Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TRABAJO FINAL DE GRADO

TÍTULO DEL TFG: Despliegue de monitorización de servicios a través de Kubernetes

TITULACIÓN: Grado en Ingeniería Telemática

AUTOR: Jonatan Romero Martínez

DIRECTOR: Roque Meseguer Pallares

SUPERVISOR: Esteban Méndez Fernández

FECHA: 19 de octubre del 2022

Títol: Desplegament de monitorització de serveis a través de Kubernetes

Autor: Jonatan Romero Martínez

Director: Roque Meseguer Pallares

Data: 19 de octubre del 2022

Resum

A causa del gran creixement de les empreses en entorns digitals i del gran increment de l'ús de microserveis a les seves infraestructures (tendència de monòlit a microserveis), ha nascut la necessitat de la creació de noves tecnologies. Aquestes tecnologies, anomenades eines d'orquestració, són capaces de proporcionar: Alta disponibilitat, escalabilitat, recuperació de dades a través de còpies de seguretat i la completa gestió de centenars de contenidors alhora. Pel que avui dia, es consideren imprescindibles en el desenvolupament de grans infraestructures.

Al llarg dels capítols del projecte que lliuro, tractarem principalment sobre aquest tipus de tecnologies, on es veurà: El disseny i el desplegament d'infraestructura fent ús d'eines d'orquestració, validació i desplegament d'un sistema de monitorització, proves d'avaluació dels diferents desplegaments realitzats i comprovació del bon funcionament dels mateixos.

Aquest projecte ens permetrà veure el gran potencial a esprémer que tenen aquestes eines d'orquestració i la importància que ha adquirit a les empreses per al desplegament ràpid de les infraestructures.

Un altre punt del que ens beneficiem d'aquest tipus de tecnologies és la reducció de les despeses energètiques i la millora de la sostenibilitat, principalment en els desplegaments al núvol. Per tant, l'impacte ambiental que puguem ocasionar es veu reduït gràcies a aquestes eines d'orquestració. Aquesta millora en l'impacte ambiental es deu en gran part a la bona planificació dels desplegaments que es realitzen dins dels sistemes que utilitzen eines d'orquestració. Maximitzant així l'ús dels recursos que disposem als nostres servidors. Per tant, evitem l'ús ineficient dels recursos i en conseqüència evitem la necessitat d'incorporar nous servidors.

Títol: Despliegue de monitorización de servicios a través de Kubernetes

Autor: Jonatan Romero Martínez

Director: Roque Meseguer Pallares

Fecha: 19 de octubre de 2022

Resumen

Debido al gran crecimiento de las empresas en entornos digitales y al gran incremento del uso de microservicios en sus infraestructuras (tendencia de monolito a microservicios), ha nacido la necesidad de la creación de nuevas tecnologías. Estas tecnologías, llamadas herramientas de orquestación, son capaces de proporcionar: Alta disponibilidad, escalabilidad, recuperación de datos a través de backups y la completa gestión de cientos de contenedores a la vez. Por lo que hoy en día, se consideran imprescindibles en el desarrollo de grandes infraestructuras.

A lo largo de los capítulos del proyecto que entrego, trataremos principalmente sobre este tipo de tecnologías, donde se verá: El diseño y despliegue de infraestructura haciendo uso de herramientas de orquestación, validación y despliegue de un sistema de monitorización, pruebas de evaluación de los diferentes despliegues realizados y comprobación del buen funcionamiento de los mismos.

Este proyecto nos permitirá ver el gran potencial a exprimir que tienen estas herramientas de orquestación y la importancia que ha adquirido en las empresas para el rápido despliegue de sus infraestructuras.

Otro punto del que nos beneficiamos de este tipo de tecnologías es en la reducción de los gastos energéticos y en la mejora de la sostenibilidad, principalmente en los despliegues en la nube. Por lo que el impacto ambiental que podamos ocasionar, se ve reducido gracias a estas herramientas de orquestación. Esta mejora en el impacto ambiental, es debido en gran parte a la buena planificación de los despliegues que se realizan dentro de los sistemas que utilizan herramientas de orquestación. Maximizando de esta manera el uso de los recursos que disponemos en nuestros servidores. Por lo que evitamos el uso ineficiente de los recursos y en consecuencia evitamos la necesidad de tener que incorporar nuevos servidores.

Title: Deployment of service monitoring through Kubernetes

Author: Jonatan Romero Martínez

Director: Roque Meseguer Pallares

Date: Oct 19 th 2022

Abstract

Due to the great growth of companies in digital environments and the great increase in the use of microservices in their infrastructures (trend from monolith to microservices), the need to create new technologies has arisen. These technologies, called orchestration tools, are capable of providing: High availability, scalability, data recovery through backups and the complete management of hundreds of containers at the same time. Therefore, today, they are considered essential in the development of large infrastructures.

Throughout the chapters of the project that I deliver, we will deal mainly with this type of technology, where you will see: The design and deployment of infrastructure using orchestration tools, validation and deployment of a monitoring system, evaluation tests of the different deployments carried out and verification of their proper functioning.

This project will allow us to see the great potential that these orchestration tools have and the importance that they have acquired in companies for the rapid deployment of their infrastructures.

Another point from which we benefit from this type of technology is the reduction of energy costs and the improvement of sustainability, mainly in cloud deployments. So the environmental impact that we can cause is reduced thanks to these orchestration tools. This improvement in the environmental impact is largely due to the good planning of the deployments that are carried out within the systems that use orchestration tools. Maximizing in this way the use of the resources that we have on our servers. Therefore, we avoid the inefficient use of resources and consequently avoid the need to incorporate new servers.

Dedicatoria

Quiero dedicarle este trabajo principalmente a mi pareja Desirée, ya que me ha apoyado en todo momento. Ella me ha ayudado y motivado desde un principio, también me ha aportado ideas, libros y conocimiento. Sin ella no habría disfrutado de la misma manera todos los pasos, todo el recorrido, que al final me han llevado a cumplir el propósito, mi meta, la etapa más importante de una carrera universitaria. Aprovecho esta oportunidad para dar gracias a la vida por tenerla a mi lado. Siempre será mi gran apoyo incondicional.

Agradecer también a ToBeIT por darme la oportunidad de poder formar parte del crecimiento de la empresa. Este proyecto, no únicamente ha sido un estudio continuo de diferentes tecnologías, sino que ha servido como base y parte de un proyecto real que ha hecho sentirme mucho más gratificado y realizado conmigo mismo.

Me gustaría además mencionar a Roc Meseguer Pallarès, ya que desde un principio se ha esmerado en ayudarme en todo lo que ha estado en su mano. Me ha mantenido motivado en los momentos más difíciles para mi. Estoy muy agradecido por haberlo tenido como director de este proyecto.

Por último, quiero dedicar este trabajo especialmente a mi familia. Ellos son el motivo por el cual hoy estoy escribiendo esta dedicatoria, y también el motivo de que hoy logre estar cerrando esta etapa. Quiero mostrar especial mención a mi “yaya”, siempre daré lo mejor de mi porque eso me enseñó. En su memoria, y gracias.

ÍNDICE

1. Introducción……..1
1.1. Motivación del proyecto……...1
1.2. Contexto……1
1.3. Identificación del problema…… 2
1.4. Objetivos…… 2
1.4.1. Generales…… 2
1.4.2. Específicos……...3

2. Conceptos y Tecnologías……..4
2.1. Conceptos tecnológicos…….. 4
2.1.1. Infraestructura hardware…… 4
2.1.2. Infraestructura contenedores y virtualización…… 4
2.1.3. Servicios……5
2.2. Estudio y validación de las tecnologías de monitorización…… 5
2.2.1. Estudio de algunas de las tecnologías de monitorización actuales 5
2.2.2. Validación de las tecnologías de monitorización……6
2.3. Tecnología utilizada como herramienta de orquestación……...7
2.3.1 Componentes principales que conforman Kubernetes…….. 7
2.3.2 Tipos de Nodos de Kubernetes……10
2.4. Recursos del cloud de Ionos disponibles para el proyecto…… 12

3. Metodología……14
3.1 Metodología DevOps…….. 14
3.2. Herramientas……...15
3.2.1. Herramientas de gestión……..15
3.2.2. Herramientas de seguimiento…… 16
3.3. Definición del plan inicial……...16

4. Diseño…….. 20
4.1. Requisitos y especificaciones…….. 20
4.2. Diseño de la infraestructura de Kubernetes……...22
4.2.1. Añadiendo un nuevo servidor Master/Worker Node……... 23

5. Despliegue de la infraestructura de Kubernetes…….. 25
5.1. Mis decisiones de mi TFG……25
5.2. Creación de la infraestructura…….. 25
5.2.1. Prerrequisitos……25
5.2.2. Configuración de la infraestructura…… 26
5.2.2.1. Inicialización de la infraestructura…… 26
5.2.2.2. Container Networking Interface e incorporación de nodos...27
5.2.2.3. Ingress Controller……29

5.3. Configuración de los Storage Class…… 32
5.3.1. Storage class escogido para Elastic…….. 33
5.3.2. Storage class escogido para Grafana……35

6. Despliegue de Elastic y Grafana……38

7. Ficheros de configuración de las diferentes tecnologías de monitorización…… 43
7.1. Ficheros de configuración de Elastic, Kibana y Grafana…… 43
7.1.1. Ficheros de configuración de Elastic…… 43
7.1.2. Fichero de configuración de Kibana…….. 45
7.1.3. Fichero de configuración de Grafana……47

8. Pruebas de evaluación……51
8.1 Verificación de los despliegues mediante Metricbeat…… 51
8.2 Validación de servicios mediante CLI de Kubernetes……54
8.3 Ficheros de configuración de Metricbeat……54

9. Conclusiones……..56
9.1. Conclusiones técnicas……...56
9.1.1. Incidencias…… 57
9.2. Conclusiones personales……..57
9.3. Futuras mejoras……..58

1. Introducción

En este capítulo se mostrará el impulso e iniciativa que ha hecho nacer este proyecto. De la misma manera se explicará la necesidad de incorporar tecnologías de orquestación, que son tendencia y que están ocupando las bases de los despliegues de infraestructura moderna.

1.1. Motivación del proyecto

Mi proyecto (*Despliegue de monitorización de servicios a través de Kubernetes)* se trata de un trabajo de final de grado realizado en la Universidad Politécnica de Cataluña (UPC), en la Escuela de Ingeniería de Telecomunicación y Aeroespacial de Castelldefels (EETAC), concretamente por el grado de Ingeniería en telecomunicaciones especializado en Telemática. Está elaborado como proyecto realizado en empresa, junto a la empresa ToBeIT, una consultoría informática.

El objetivo de este proyecto es diseñar y desplegar una infraestructura utilizando herramientas de orquestación y a su vez crear un sistema de monitorización completo. Para ello se utilizarán diferentes herramientas de monitorización, visualización y orquestación.

1.2. Contexto

Muchas empresas desarrollan sus servicios y aplicaciones a través de máquinas virtuales, por lo que se gastan muchos recursos innecesarios para cuplir estas necesidades. Esto incluso sería más grave si la empresa notase un gran crecimiento de demanda de servicios. En este caso, los gastos de infraestructura serían mayores. Ya que se realizaría el despliegue de nuevas máquinas virtuales dentro de los nuevos servidores desperdiciando aún más recursos y siendo mucho menos eficientes.

Al fin de cuentas las empresas que desarrollan este tipo de infraestructuras, se ven mucho más afectadas económicamente que otras empresas que utilizan mecanismos más innovadores como son los sistemas que incorporan el uso de herramientas de orquestación.

En los últimos dos años, debido a la llegada del covid-19, lo que no parecía de gran importancia para muchas empresas empezó a serlo. Las empresas se vieron con la necesidad de crecer de manera horizontal debido al gran aumento de demanda y de desplegar sus infraestructuras de manera distribuida. Haciendo uso de microservicios y herramientas de orquestación.

Además, también surge el requerimiento de un mayor control de sus infraestructuras a través de la monitorización, igual que del uso de sus aplicaciones.

Otra problemática hallada aquí, es que los despliegues de mecanismos de monitorización de aplicaciones y servidores, son procesos que requieren de su tiempo. Por eso nace la necesidad de migrar todo a arquitecturas que impliquen herramientas de orquestación. Lo cual facilita mucho más los despliegues, además de que nos aportan más robustez y nos permiten una escalabilidad más granular del desarrollo de nuestra infraestructura.

1.3. Identificación del problema

Debido al gran crecimiento en las infraestructuras de las empresas, nace la necesidad de incorporar mecanismos que permitan realizar despliegues de infraestructura de manera más rápida eficaz y sencilla.

Los mecanismos que hoy en día se siguen utilizando, se están quedando obsoletos. Haciendo que las empresas no utilicen sus recursos de manera eficiente y que los despliegues, además de ser complicados, consuman mucho tiempo.

Es por eso que se debe hacer uso de tecnologías emergentes para la creación de una arquitectura totalmente eficiente respecto a lo que se ha estado utilizando estos años.

Otro problema identificado son la cantidad de herramientas existentes que ofrecen la posibilidad de monitorizar infraestructuras y el rendimiento de las aplicaciones. Ya que la mayoría de las herramientas existentes, no ofrecen una solución completa. Por lo que es necesario realizar una buena selección de herramientas de monitoreo según las necesidades de cada empresa.

1.4. Objetivos

1.4.1. Generales

Los objetivos en los que consiste este proyecto son: diseñar y desplegar una infraestructura utilizando herramientas de orquestación y desplegar un sistema de monitorización completo. Para ello se utilizarán diferentes herramientas de monitorización, visualización y orquestación.

1.4.2. Específicos

Los objetivos específicos de este proyecto son:

- Validación de las diferentes tecnologías de monitorización escogidas por TobeIT.

- Diseño de una infraestructura con Kubernetes de contenedores por un clúster proporcionado por la empresa en Ionos.

