hacer un enable del servicio.

Configuración de red:
Editar el fichero /etc/dhcpcd.conf.

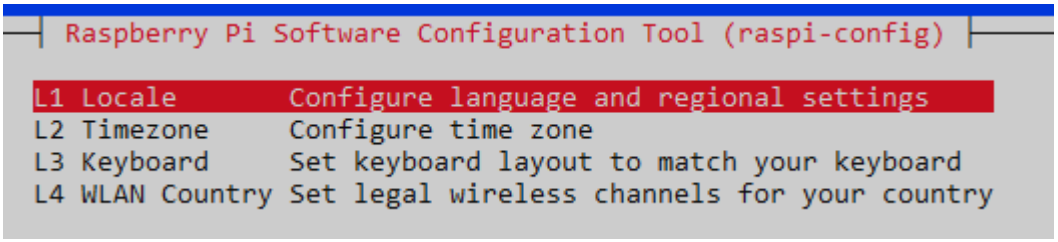
```
# Example static IP configuration:
interface eth0
static ip_address=192.168.1.220/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.1.1
static domain_name_servers=192.168.1.1 8.8.8.8
```

Para aplicar esta configuración de red hemos de reiniciar el sistema, no basta con reiniciar el servicio dhcpcd ya que no cambiará la IP.

Una vez dentro podemos proseguir con la configuración.
Configuración de hostname, zona horaria, lenguaje y teclado:
Ejecutamos *sudo raspi-config*. Seleccionamos System Options y a posteriori Hostname.



Para modificar la zona horario, lenguaje y teclado seleccionamos Localisation Options:



Y escogemos como zona horaria Madrid, lenguaje español o inglés y teclado español.
Para aplicar los cambios reiniciamos el sistema.

Por último hemos de añadir los parámetros de cgroup en el fichero del kernel
sudo sed -i '$ s/$/ cgroup_enable=cpuset cgroup_enable=memory cgroup_memory=1 swapaccount=1/' /boot/cmdline.txt

Y reiniciar la máquina para que se apliquen los cambios de la configuración y los parámetros de cgroup carguen en el kernel. Sinó durante la instalación de rancher nos saltará un error relacionado a los parámetros de cgroup y no podrá arrancar.

Ahora repetimos el mismo proceso con el master 2.

Una vez tenemos todas las máquinas preparadas podemos instalar k3s en las raspberries y después añadir a los trabajadores. Por último añadiremos a la máquina administradora donde podremos ver el dashboard.

7.4 Crear un deployment

Creamos un fichero con el siguiente contenido:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 10
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

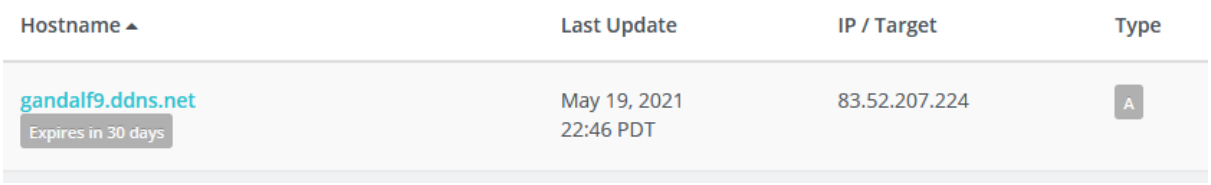
Lo ejecutamos y crearemos 10 pods con nginx escuchando en el puerto 80:
kubectl apply -f prueba-deployment.yaml

7.5 Acceder a la red del cluster desde el exterior

El objetivo es poder acceder al laboratorio donde tenemos las raspberry funcionando desde fuera de la red,es decir sin necesitar estar en casa. Para eso primero hemos de abrir el puerto 220 y linkearlo a la máquina 192.168.1.210 por ejemplo, que es la máquina que hace de balanceador de cargas.

Ahora hemos de registrarnos en noip:
https://www.noip.com

El dominio tiene que apuntar a la IP pública de nuestro router, ya que nuestro router tiene el puerto 2200 abierto donde está la IP 192.168.1.210 escuchando. Entonces cualquier petición dirigida al puerto 2200 a gandalf7.ddns.net irá a 192.168.1.210.
Por ende ssh -p 2200 usuario@gandalf7.ddns.net funciona.



Solo hemos de crear un hostname y no hemos de configurarlo, se asocia solo a la IP pública de nuestro router.

Para cambiar el puerto ssh hemos de editar el fichero /etc/ssh/sshd_config:

```
Port 2200
```

Indicamos el puerto por el que ssh estará escuchando.

Y ahora reiniciamos los servicios:

```
sudo service ssh restart_
```

```
sudo service sshd restart
```

Y comprobamos que están escuchando en el 2200:

```
usuario@firewall:~$ ss -tulpn | grep 2200
tcp   LISTEN 0      128          0.0.0.0:2200       0.0.0.0:*
tcp   LISTEN 0      128             [::]:2200          [::]:*
```

7.6 Acceder al dashboard desde el exterior:

Hemos de hacer un puente del puerto 8001 a nuestra máquina localhost, y este puerto 8001 recibe la información de la máquina donde se encuentra instalado el dashboard. Pero antes de nada hemos de abrir un puerto del router asociado a la máquina donde está instalado el dashboard, en este caso hemos abierto el 2201, pero puede ser cualquier otro. Lo único que esta máquina ha de escuchar en el puerto 2201 también:

Nos conectamos vía ssh:
ssh -p 2201 -L 8001:127.0.0.1:8001 usuario@gandalf9.ddns.net

Obtenemos el token en la misma conexión:
kubectl -n kubernetes-dashboard describe secret admin-user-token | grep '^token'

Y en esta sesión ejecutar kubectl proxy. Ahora podemos acceder desde localhost al dashboard:
http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/

Al entrar nos pedirá el token que hemos obtenido con el comando anterior.