- Desarrollo de las infraestructuras de Kubernetes y de monitorización a partir de la lista de tecnologías que me ha proporcionado la empresa.

- Creación y desempeño de los diferentes ficheros de configuración de las tecnologías de monitorización escogidas por la empresa.

- Realización de pruebas a través de servicios para la obtención de métricas del sistema operativo para posteriormente poder analizar el rendimiento de los servidores.

Una vez vistos los objetivos se da por concluido este capítulo. Ahora se dará paso al capítulo de conceptos y tecnologías.

2. Conceptos y Tecnologías

Vista la introducción, hablaremos de los conceptos tecnológicos, de las diferentes tecnologías de monitorización más competentes en el mercado, de la validación de las tecnologías de monitorización escogidas por la empresa, de la arquitectura utilizada para el despliegue de una infraestructura mediante el uso de herramientas de orquestación, y de los recursos disponibles del cloud para la realización de este despliegue.

2.1. Conceptos tecnológicos

Para poder tener una mejor lectura y comprensión del proyecto, es necesario tener una noción básica de los términos más utilizados y que se pueden ir encontrando a lo largo de la lectura del documento. Estos términos son explicados a continuación.

2.1.1. Infraestructura hardware

- **Nube (también conocida como Cloud):** Término utilizado que hace referencia a servidores remotos que están conectados a internet y cuya ubicación se encuentra en los Data Centers. Estos servidores pueden proporcionar gran variedad de servicios; ya sea almacenamiento y procesado de datos, compra de servidores con personalización total de recursos, bases de datos, redes y software.

  Ejemplos de empresas que proporcionan estos servicios son: AWS de Amazon, IBM Cloud de International Business Machines Corporation, Google Cloud Plataform de Google, Microsoft Azure de Microsoft, Ionos [1]…

- **Clúster:** Agrupación de servidores que forman parte de la misma red cuya gestión de tareas la realizan conjuntamente.

- **Nodo:** Se puede definir como un servidor físico o en la nube con un sistema operativo configurado en su interior y con sus respectivos recursos de CPU, RAM y sistema de archivos local (Disco) asignados.

2.1.2. Infraestructura contenedores y virtualización

- **Contenedor:** Tecnología que se usa para agrupar una aplicación con todos sus archivos necesarios en un entorno de ejecución. Además,

puede moverse con facilidad y ejecutarse en cualquier sistema operativo en cualquier contexto [2].

- **Docker:** Utilizado para la ejecución de contenedores. Programa Open Source que te permite: Crear, instalar, ejecutar y modificar contenedores [3].

- **Kubernetes:** Es un orquestador de contenedores, open source, que se encarga de administrar y gestionar contenedores que ya existen y de posicionarlos en los diferentes servidores, que forman parte de la infraestructura, dependiendo de los recursos disponibles en cada uno de ellos [4].

- **Docker Swarm:** Orquestador de contenedores, open source, similar a Kubernetes, que se encarga de administrar y gestionar contenedores dentro de una infraestructura [5].

2.1.3. Servicios

- **Elastic Stack:** Son un conjunto de herramientas utilizadas para la captación, procesado, visualización y almacenamiento de datos.
  Este conjunto está compuesto por: Elasticsearch, Kibana, Logstash y Beats [6].

- **Kibana:** Panel de visualización de datos de Elastic.

- **Metricbeat**: Uno de los agentes que nos proporciona Beats utilizado principalmente para la recolecta de datos de métricas de recursos de servidores, como datos del sistema operativo.

- **Grafana:** Permite visualización personalizada de datos con abundante variedad de cuadros de mando y gráficos [7].

2.2. Estudio y validación de las tecnologías de monitorización

Se ha realizado un estudio de las posibles tecnologías que se podrían encontrar en el mercado actualmente. Aunque la empresa ya tenía preseleccionadas una serie de tecnologías para hacer la monitorización y por lo tanto las valoraremos.

2.2.1. Estudio de algunas de las tecnologías de monitorización actuales

Elastic Stack [6]

Es una herramienta muy potente que nos proporciona a partir de los datos almacenados: obtener, procesar y realizar búsquedas de datos de una manera eficiente, rápida y en tiempo real.

Se ha mencionado la primera, ya que es la alternativa más fuerte del mercado. Ofrece agentes para la recolección de datos (tanto de registros como de métricas del sistema), base de datos de alto rendimiento, una atractiva visualización de los datos y medida de rendimiento de aplicaciones.

Todo y que la visualización de los datos es muy atractiva, ofrece menos funcionalidades que otras tecnologías como Grafana.

Prometheus [8]

Es software enfocado puramente a la recogida y almacenamiento de métricas del sistema, al igual que otras tecnologías como InfluxDB. No tiene visualizadores de datos, y es por esto que normalmente se despliega juntamente con Grafana.

Dynatrace [9]

Se vende como una solución completa (muy costosa económicamente) que ofrece todas las funcionalidades que un sistema de monitoreo podría necesitar. Pero es un software propietario y muy cerrado. Esto dificulta las opciones de integración de otras herramientas, agregar nuevas funcionalidades, monitorizar sistemas o procesos que por defecto no haga. La parte de visualización de datos, también es bastante pobre.

2.2.2. Validación de las tecnologías de monitorización

Dynatrace, aunque parece la solución más adecuada, sólo ofrecería un sistema de monitorización cerrado y sin muchas posibilidades de expansión. Así que no sería una buena candidata como tecnología utilizada para la monitorización en mi proyecto.

Prometheus, tampoco habría sido seleccionada para mi proyecto, ya que Elastic ofrece lo mismo y muchas otras funcionalidades.

Elastic Stack, es una de las herramientas que juntamente con Grafana al tener mucha sinergia juntas, se validan como integración de solución completa para el despliegue del sistema de monitorización de mi proyecto.

2.3. Tecnología utilizada como herramienta de orquestación

La empresa con la que se realiza el TFG, requiere que el despliegue se haga con la herramienta de orquestación Kubernetes. Esta empresa tiene experiencia con otra tecnología de orquestación, Docker Swarm, y al ver que tiene ciertas limitaciones quiere cambiar a Kubernetes ya que cree que cubre esas limitaciones.

Otro motivos por lo que se utilizará esta tecnología son:

- Tecnología que nos permite realizar despliegues de infraestructura de manera sencilla, fiable y escalable.

- Open Source, ecosistema grande y en constante crecimiento.

- Utilizado por la mayoría de empresas tecnológicas a nivel mundial como Amazon, Azure, Google, Red Hat, IBM, VMware y Oracle.

2.3.1 Componentes principales que conforman Kubernetes

Explicados los motivos por los cuales se utilizará esta tecnología, para saber cómo funciona y qué componentes son los que la conforman, se realizará una explicación de cada uno de los componentes principales de Kubernetes:

Pod: Es el objeto más pequeño y simple de Kubernetes. Es una abstracción por encima de los contenedores, utilizado para el amacenamiento de los mismos (figura 2.1).

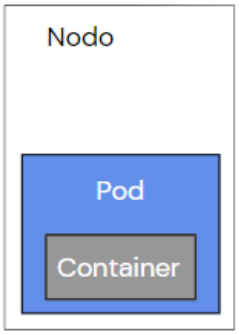

Fig. 2.1 Ejemplo de Pod

Deployment: Es un conjunto de pods que contienen las mismas configuraciones y aplicaciones (figura 2.2).

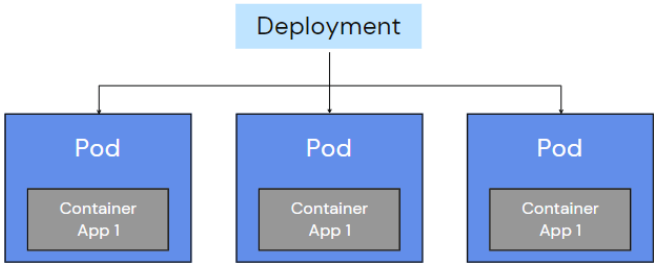

Fig. 2.2 Ejemplo de creación de un Deployment con tres réplicas

Service: Es una dirección IP estática/permanente que puede ser adjuntada a varios pods a la vez siempre y cuando contengan la misma aplicación (figura 2.3).

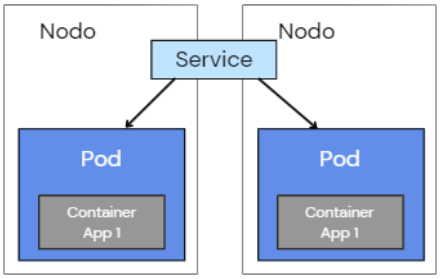

Fig. 2.3 Ejemplo de Service

Ingress: Nos proporciona acceso a los servicios creados desde internet (figura 2.4).

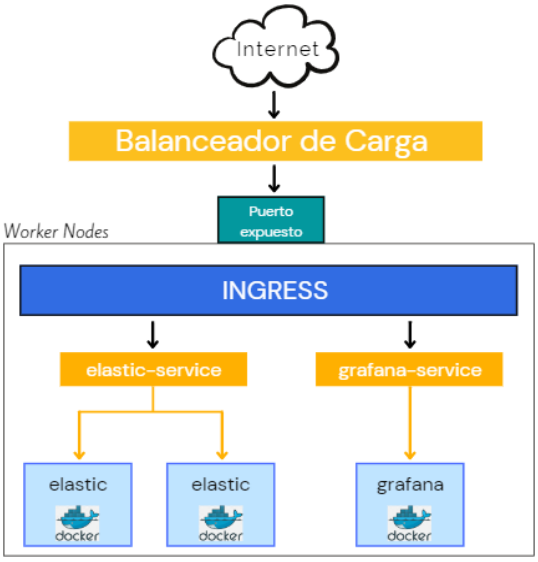

Fig. 2.4 Acceso a servicios diferentes mediante Ingress

Namespace: Este componente organiza los recursos en Namespaces (figura 2.5). Además, es utilizado principalmente para separar despliegues y tener mayor control de acceso a los mismos.

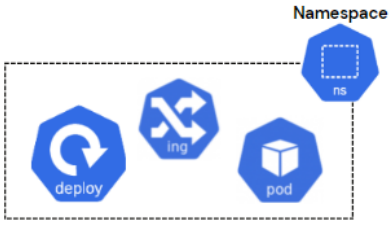

Fig. 2.5 Ejemplo de Namespace con componentes dentro

Persistent Volume (PV): Componente abstracto que nos permite obtener recursos a partir del almacenamiento físico disponible de los servidores. Estos recursos pueden ser: RAM, CPU, disco entre otros (figura 2.6).

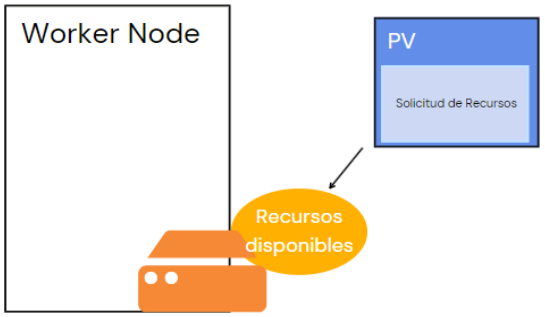

Fig. 2.6 Ejemplo de petición de recursos físicos mediante PV

Persistent Volume Claim (PVC): Reclama un Persistent Volume creado previamente (figura 2.7).

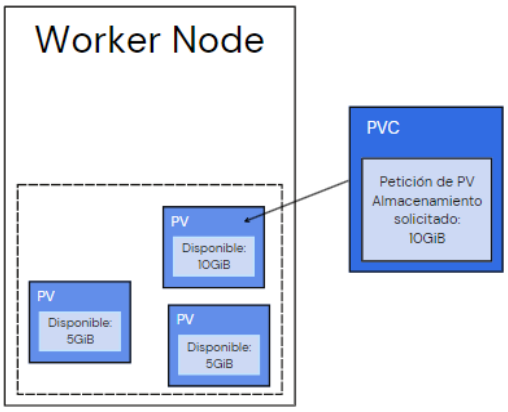

Fig. 2.7 Solicitud de PV que cumpla las condiciones del PVC

Este componente, es el encargado de buscar si algún Persistent Volume cumple con las condiciones de los recursos solicitados. En el caso que el Persistent Volume coincida con la petición solicitada por este componente, automáticamente se establece la conexión del PVC con el PV.

Storage Class: Componente que se encarga de provisionar PV's dinámicamente (figura 2.8).

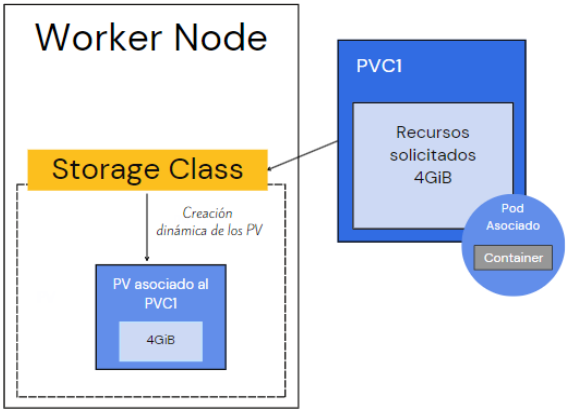

Fig. 2.8 Asignación de recursos utilizando Storage Class

Mientras que la creación de los PV era administrada por los operadores manualmente, con Storage Class se realiza de manera automática.

2.3.2 Tipos de Nodos de Kubernetes

Kubernetes básicamente está compuesto por dos tipos de nodos: Master Node y Worker Node.

Los nodos que se utilizan para planificar y gestionar los pods son los Worker Nodes (figura 2.9).

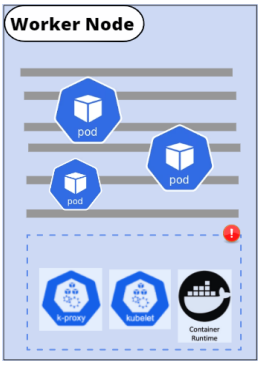

Fig. 2.9 Ejemplo de nodo Worker Node

Mientras que los Master Nodes, como su propio nombre indica, son los nodos encargados de administrar toda la infraestructura de Kubernetes y asegurarse del buen funcionamiento del misma (figura 2.10).

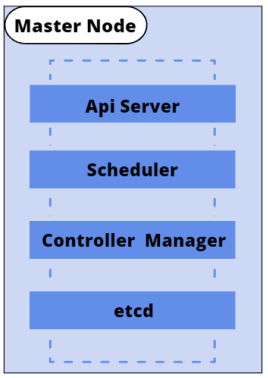

Fig. 2.10 Ejemplo de nodo Master Node

Para poder desplegar Kubernetes, es necesario instalar estos tres procesos en cada uno de los Nodos:

- *Container runtime:* Planifica y ejecuta contenedores.

- *Kubelet:* Encargado de programar y arrancar un pod con un contenedor dentro.

- *Kube proxy:* Es el responsable de reenviar las peticiones desde los services hacia los pods.

Para la administración de la infraestructura de Kubernetes, los Master Nodes utilizan estos cuatro procesos:

- *API Server:* Es el punto de entrada que permite operar a los usuarios con la infraestructura de Kubernetes.

- *Scheduler:* Encargado de elegir en qué Worker Node se iniciará un nuevo componente.

  Para tomar esta decisión lo primero que hace el Scheduler es mirar la petición y ver cuántos recursos son necesarios para programar ese componente (Cuanta CPU, RAM y Disco son necesarios).

  Lo siguiente que hará es mirar la disponibilidad de recursos en cada uno de los Worker Nodes. El Scheduler programará el nuevo componente en el nodo que esté menos ocupado (menos % usado), o lo que es lo mismo, que tenga más recursos disponibles.

  Por último, kubelet se encargará de iniciar el componente en el nodo programado por Scheduler.

- *Controller Manager:* Encargado de detectar cambios de estado en la infraestructura de Kubernetes: pods rotos, despliegues que no se han podido realizar con éxito…

  Cuando detecta un cambio de estado, prueba de recuperar su estado tan pronto como sea posible: Notifica al Scheduler que se encargará de planificar en qué Worker Node se redesplegará ese componente y seguidamente kubelet iniciará el despliegue en ese nodo.

- *ETCD:* Es el cerebro de la infraestructura de Kubernetes, almacén de clave-valor del estado del mismo.

  Los mecanismos de API Server, Scheduler, Controller Manager y otros componentes, funcionan debido a los datos almacenados en él. Además, forma un almacenamiento distribuido en todos los Master Nodes de la infraestructura de Kubernetes.

2.4. Recursos del cloud de Ionos disponibles para el proyecto

El proyecto ha sido desplegado en el cloud de Ionos por lo que todas las decisiones tomadas para el despliegue de la infraestructura de Kubernetes se han realizado en base a la infraestructura del cloud de Ionos proporcionada.

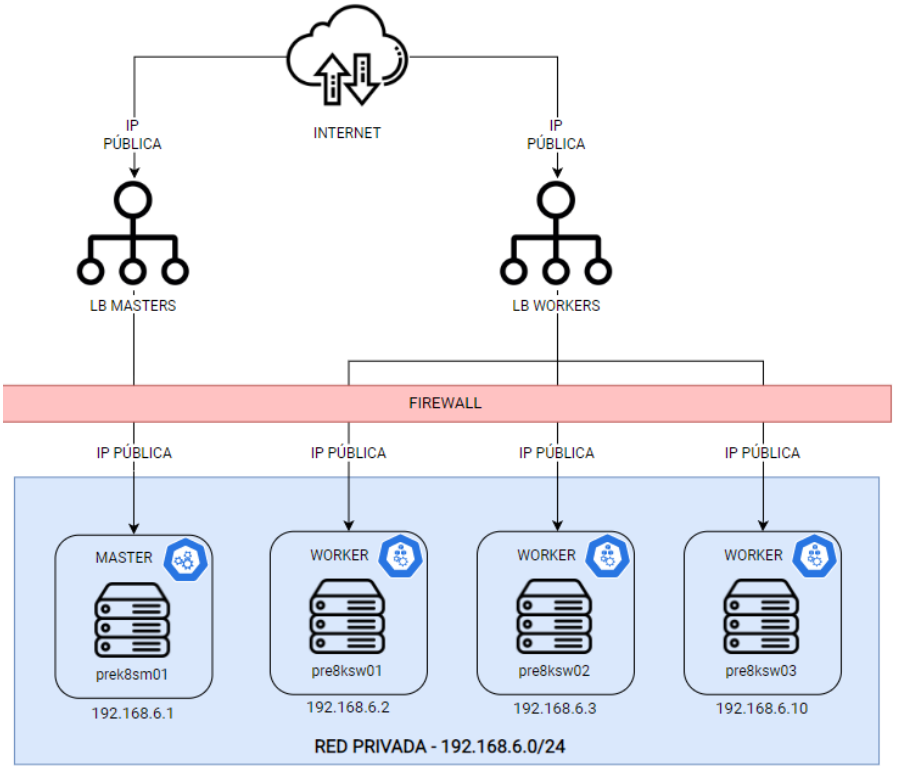

Fig. 2.11 Configuración inicial de la infraestructura en Ionos

En esta sección se muestra gráficamente la arquitectura utilizada para este proyecto (figura 2.11). Como se observa en el diagrama, la infraestructura de Kubernetes, estará compuesta por cuatro servidores: un Master Node y tres Worker Nodes.

Las IPs públicas asignadas para cada nodo son las mostradas entre las flechas y los servidores. Mientras que la red privada que se utilizará para la comunicación entre los diferentes nodos del Kubernetes es la siguiente: 192.168.6.0/24.

También se ha incorporado un sistema de firewall para limitar el acceso a la infraestructura de Kubernetes y proporcionar un entorno más seguro: Únicamente se podrá acceder a la infraestructura desde el exterior mediante la dirección IP pública de las oficinas de TobeIT.

En la parte superior del diagrama, se puede observar el uso de dos balanceadores: Un balanceador de carga para el Master Node (LB MASTERS) y otro para los Worker Nodes (LB WORKERS). Gracias a la incorporación de los balanceadores, obtendremos una mayor disponibilidad y un mejor rendimiento de nuestra infraestructura.

Aunque en este caso el balanceador del Master Node no es necesario, se implementa para posibles futuras incorporaciones de servidores que también harán de Master Node.

Por último, las IPs públicas que se visualizan arriba del todo del diagrama, hace referencia a las IPs públicas de los balanceadores. Estas IPs serán utilizadas como puntos finales (endpoints), que se encargaran de recibir las peticiones que provienen desde internet y redirigirlas hacia la infraestructura de Kubernetes.

Con esto cerramos el capítulo dos. En el próximo capítulo nos centraremos en la metodología aplicada.

3. Metodología

Una vez validadas las tecnologías de monitorización utilizadas y definida la arquitectura de la infraestructura de Kubernetes con la que partimos, se deberá definir una estrategia para poder desarrollar el proyecto de forma organizada y dentro del plazo límite establecido.

Por otra parte, lo que se pretende con este proyecto, es conseguir aplicar la metodología DevOps en empresas (prácticas que combinan el desarrollo con las tareas de un operador IT cómo serían: compilar, desplegar, configurar y monitorizar).

El director del proyecto será el encargado de las reuniones, pero de igual manera, podrá estar en cualquier momento si fuese necesario para poder ayudar en el desarrollo del mismo.

3.1 Metodología DevOps

¿Cómo funciona el desarrollo de un proyecto de DevOps?

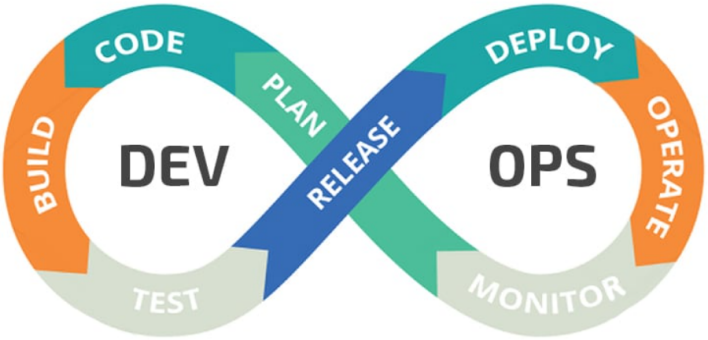

Fig. 3.1 Metodología DevOps (Chicharro, Jesús. “Sentando las bases de la metodología DevOps.”)

En la metodología DevOps, se sigue un ciclo de vida en el que intervienen 8 fases. En mi proyecto estás fases se han ordenado de la siguiente manera:

- *Plan:* En esta etapa se realiza el plan que se va a seguir en el momento de querer realizar un despliegue de infraestructura.

- *Code:* Una vez el plan se ha definido, se lleva a cabo la configuración de los scripts necesarios para la realización del despliegue.

- *Build:* En esta fase se realiza el despliegue ejecutando los scripts configurados anteriormente. Estos scripts, nos permiten obtener los contenedores que en conjunto hacen funcionar la infraestructura de Kubernetes.

- *Test:* A través de la interfaz de línea de comandos se verifica que el despliegue de los contenedores se ha realizado correctamente.

- *Deploy:* Se realiza el despliegue del sistema de monitorización encima de la infraestructura de Kubernetes creada.

- *Operate:* A partir de un agente instalado en el ordenador portátil que forma parte de la infraestructura interna de la empresa ToBeIT, se obtiene el tráfico de datos de las métricas y procesos del sistema operativo del ordenador portátil.

- *Monitor:* Se implementa en producción un sistema de monitorización completo en el que se obtienen métricas reales de un cliente final para posteriormente comprobar el rendimiento de la infraestructura del cliente. Si el rendimiento es el adecuado o en su contra, si hay fallas y posibles mejoras, volvemos a realizar una planificación nueva.
  Podemos llegar a estar en la etapa de monitorización infinitamente. Ya que siempre pueden haber posibles mejoras.

  En el caso de mi TFG, esta etapa no ha sido posible realizarla por parte del cliente final ya que el despliegue con el cliente se hará en fechas futuras.

3.2. Herramientas

3.2.1. Herramientas de gestión

Por tal de poder desarrollar el proyecto de una manera más organizada, se utilizarán diferentes herramientas que facilitarán el trabajo haciendo sesiones mucho más productivas.

- **Evernote** [10]**:** Herramienta de gestión de documentación. Nos permite crear carpetas donde podemos ir incorporando información en forma de notas de lo que se vaya investigando. Teniendo la posibilidad de incorporar código, tablas, esquemas y muchas más opciones por nota generada.

  También se puede utilizar para llevar un control de las actividades a realizar, ya que nos permite hacer listas de checks.

- **Todoist** [11]**:** Herramienta de gestión de tareas mediante el uso de notas (parecido a Trello). Utilizada para llevar un seguimiento de los objetivos marcados para la realización del trabajo.

- **Focus To-Do** [12]**:** Aplicación de gestión de tiempo. Utilizada para administrar las tareas y realizarlas de manera eficiente.

3.2.2. Herramientas de seguimiento

Se han utilizado las siguientes herramientas de seguimiento:

- **Git** [13]**:** Sistema de control de versiones que nos permite mejorar el flujo de trabajo.

- **GitHub** [14]**:** Servicio de alojamiento en línea por excelencia para proyectos en Git.

Estas dos herramientas en conjunto nos ayudan a tener un mayor control sobre todas las nuevas implementaciones que iremos haciendo durante el transcurso de este proyecto.

Por último, se ha utilizado **Visual Studio Code** [15], como herramienta utilizada para el desarrollo de los ficheros de configuración de los diferentes componentes y aplicaciones dentro de la infraestructura.

3.3. Definición del plan inicial

Debido a las dimensiones del proyecto y a realizarlo de manera individual, se ha optado por un método de planificación de desarrollo en cascada [16].

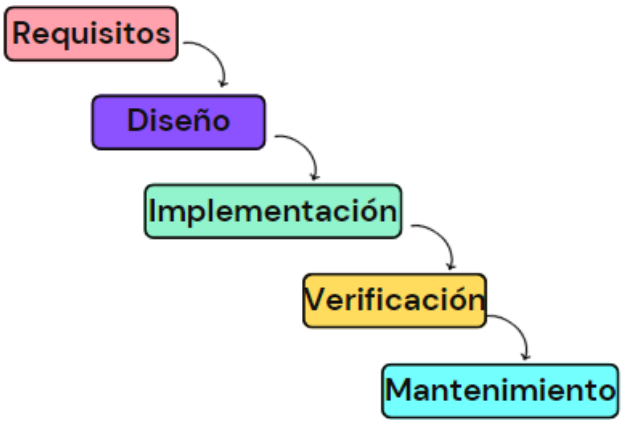

Fig. 3.2 Método de desarrollo en cascada

Es un enfoque metodológico que ordena en etapas el proceso de desarrollo del sistema. Al final de cada etapa se realiza una revisión que se encarga de evaluar si el producto está listo para pasar a la siguiente etapa. Éste es el modelo tradicional de desarrollo de software (figura 3.2).

Con tal de obtener una planificación organizada de los objetivos establecidos para mi proyecto y que estén dentro del tiempo estipulado para la entrega, se ha implementado un diagrama de Gantt, que presentaré a continuación:

TÍTULO DE LA TAREA	FECHA DE INICIO	FECHA FIN	DURACIÓN (DÍAS)
GESTIÓN DEL PROYECTO	**24/03/22**	**19/10/22**	205
Documentación del proyecto	24/03/22	19/10/22	205
Introducción	24/03/22	02/04/22	8
Conceptos y Tecnologías	02/04/22	14/05/22	42
Metodología	15/05/22	01/06/22	16
Diseño	02/06/22	18/06/22	16
Reunión de seguimiento 1	28/04/22	28/04/22	0
Reunión de seguimiento 2	20/05/22	20/05/22	0
Reunión de seguimiento 3	16/06/22	16/06/22	0
Reunión de seguimiento 4	03/09/22	03/09/22	0
Reunión de seguimiento 5	10/10/22	10/10/22	0
Reunión de seguimiento 6	15/10/22	15/10/22	0
Reunión de seguimiento 7	19/10/22	19/10/22	0
Tareas de desarrollo	**19/06/22**	**16/10/22**	117
Despliegue del clúster de Kubernetes	19/06/22	29/07/22	40
Despliegue de Elasticsearch y Grafana	01/08/22	22/08/22	21
Ficheros de configuración de las diferentes tecnologías de monitorización	30/07/22	01/08/22	1
Conclusiones	01/10/22	16/10/22	15
TAREAS DE IMPLEMENTACIÓN	**24/08/22**	**02/09/22**	8
Pruebas de evaluación	24/08/22	02/09/22	8

Fig. 3.3 Diagrama de Gantt

Para el desarrollo de este proyecto se han estructurado las tareas en dos fases: Fase de desarrollo y fase de implementación.

Cada fase se realizará de manera lineal: Primero desarrollando el entorno de Kubernetes (fase de desarrollo) y seguidamente implementando las tecnologías validadas previamente (fase de implementación).

Fase de desarrollo

- *Creación de la infraestructura de Kubernetes:* Paso inicial, la puesta en marcha del entorno donde se desplegará todo el sistema de monitorización.

- *Configuraciones de red y balanceo de carga de Kubernetes:* Pensar y configurar cómo comunicar los nodos de Kubernetes entre ellos y cómo se balanceará la carga.

- *Configuración almacenamiento persistente:* Debido a la volatilidad de los pods de Kubernetes, se debe configurar un almacenamiento persistente que puedan utilizar los servicios.

Fase de implementación

- *Despliegue de la infraestructura de las tecnologías de monitorización:* Se configuraran y ejecutarán los scripts necesarios para desplegar la infraestructura de monitorización.

- *Instalación agentes de Elastic:* Se instalarán los agentes de Elastic para recolectar la información necesaria para la monitorización. Concretamente Metricbeat.

- *Creación de paneles en Grafana:* Se crearán los paneles para la visualización de datos en Grafana.

Es posible que haya aspectos técnicos o tecnologías que no hayan sido previamente estudiadas. Por lo que la realización de estas fases, quizás lleve más tiempo del que en un principio se pretende y se llegue hasta la fecha límite de entrega asignada para mi TFG. Personalmente, considero que donde existe más riesgo es en la integración de todas las herramientas con el orquestador Kubernetes.

Durante el transcurso del Grado en Ingeniería de Telemática aprendemos a integrar diferentes herramientas entre sí (a través de las asignaturas como *Enginyeria d'Aplicacions* e *Infraestructures i Operació de Telecomunicacions*). Pero hemos utilizado tecnologías diferentes a las que nos pueden servir en este proyecto. Así que el proyecto en su totalidad, supone un gran reto.

Una vez vista la metodología y la planificación que se realizará, damos paso al capítulo siguiente.

4. Diseño

A partir de este capítulo, se empezará a utilizar el método de planificación anteriormente explicado.

Los temas que trataremos en esta sección principalmente son: Requisitos especificados por la empresa para la realización del montaje de la infraestructura, diseño de la infraestructura de Kubernetes y posibles mejoras para aumentar la disponibilidad de las tecnologías de monitorización que serán desplegadas.

4.1. Requisitos y especificaciones

Todos los requisitos y especificaciones dictados por la empresa y que se deberán cumplir se muestran a continuación:

Requisitos funcionales

- Se deberá desplegar la infraestructura de la infraestructura de Kubernetes mediante la infraestructura proporcionada por la empresa
- Se deberá desplegar un sistema de monitorización completo a través de la infraestructura desplegada por Kubernetes.
- Se deberá poder recibir datos en Elastic
- Se deberá poder visualizar gráficos de los datos en Grafana

Requisitos no funcionales

- Las conexiones con los diferentes servicios creados en Kubernetes, deben ser conexiones seguras.
- La infraestructura de Kubernetes deberá incluir tolerancia a fallos.
- La conectividad con los servidores deberá ser a través de la VPN propia de la empresa.
- Cada semana se tendrá que reportar los avances del proyecto al tutor del TFG.
- Todas las tardes laborales serán específicamente utilizadas para la realización del proyecto.

- Todas las pruebas que se realicen y sean pruebas de éxito deberán ser documentadas para el futuro despliegue de producción de la misma infraestructura.
- Se deberán realizar pruebas de rendimiento de mi equipo de empresa utilizando el sistema de monitorización completo a través de la instalación de Metricbeat.

Las especificaciones de infraestructura aportadas son las siguientes:

Tabla 4.1. Infraestructura (1/3)

Servidores	Sistema Operativo	CPU	RAM	SSD
prek8sm01	CentOS 7	2vCore	4 GB	40 GB
prek8sw01	CentOS 7	2vCore	4 GB	40 GB
prek8sw02	CentOS 7	2vCore	4 GB	40 GB
prek8sw03	CentOS 7	2vCore	4 GB	40 GB

Tabla 4.2. Infraestructura (2/3)

Almacenamiento de bloques	Tamaño	Asignado a
precstorw01	20 GB	prek8sw01
precstorw02	20 GB	prek8sw02
precstorw03	20 GB	prek8sw03
prelvmw01	60 GB	prek8sw01
prelvmw02	60 GB	prek8sw02
prelvmw03	60 GB	prek8sw03

Tabla 4.3. Infraestructura (3/3)

Red	
Políticas de firewall	Acceso permitido únicamente desde las oficinas de ToBeIT (conexión mediante VPN).
Balanceadores de carga	Balanceador de carga para los Worker Nodes. Balanceador de carga para los Master Nodes.
Red privada	192.168.6.0/24

4.2. Diseño de la infraestructura de Kubernetes

A partir de la validación tecnológica realizada y de las especificaciones de infraestructura detallada en las tablas anteriores, a continuación, se realizará el diseño de la infraestructura de Kubernetes.

En la siguiente figura, se muestra el diseño, los componentes y la conectividad entre los diferentes servidores que formarán parte de la infraestructura:

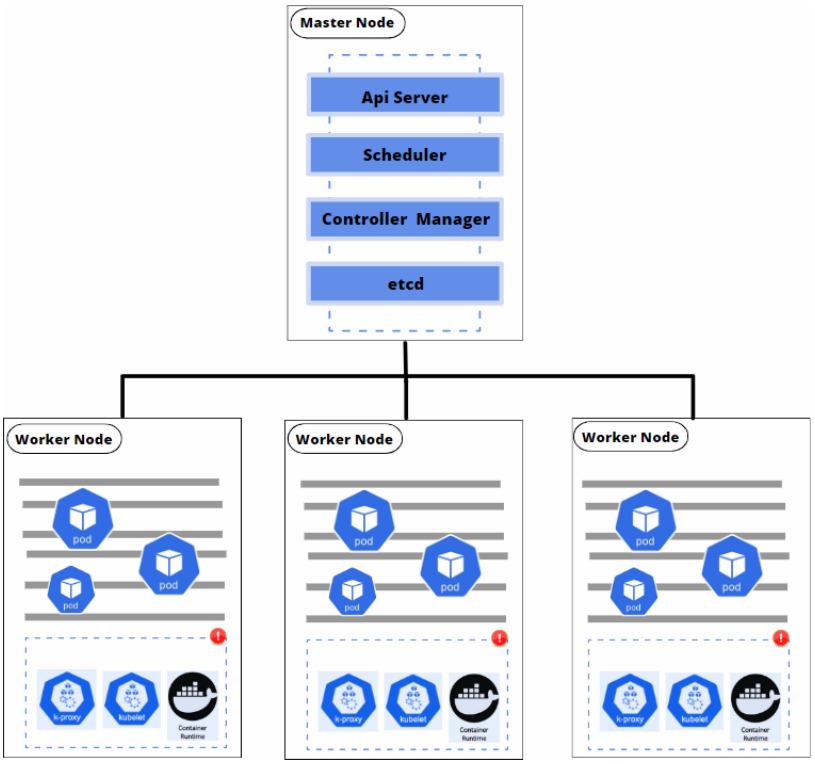

Fig. 4.1 Infraestructura de Kubernetes que será implementada

Este diseño se adapta a los recursos de los que disponemos del cloud de Ionos. Por lo que se ha decidido que la infraestructura de Kubernetes este formada por tres Worker Nodes y un Master Node.

Al no disponer de una infraestructura con gran número de nodos, la gestión de los despliegues de las diferentes tecnologías de monitorización y el control del buen funcionamiento de la infraestructura, lo puede efectuar un único Master Node. Mientras que los despliegues de los sistemas de monitorización, deben disponer de más de un Worker Node por si hiciese falta el uso de los recursos de los diferentes nodos.

4.2.1. Añadiendo un nuevo servidor Master/Worker Node

En el momento de implementar un Master Node o un Worker Node, tenemos que tener en cuenta diferentes factores:

- Un Master Node no necesita de muchos recursos (CPU, RAM, Disk). Ya que únicamente necesita los recursos necesarios para que la funcionalidad de los procesos que corren en él se ejecuten de manera óptima.

- En los Worker Nodes, al ser donde se despliegan los pods con los contenedores corriendo dentro, hay alta carga de trabajo. Por lo que se necesitan más recursos que en los Master Nodes.

- Los procesos que corren en el Master Node, son mucho más importantes que los que corren en los Worker Nodes. Sin el buen funcionamiento de los procesos del Master Node, la infraestructura tiende al mal funcionamiento de la misma.

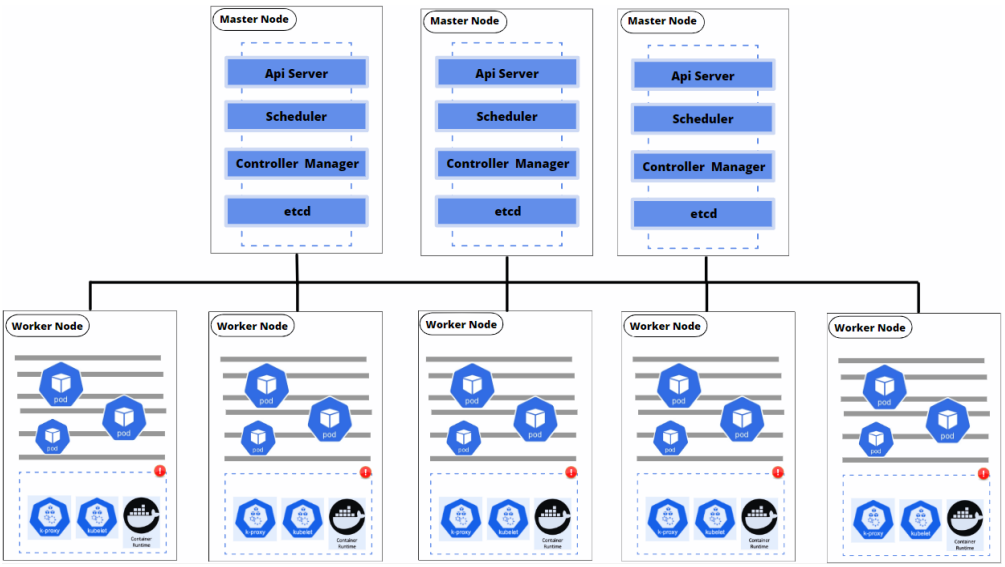

Fig. 4.2 Ejemplo de configuración de una infraestructura robusta

En el caso de que se requiera incorporar nuevos nodos a la infraestructura, ya sea por la alta demanda o por querer obtener una infraestructura más robusta (figura 4.2), se haría de manera muy sencilla y cómoda:

1. Se obtiene un servidor dedicado.

2. Se instalan los procesos necesarios dependiendo si queremos un Worker o Master Node.

3. Se añade a la infraestructura este nuevo nodo.

Tras haber desarrollado el diseño de la infraestructura de Kubernetes a través de las especificaciones otorgadas por la empresa (en el cloud de Ionos) y visto las posibles mejoras para aumentar la disponibilidad de la infraestructura a desplegar, iniciaremos el siguiente capítulo en el que daremos paso al despliegue de la infraestructura de Kubernetes.

5. Despliegue de la infraestructura de Kubernetes

Con los requisitos especificados y con el diseño de la infraestructura establecido, ahora comenzaremos con la instalación y configuración de la infraestructura de Kubernetes.

5.1. Mis decisiones de mi TFG

Como bien se ha mencionado anteriormente, se ha establecido Kubernetes como un orquestador de aplicaciones contenedorizadas junto a las tecnologías Elastic y Grafana para la realización de un despliegue de monitorización completo.

Aunque la utilización de las tecnologías anteriores ha sido decisión de la empresa, la manera de gestionarlas, configurarlas y desplegarlas ha sido decisión propia.

5.2. Creación de la infraestructura

5.2.1. Prerrequisitos

Desarrollaremos nuestra infraestructura de Kubernetes con los diferentes servicios a través de la infraestructura definida por ToBeIT. Donde hemos visto que tendremos un total de cuatro servidores.

Para crear la infraestructura, revisamos la documentación oficial para tener en cuenta los prerrequisitos y requisitos mínimos que son necesarios para desplegar una infraestructura de Kubernetes [17].

De los prerrequisitos que se piden, realmente solo es necesario desactivar la swap de todos los servidores y abrir los puertos que se especifican para la buena comunicación entre los componentes de la infraestructura y para el buen funcionamiento de los mismos. Esta configuración se realiza para cada uno de los nodos que formarán parte de la infraestructura de Kubernetes.

Respecto al apartado de los recursos, se nos especifica que cada servidor que va a formar parte de la infraestructura debe tener mínimo 2 CPU y 2GB RAM.

Estas especificaciones se cumplen desde un inicio. Ya que ToBeIT nos proporciona para cada servidor los recursos suficientes como para realizar el despliegue de Kubernetes sin tener la necesidad de aumentar los recursos de los servidores.

A continuación, se explicará cómo se ha realizado la instalación de la infraestructura sin entrar al detalle de los comandos a ejecutar en cada uno de los servidores. Si se quiere ver cómo realizar una instalación en detalle, se puede consultar la documentación oficial [18].

5.2.2. Configuración de la infraestructura

Una vez realizado los prerrequisitos empezamos con el despliegue de la infraestructura:

Lo primero que instalamos en cada nodo de la infraestructura es:

- *Docker Engine:* Tecnología escogida por defecto como container runtime.
- *Kubelet:* funcionamiento explicado anteriormente en la sección 2.3.2.
- *Kubeadm:* Herramienta de línea de comandos para inicializar la infraestructura.
- *Kubectl:* Herramienta de línea de comandos para interactuar con la infraestructura.

5.2.2.1. Inicialización de la infraestructura

Ahora ya podemos inicializar la infraestructura con un primer Master Node. Para ello, se hace uso de kubeadm:

```
root@prek8sm01:~# kubeadm init --control-plane-endpoint \
"<ip publica load balancer masters>:8443" --upload-certs
```

Fig. 5.1 Inicialización de la infraestructura de Kubernetes con un Master Node

Siendo la IP y puerto el del balanceador previamente configurado en IONOS. Este comando puede tardar algunos minutos en finalizar. Cuando acabe podemos ver en el output el comando necesario para que otros nodos puedan unirse a la infraestructura:

```
kubeadm join <ip publica load balancer masters>:8443 --token XXX \
--discovery-token-ca-cert-hash sha256:XXX \
--control-plane --certificate-key XXX
```

Fig. 5.2 Comando para que los nodos puedan unirse a la infraestructura

Este comando muestra los certificados necesarios para unirse a la infraestructura. Tienen una caducidad de 2h, por tanto, si se quieren añadir nuevos nodos, tenemos una ventana de 2h para hacerlo.
El comando *kubeadm init* nos habrá creado un archivo de configuración que nos permitirá conectarnos a la infraestructura y poder administrarlo. Este archivo está localizado en /etc/kubernetes/admin.conf.

Otras configuraciones de las que se encarga kubeadm cuando inicializamos la infraestructura, son:

- Genera los certificados necesarios para que la comunicación entre los diferentes componentes de la infraestructura sea segura. Utilizando PKI, Public key infrastructure.

- Genera ficheros kubeconfig. Estos ficheros permiten que tanto kubelet, controller-manager y scheduler, puedan conectarse a la API server. De la misma manera permiten la administración, tanto desde el interior como desde el exterior, de la infraestructura.

- Despliega los addons (plugins) esenciales para el buen funcionamiento de la infraestructura. Estos plugins son: DNS server y kube-proxy.

En el caso de que se quiera conectar de forma remota y administrar la infraestructura, debe descargarse el archivo *admin.conf* del servidor y guardarlo en local. Posteriormente, descargar la herramienta kubectl en local y hacer que utilice el archivo previamente descargado como configuración de la conexión con la infraestructura.

5.2.2.2. Container Networking Interface e incorporación de nodos

Una vez tenemos la infraestructura ya iniciada, debe instalarse el plugin de red (***CNI***, *Container Networking Interface*). Este plugin es indispensable para cualquier infraestructura de Kubernetes, ya que es el encargado de crear la red interna para que los Pods puedan comunicarse. Existen diferentes plugins, pero el más popular y el que ofrece un abanico más amplio de funcionalidades es **calico** [19].

Ahora sólo falta añadir los nuevos nodos a la infraestructura. Utilizaremos los comandos que se han visto previamente en la creación de la infraestructura

(kubeadm init). Aquí debe tenerse en cuenta que el comando variará dependiendo de si se une un master o worker node. Para los masters se añadiría lo siguiente en *kubeadm join*: ***–control-plane –certificate-key XXXXXX***.

La imagen final después de añadir todos los nodos es la siguiente:

```
# kubectl get nodes
NAME        STATUS   ROLES                  AGE     VERSION
prek8sm01   Ready    control-plane,master   2d17h   v1.23.5
prek8sw01   Ready    <none>                 2m55s   v1.23.5
prek8sw02   Ready    <none>                 2m56s   v1.23.5
prek8sw03   Ready    <none>                 2m57s   v1.23.5
```

Fig. 5.3 Nodos que conforman la infraestructura de Kubernetes

El resultado de las herramientas y componentes instalados en cada nodo se muestra en la figura 5.4.

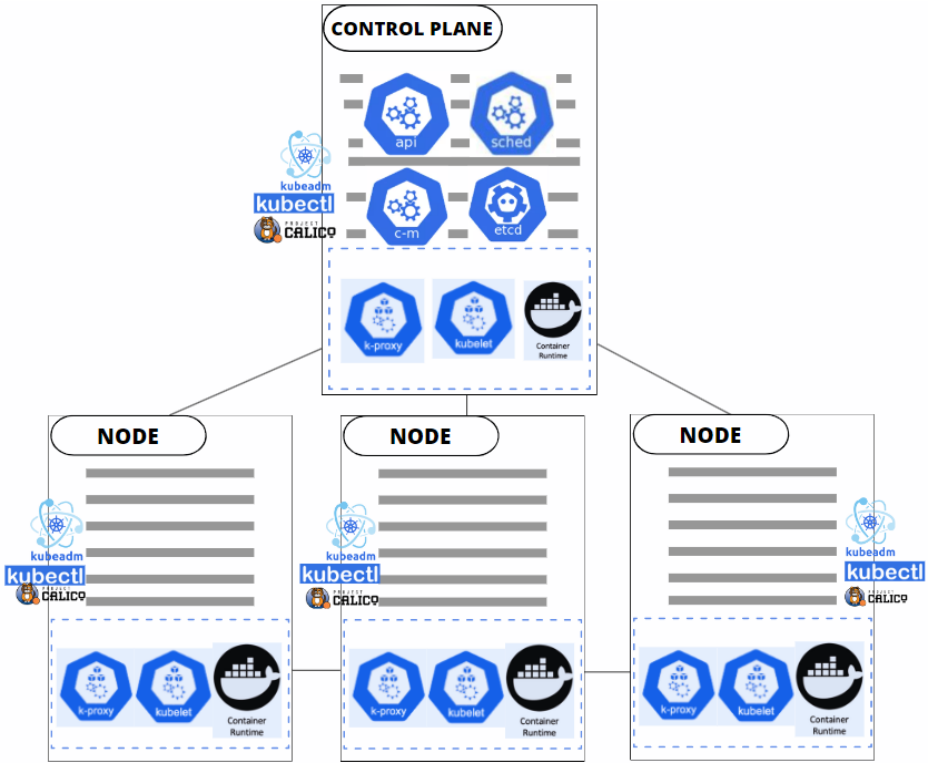

Fig. 5.4 Infraestructura de Kubernetes: Configuración básica.

Respecto a la seguridad de la comunicación entre los diferentes componentes de la infraestructura, se utilizará TLS. Este encriptado de la comunicación nos lo proporcionará también kubeadm cuando se ejecute por primera vez, en el arranque de la infraestructura.

5.2.2.3. Ingress Controller

Lo siguiente a configurar una vez están todos los componentes instalados es el **Ingress Controller**. Utilizado para poder acceder desde el exterior a los servicios que creemos dentro de la infraestructura. Este componente, será la única entrada posible hacia los servicios dentro de la infraestructura de Kubernetes.

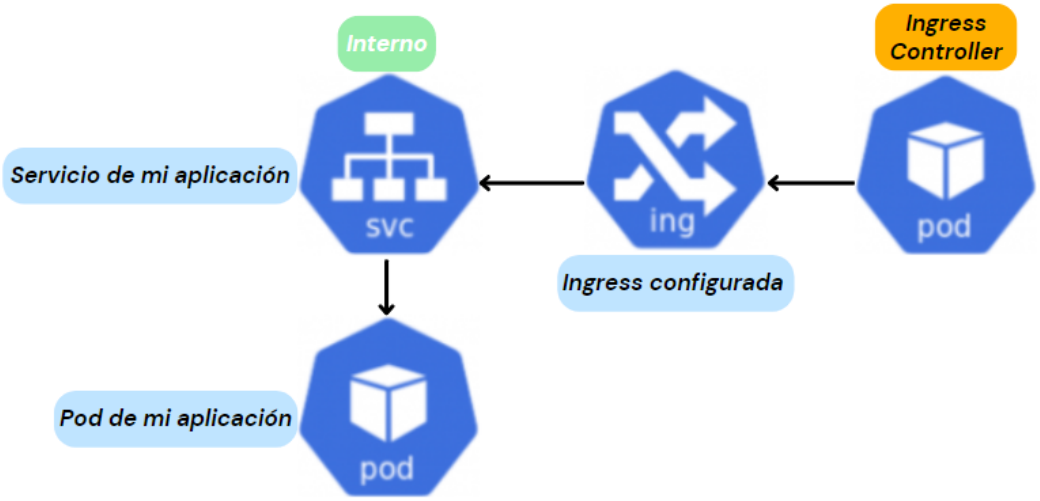

Fig. 5.5 Proceso de configuración de una regla en el Ingress Controller

El reenvío de las peticiones a los diferentes servicios que hayamos configurado, se hará mediante la observación de las rutas que tenga almacenado el pod Ingress Controller en su interior. Para incorporar rutas dentro del Ingress Controller, lo haremos a través del componente **Ingress** [20]. Este componente se encargará de crear rutas, asociarlas con servicios ya creados y envíar estás configuraciones al Ingress Controller. Todo este proceso lo podemos visualizar en la figura 5.5.

Existe una lista de los diferentes tipos de Ingress Controller que se pueden configurar [21]. Para este proyecto, se utilizará **Ingress NGINX Controller** ya que tiene soporte por parte del proyecto propio de kubernetes y está en constante mantenimiento.

Para la configuración del Ingress Controller instalaremos y utilizaremos el gestor de paquetes **Helm** [22] (administrador de paquetes para Kubernetes).

Helm nos ayudará a realizar el despliegue automáticamente sin la necesidad de tener que crear los componentes que conforman el Ingress Controller.

La guía seguida para la instalación del Ingress Controller se puede observar en la documentación oficial [23].

Una vez instalado, observamos en la siguiente imagen los componentes que se han generado para el funcionamiento de este controlador:

```
# kubectl get svc -n ingress-nginx
NAME                                 TYPE        CLUSTER-IP      EXTERNAL-IP    PORT(S)
ingress-nginx-controller             NodePort    10.108.23.152   82.223.6.122   80:30080/TCP,443:30443/TCP
ingress-nginx-controller-admission   ClusterIP   10.96.107.195   <none>         443/TCP
# kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS    RESTARTS
ingress-nginx-controller-f9d6fc8d8-g8r7q    1/1     Running   0
ingress-nginx-controller-f9d6fc8d8-tbll9    1/1     Running   0
```

Fig. 5.6 Ingress Controller

Disponemos de dos servicios y dos pods. El servicio ingress-nginx-controller, espera peticiones de la IP pública del balanceador LB WORKERS por los puertos 30080 y 30443 de los Worker Nodes. Por lo que si una petición entra por uno de esos dos puertos, será redirigida inmediatamente a los servicios correspondientes.

Estos servicios, dependiendo desde qué puerto de los Worker Nodes haya llegado la petición, redirigiran la petición por el puerto 80 o 443 hacía cualquiera de los pods desplegados del ingress controller. A su vez, estos pods, si tenemos alguna ruta configurada por el componente ingress que coincida con la petición, la redirigirá al servicio y puerto correspondiente.

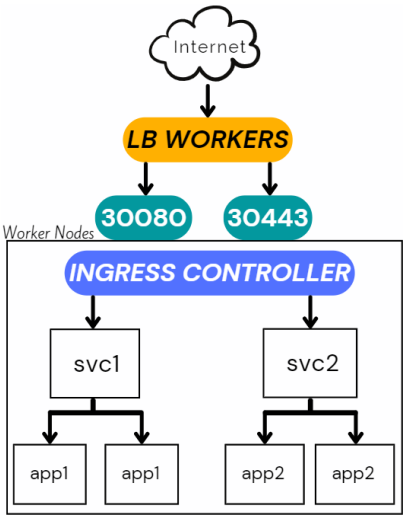

Fig. 5.7 Acceso a los servicios internos mediante Ingress Controller

El servicio ingress-nginx-controller-admission es utilizado para realizar procesos internos para el buen funcionamiento del ingress-nginx-controller.

Respecto a los pods generados, en este caso, se ha realizado una réplica del ingress-nginx-controller. Se recomienda que en los despliegues del ingress controller, mínimo se tenga una réplica, aunque lo ideal es tener tantos pods de ingress controller como Worker Nodes dispongas en la infraestructura. Esto es necesario ya que si un pod muere o un Worker Node cae, en el caso de tener más de un pod de Ingress Controller, no dejamos a la infraestructura sin rutas hacia los servicios. Ya que estos pods almacenan en su interior las asociaciones de rutas y servicios internos de la infraestructura de Kubernetes.

Para visualizar cómo se accedería a los servicios internos de la infraestructura mediante la incorporación del Ingress NGINX Controller observamos la figura 5.7 mostrada anteriormente.

5.2.2.4. Creación de dominios y gestor de certificados

Con el Ingress Controller incorporado a la infraestructura, ya sería posible crear cualquier nuevo servicio con su propio dominio. Pero en contra, aún no se han configurado ni asociado dominios propios hacia ninguna IP. Además, los servicios que se generen en un futuro, al no crear ni tener certificados de autoridades de certificación reconocidas por los navegadores, la conexión hacia nuestros servicios se establecería como insegura en cualquier navegador.

Para solucionar estas problemáticas, se han elegido las tecnologías siguientes:

Para la creación de dominios, se ha utilizado **No-IP** [24]. Esta tecnología nos permite generar dominios con wildcard al principio. Este factor es interesante ya que cualquier dominio que se genere con esa wildcard, tendrá asociada la misma IP.

Como para entrar a los servicios que se encuentran dentro de la infraestructura es necesario dirigir las peticiones al balanceador de carga LB WORKERS, todos los dominios que vayamos a generar, apuntarán a la IP del LB WORKERS.

A continuación, vemos la asociación entre el dominio escogido con una wild card al principio, y la IP del balanceador de los Worker Nodes asociada a esos dominios.

DOMINIOS	IP ASOCIADA
*.dev.tobeit.net	82.223.6.122

Fig. 5.8 Asociación entre dominios escogidos e IP asignada

Para la creación de certificados se ha utilizado **let's encrypt** [25]. Esta tecnología, además de ser Open Source y tener un buen soporte técnico por la comunidad, genera certificados de confianza para cualquier navegador que se utilice.

En el momento que añadamos let's encrypt en la ingress de nuestros servicios, se generará un certificado válido, que emitirá let's encrypt, para el dominio (host) de ese servicio. Con lo cual a partir de ese momento, nuestro dominio dispondrá de un certificado de confianza y será confiable su acceso desde cualquier navegador.

Más adelante, en la configuración de los servicios que desplegaré, se mostrará cómo se incorpora let's encrypt en la ingress asociada a los servicios que generemos.

5.3. Configuración de los Storage Class

Antes de empezar con el despliegue de Elastic y Grafana, necesitamos que los pods que contengan los contenedores de estas aplicaciones y las nuevas aplicaciones que se puedan incorporar en el futuro, tengan persistencia de datos. Como está configurada ahora la infraestructura, si un pod que contiene la aplicación con datos dentro muere, esa información se pierde para siempre aunque se reinicie el pod.

Para evitar este problema, configuraré y asignaré los tipos de Storage Class necesarios para cada aplicación que vaya a implementar. De esta manera, obtendremos persistencia de datos y evitaremos la posible pérdida de los mismos.

La tecnología utilizada para la implementación de estos Storage Class, ha sido **OpenEBS** [26]. Esta tecnología, aparte de orquestar los volúmenes de Kubernetes, nos permitirá asociar los recursos físicos de almacenamiento que disponemos con los volúmenes persistentes de Kubernetes.

Para realizar la instalación de esta tecnología, se ha seguido la guía oficial [27].

Una vez OpenEBS ha sido instalado, es necesario saber qué tipo de Storage class, que nos proporciona OpenEBS, se quiere incorporar a la infraestructura. Ya que por defecto, en la instalación de OpenEBS, no se incorporan los motores de almacenamiento de los diferentes tipos de Storage Class. Por lo que una vez decidido cuales se van a utilizar, se instalarán sus motores de almacenamiento [28].

Dependiendo de los Storage Class escogidos, los despliegues de volúmenes lógicos que creemos, utilizarán los recursos físicos de almacenamiento disponible de una manera u otra.

5.3.1. Storage class escogido para Elastic

Para saber que tipo de Storage Class es más conveniente para esta tecnología, primero se ha realizado un análisis de la forma en que Elastic almacena sus datos.

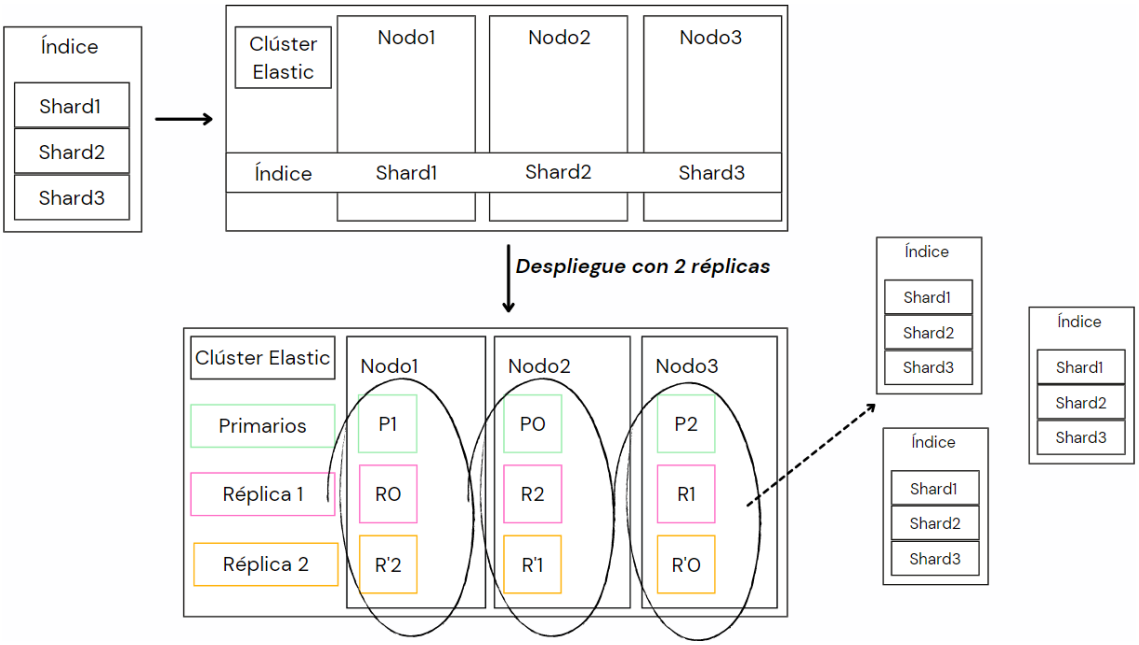

Fig. 5.9 Almacenamiento de datos en la infraestructura de Elastic

La manera de almacenar datos que tiene elastic es a través de índices. Estos índices pueden llegar a guardar datos de hasta 50 GB, aunque este valor es configurable. Además, los índices son fraccionados en fragmentos (shards). Cada fracción almacena parte de los datos del índice y en conjunto se obtiene el índice completo. Por norma general y por defecto, los índices están divididos en tres shards. Los cuales, están repartidos por los nodos de la infraestructura de Elastic (figura 5.9). Esta forma tan peculiar que tiene Elastic de guardar los datos es para que las búsquedas sean más rápidas y eficientes.

Otra particularidad de Elastic es la réplica de datos. Cuando se realiza un despliegue de Elastic, como en la figura 5.9 por ejemplo, los shards se pueden configurar para ser replicados tantas veces como nodos hayan en la infraestructura de Elastic. Podremos diferenciar entre shards primarios y shards replicados.

En el caso que se ilustra, se han generado dos réplicas de los shards primarios (mostradas en rectángulos rosas y naranjas). Estos shards replicados se reparten automáticamente entre los nodos de la infraestructura, pero, con una condición, el número de shard no puede coincidir con el mismo número de otro shard en el mismo nodo. Dicho de otra manera, el índice fragmentado en los diferentes shards, debe verse completado en cada nodo de la infraestructura.

Esta característica nos asegura consistencia de datos: En el caso que se cayese el Nodo1, por ejemplo, seguiría habiendo total disponibilidad de los índices en el resto de nodos.

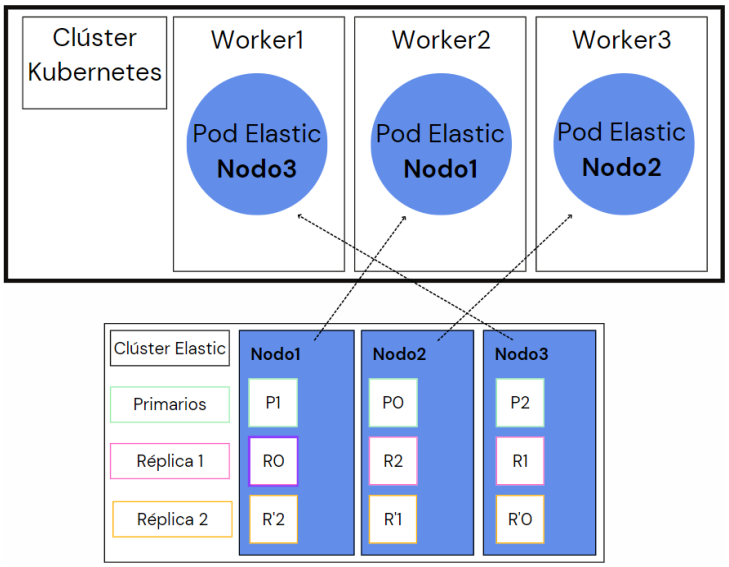

Fig. 5.10 Extrapolación de los nodos de Elastic a Kubernetes

El despliegue de Elastic sobre Kubernetes sería lo mismo, la diferencia es que en este entorno los nodos de la infraestructura de Elastic son los pods, figura 5.10.

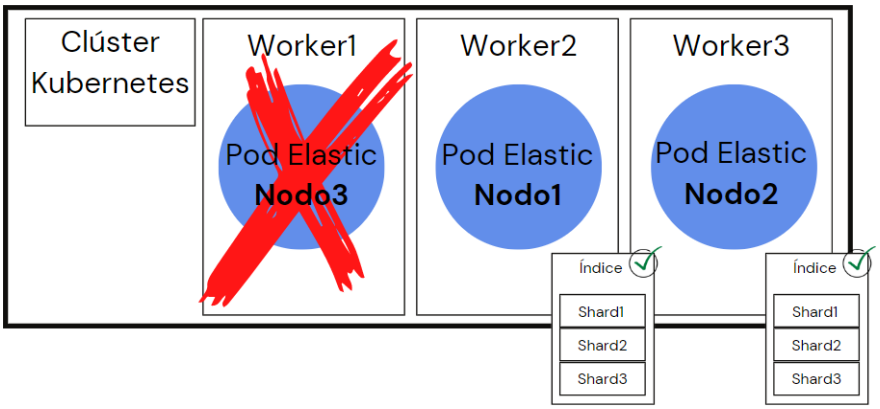

Fig. 5.11 Alta disponibilidad en la infraestructura de Elastic

En el caso que se cayese un pod de Elastic en uno de los Worker Nodes, aún permanecerán los pods en los otros Worker Nodes manteniendo la alta disponibilidad de la infraestructura de Elastic, figura 5.11.

Elastic, al ser una base de datos con estado (stateful) y de alta disponibilidad, no necesitará replicar la información de un nodo de Elastic a los diferentes Worker Nodes. Ya que los otros nodos, al tener réplicas de los índices en sus almacenamientos, no necesitan esta redundancia. Por lo que el sistema de Storage Class que se utilice, en este caso, tendrá que ser un almacenamiento persistente dedicado. De los posibles Storage Class que nos proporciona OpenEBS, se ha escogido **Local PV LVM** [29].

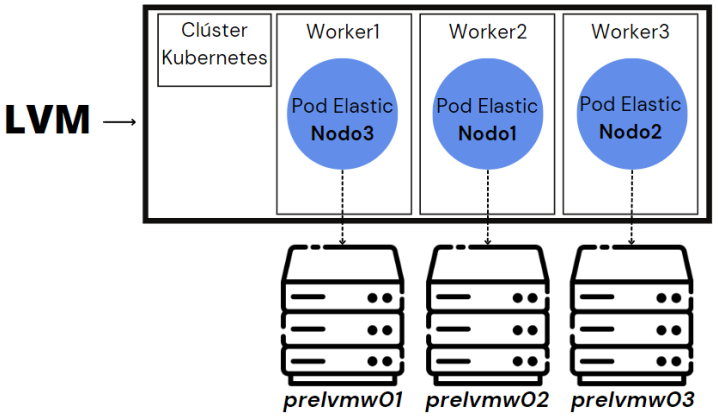

Fig. 5.12 Storage Class LVM de OpenEBS

Este Storage Class usa el almacenamiento como volúmenes físicos de cada uno de los nodos. Básicamente guarda los datos de forma local en el PV del Sistema Operativo y sin replicar. Es por eso que sólo se puede usar en aplicaciones que su tecnología ya replica los datos de alguna forma. El escenario de montaje que se realizará con este Storage Class se muestra en la figura 5.12.

5.3.2. Storage class escogido para Grafana

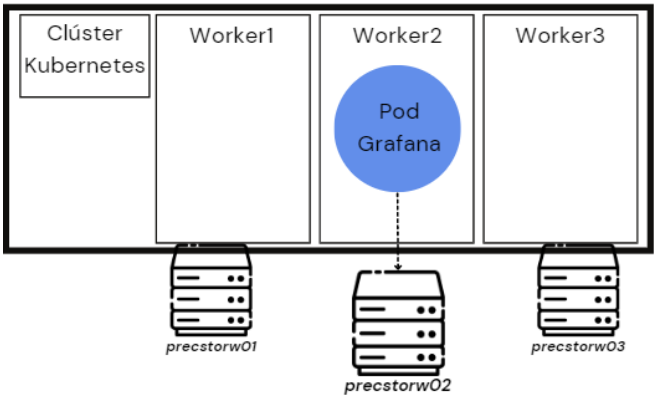

Fig. 5.13 Conexión de un PVC en precstorw02 con Grafana

En el caso de Grafana, se podría escoger un Storage Class como el escogido para Elastic: Grafana tendría su propio PV en el Worker Node donde se haya desplegado el pod, figura 5.13.

Pero esta solución, tiene un inconveniente. Ya que al ser una aplicación sin estado (stateless), no proporciona ni replicado de datos ni consistencia de los mismos. Por lo que si el pod de Grafana cae y se levanta en otro nodo de la infraestructura, no encontraría los datos de las configuraciones ahí, y se levantaría un Grafana vacío.

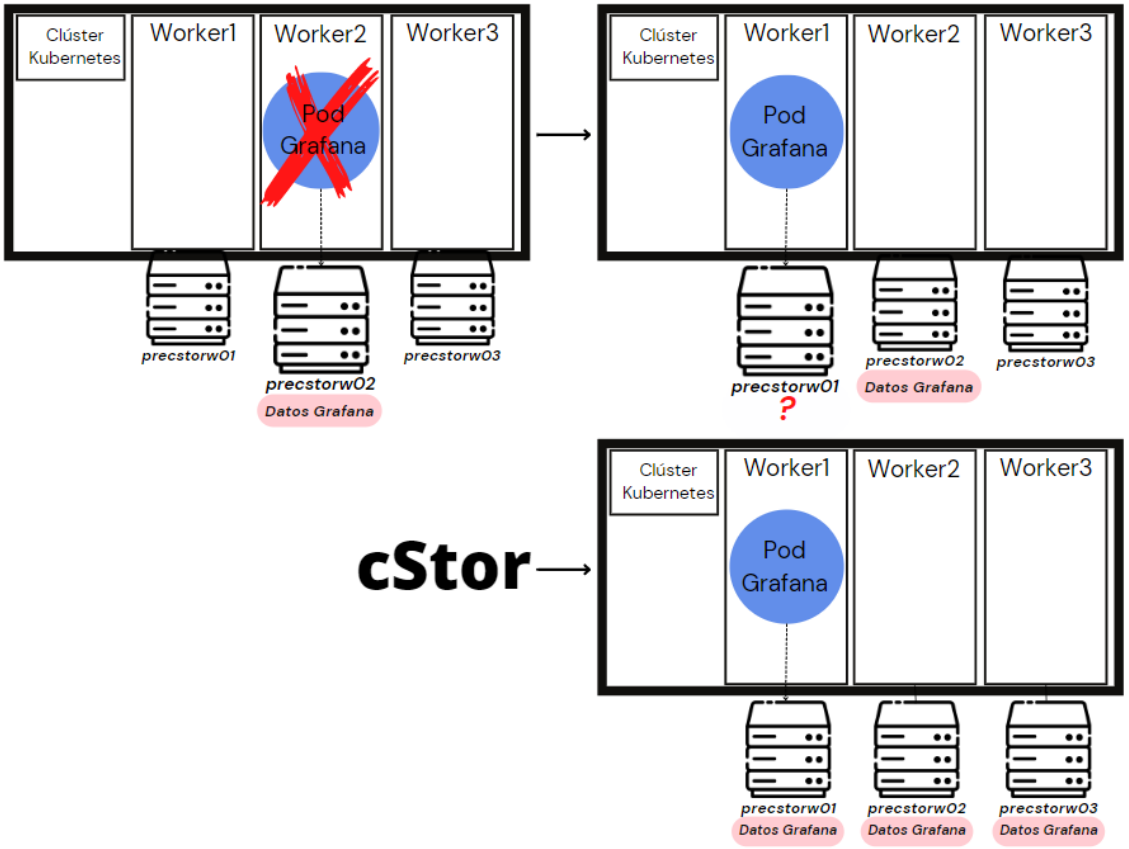

Fig. 5.14 Grafana con almacenaje persistente utilizando cStor Storage Class

Por ese motivo la solución es un Storage Class que replique los datos en los diferentes discos de la infraestructura, ya que no sabemos en que nodo se restablecerá el nuevo pod. El Storage Class que nos proporciona este tipo de almacenamiento es **cStor** [30].

A partir de que se configure el PV de Grafana con cStor, el pod que se despliegue siempre tendrá los datos disponibles, figura 5.14.

Una vez escogidos los Storage Class se procede a instalarlos mediante las guías [29] y [30]. Posteriormente, comprobamos que se haya realizado la instalación con éxito:

```
# kubectl get sc
NAME                      PROVISIONER           RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION
cstor-csi-disk            cstor.csi.openebs.io  Delete          Immediate              true
openebs-device            openebs.io/local      Delete          WaitForFirstConsumer   false
openebs-hostpath          openebs.io/local      Delete          WaitForFirstConsumer   false
openebs-lvmpv (default)   local.csi.openebs.io  Delete          WaitForFirstConsumer   false
openebs-rwx               openebs.io/nfsrwx     Delete          Immediate              true
```

Fig. 5.15 Openebs-lvmpv (LVM) y cstor-csi-dick (cStor) instalados

Tras haber desplegado la infraestructura de Kubernetes, en el siguiente capítulo veremos el despliegue de la infraestructura de monitorización completa.

6. Despliegue de Elastic y Grafana

Terminada la configuración de la infraestructura de Kubernetes, ahora es el momento de desplegar las tecnologías mencionadas previamente para realizar el sistema de monitorización.

Para la realización de este despliegue, únicamente ha sido necesario configurar tres pods de Elastic un pod de Kibana y un pod de Grafana (figura 6.1). Aunque realmente con el despliegue de un único pod de cada tecnología, ya sería suficiente para poder obtener un sistema completo de monitorización. Pero al configurar tres pods de Elastic, nos aseguramos de obtener una infraestructura de Elastic con alta disponibilidad y por lo tanto más robusto.

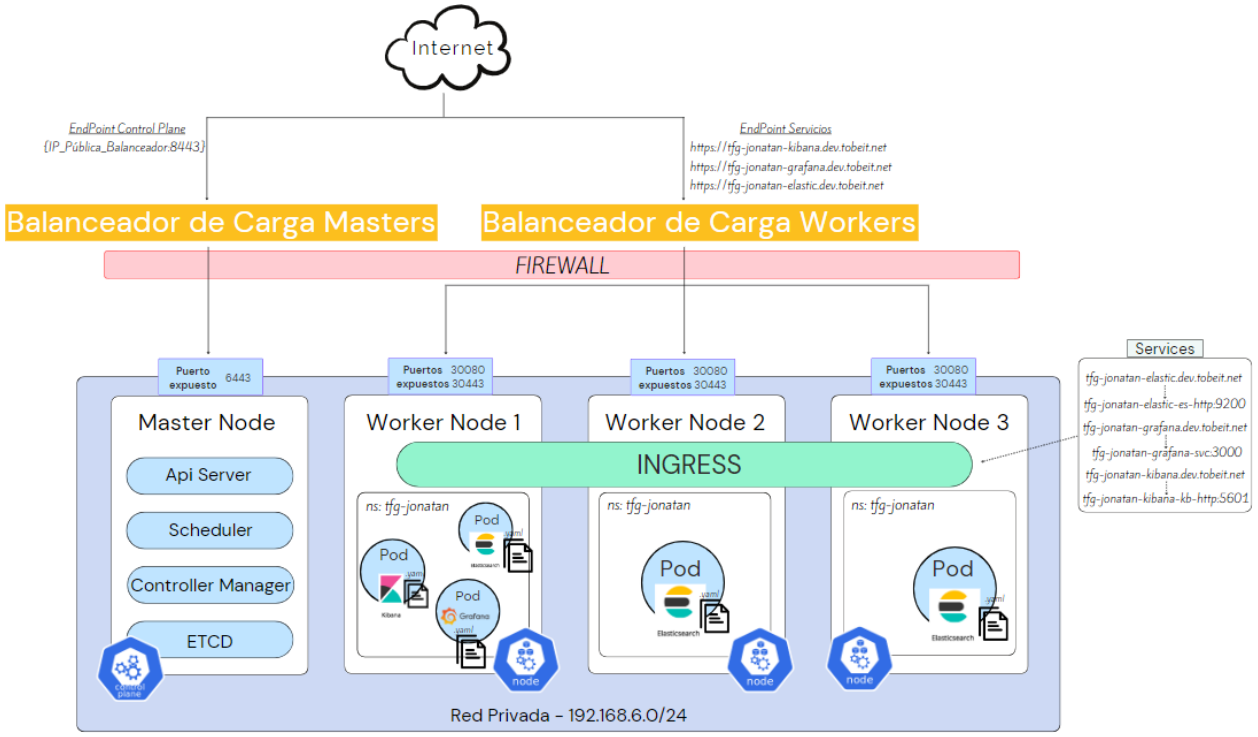

Fig. 6.1 Esquema explicativo de los servicios desplegados

También se ha configurado un namespace donde se ubicaran todos los pods desplegados (figura 6.2).

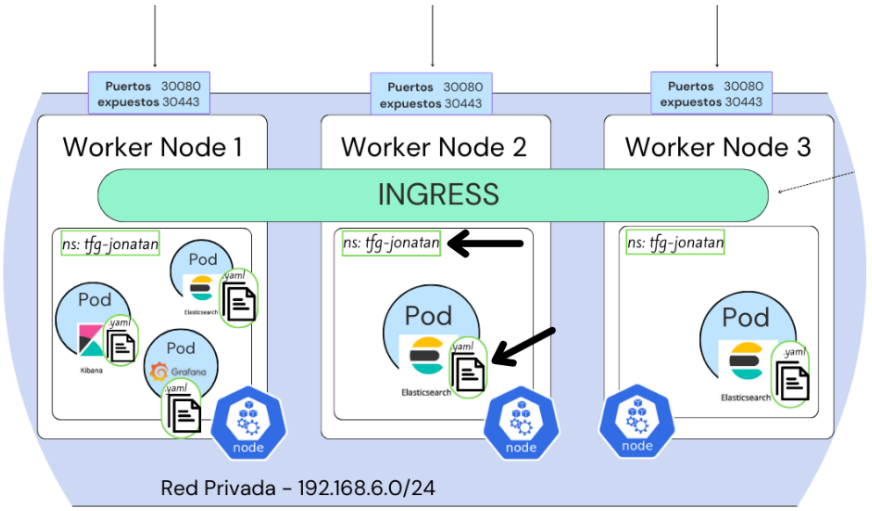

Fig. 6.2 Pods, namespace y ficheros de configuración establecidos

Los despliegues y las configuraciones de los pods, se efectúan mediante ficheros de configuración .yaml. En el momento que se aplican estos ficheros a través de la línea de comandos de kubectl, la instalación de Elastic, Kibana y Grafana se inicia automáticamente.

Los pods generados, se distribuyen de forma aleatoria entre los diferentes Worker Nodes que tengan recursos suficientes para realizar los despliegues.

En la siguiente imagen (figura 6.3) se muestran los Endpoints escogidos para los servicios que expondremos mediante la creación de los hosts en el Ingress. Estos EndPoints, están implementados con el protocolo https, por lo que la comunicación será segura.

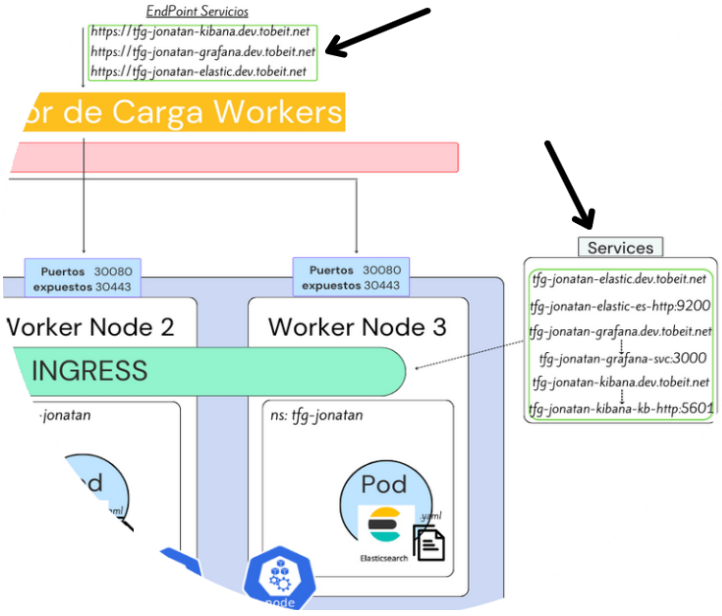

Fig. 6.3 Endpoints expuestos junto a sus servicios asociados

A su vez, en la parte derecha de la imagen, vemos la asociación entre hosts y servicios que gestiona nuestro ingress controller y que a su vez se almacenan en un pod del ingress controller.

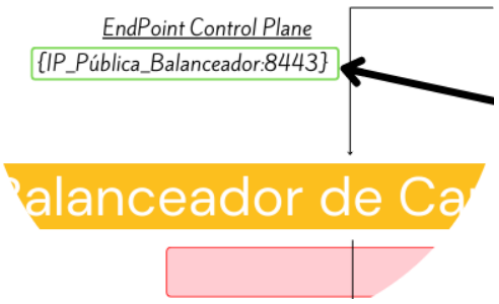

Fig. 6.4 EndPoint de acceso a la API Server desde el exterior

Por último, en la figura 6.4, observamos el EndPoint por el cual accederemos a la infraestructura desde el exterior.

Para comprobar que los servicios se han desplegado con éxito, realizamos las siguientes comprobaciones desde kubectl, el CLI de Kubernetes:

```
PS C:\Users\jona_\Desktop> kubectl get pod -n tfg-jonatan
NAME                                  READY   STATUS    RESTARTS
tfg-jonatan-elastic-es-mdn-0          1/1     Running   0
tfg-jonatan-elastic-es-mdn-1          1/1     Running   0
tfg-jonatan-elastic-es-mdn-2          1/1     Running   0
tfg-jonatan-grafana-6757dc5dd8-lhm4h  1/1     Running   0
tfg-jonatan-kibana-kb-5c5f6d448c-tdstq  1/1     Running   0
```

Fig. 6.5 Comprobación de los pods desplegados de cada tecnología

```
PS C:\Users\jona_\Desktop> kubectl get svc -n tfg-jonatan
NAME                                TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
tfg-jonatan-elastic-es-http         ClusterIP   10.105.1.76      <none>        9200/TCP
tfg-jonatan-elastic-es-internal-http  ClusterIP   10.99.46.200     <none>        9200/TCP
tfg-jonatan-elastic-es-mdn          ClusterIP   None             <none>        9200/TCP
tfg-jonatan-elastic-es-transport    ClusterIP   None             <none>        9300/TCP
tfg-jonatan-grafana-svc             ClusterIP   10.106.159.77    <none>        3000/TCP
tfg-jonatan-kibana-kb-http          ClusterIP   10.108.132.119   <none>        5601/TCP
```

Fig. 6.6 Comprobación de los services creados de cada tecnología

```
PS C:\Users\jona_\Desktop> kubectl get ingress -n tfg-jonatan
NAME                          CLASS    HOSTS                                ADDRESS                       PORTS
tfg-jonatan-elastic-ingress   <none>   tfg-jonatan-elastic.dev.tobeit.net   192.168.6.10,192.168.6.3   80, 443
tfg-jonatan-grafana-ingress   <none>   tfg-jonatan-grafana.dev.tobeit.net   192.168.6.10,192.168.6.3   80, 443
tfg-jonatan-kibana-ingress    <none>   tfg-jonatan-kibana.dev.tobeit.net    192.168.6.10,192.168.6.3   80, 443
```

Fig. 6.7 Comprobación de las ingress generadas de cada tecnología

```
PS C:\Users\jona_\Desktop> kubectl get pvc -n tfg-jonatan
NAME                                          STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS
elasticsearch-data-tfg-jonatan-elastic-es-mdn-0   Bound    pvc-29a8eccb-b112-4dc1-8052-66c8c6463ce3   1Gi        RWO            openebs-lvmpv
elasticsearch-data-tfg-jonatan-elastic-es-mdn-1   Bound    pvc-0f2121ae-1fdb-4d51-bd13-a7def93c8b72   1Gi        RWO            openebs-lvmpv
elasticsearch-data-tfg-jonatan-elastic-es-mdn-2   Bound    pvc-af2377b9-3151-4940-a41c-e920fa06ba36   1Gi        RWO            openebs-lvmpv
tfg-jonatan-grafana-pvc                       Bound    pvc-93e65f2f-d2bd-4efb-9137-88aa541e59fe   1Gi        RWO            cstor-csi-disk
```

Fig. 6.8 Comprobación de los PVCs asignados para cada tecnología

Una vez comprobado su buen funcionamiento dentro de la infraestructura, accedemos a cada uno de esos tres servicios expuestos mediante un navegador para verificar que los despliegues se han realizado con éxito:

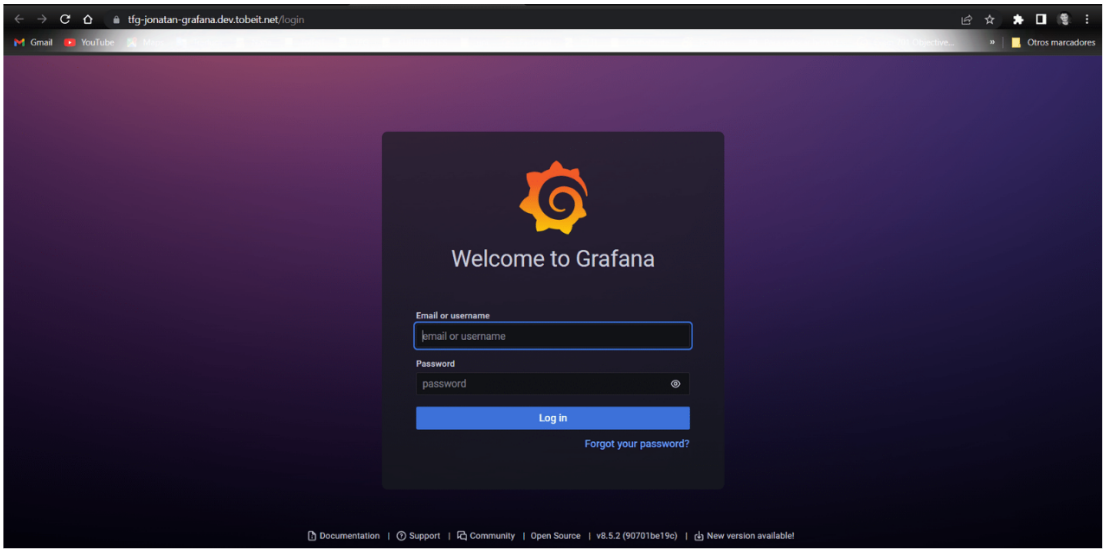

Fig. 6.9 Página de Login de Grafana

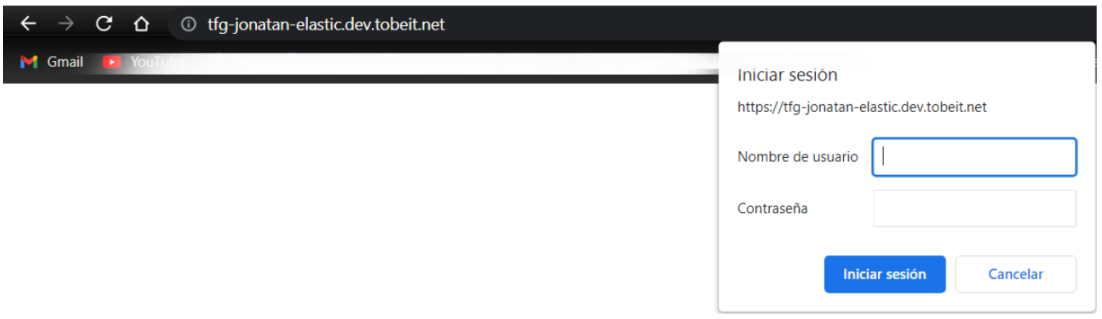

Fig. 6.10 Página de Login de Elastic

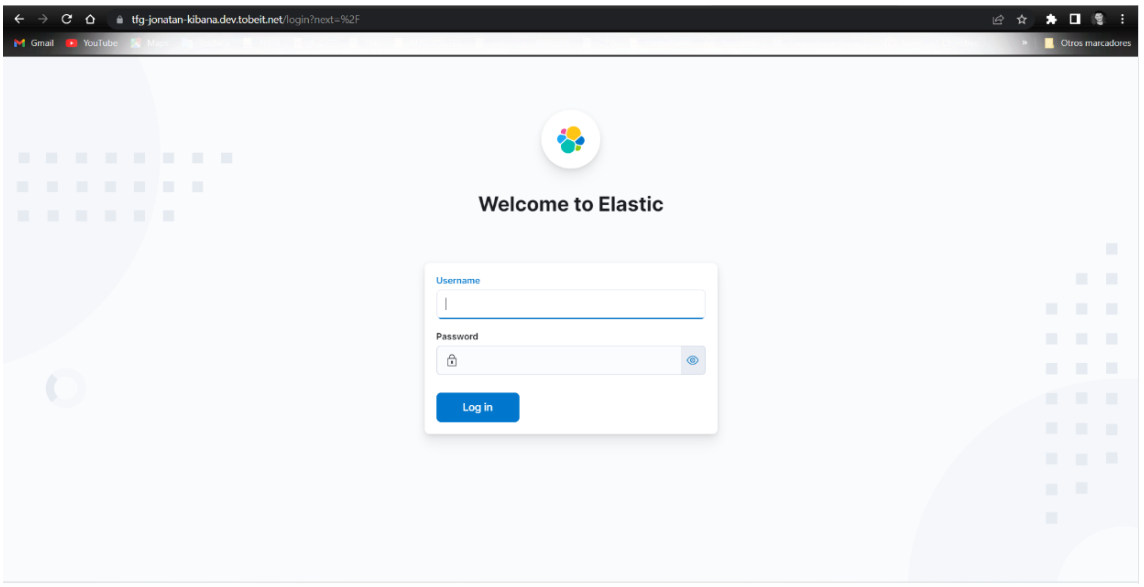

Fig. 6.11 Página de Login de Kibana

Como se ha comprobado, el despliegue de las tecnologías utilizadas para la monitorización se ha realizado con éxito. A continuación en el siguiente capítulo, visualizaremos los ficheros de configuración de las diferentes tecnologías de monitorización.

7. Ficheros de configuración de las diferentes tecnologías de monitorización

Visto el despliegue de las diferentes tecnologías, en este capítulo se explicará con profundidad los ficheros de configuración utilizados.

7.1. Ficheros de configuración de Elastic, Kibana y Grafana

7.1.1. Ficheros de configuración de Elastic

Para el despliegue de esta tecnología, se han utilizado diferentes ficheros. Los cuales son: operator.yaml, crds.yaml, elastic.yaml, ingress.yaml.

Respecto a los ficheros operator.yaml y crds.yaml, son ficheros que nos proporciona Elastic. Y se utilizan para que Kubernetes pueda reconocer el despliegue del fichero elastic.yaml. Estos ficheros, como son proporcionados por Elastic [31], no se ha realizado ninguna modificación en ellos.

```yaml
apiVersion: elasticsearch.k8s.elastic.co/v1
kind: Elasticsearch
metadata:
  name: tfg-jonatan-elastic
  namespace: tfg-jonatan
spec:
  version: 8.3.2
  http:
    tls:
      selfSignedCertificate:
        disabled: true
  nodeSets:
    - name: mdn
      count: 3
      config:
        node.roles: ["master","data_hot", "data_content", "ingest"]
```

Fig. 7.1 Fichero de configuración elastic.yaml (1/2)

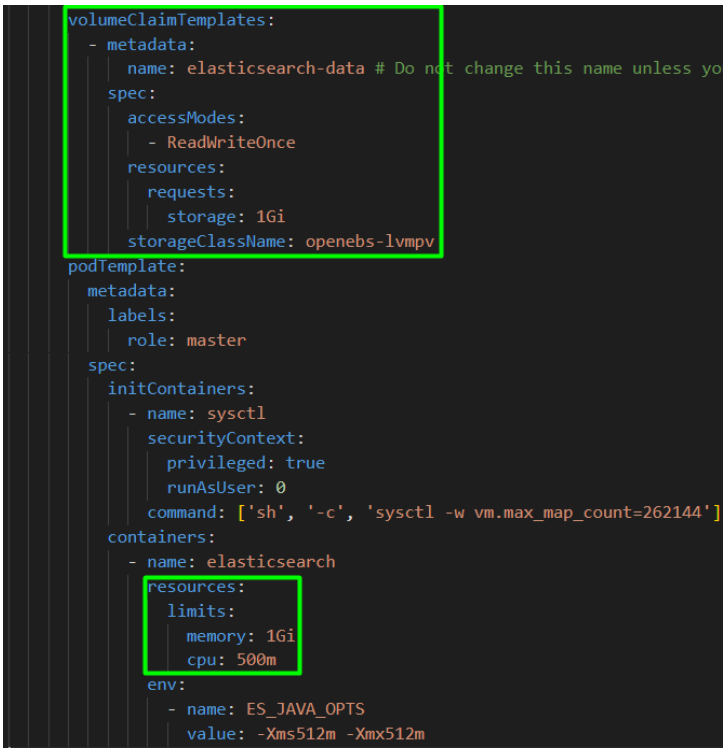

Fig. 7.2 Fichero de configuración elastic.yaml (2/2)

Las figuras 7.1 y 7.2, pertenecen al fichero de configuración elastic.yaml. En la figura 7.1 observamos que el tipo de despliegue es de tipo Elastic. Por lo que este componente, aunque por defecto no pertenece a Kubernetes, gracias al fichero crd.yaml y al fichero del operador de elastic, Kubernetes lo reconocerá y podrá efectuar el despliegue con éxito.

Si seguimos viendo la imagen, la parte marcada con un rectángulo verde, contiene el campo count, donde se indica la cantidad de pods de elastic que se van a desplegar, y el campo node.roles que nos indica el tipo de rol que tendrán esos nodos en concreto. En este caso, los tres nodos podrán hacer de Master Node, por lo que se realizará un diálogo entre los diferentes nodos para decidir cuál de ellos hará de Master Node.

Estos pods permitirán la ingesta de datos en su interior (ingest) y a su vez, almacenarán los datos más recientes (data_hot).

El volumen reclamado por parte de cada pod, será accesible únicamente por el mismo pod que esté asociado a ese volumen. Esto se puede observar mediante el campo accesModes (figura 7.2).

Además, se ha configurado para cada pod una solicitud de almacenamiento de tipo openebs-lvmpv de 1Gi. Por lo que cada pod que se despliegue obtendrá 1Gi de almacenamiento para sus índices.

Si observamos el segundo rectángulo de color verde de la figura 7.2, vemos los recursos solicitados para cada contenedor de elastic. En este caso se ha solicitado un límite de recursos de 1GB RAM y 500 mil cores de CPU.

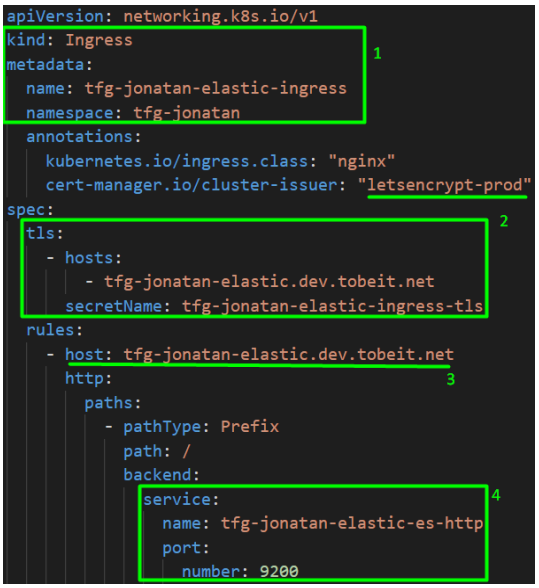

Fig. 7.3 Fichero de configuración ingress.yaml, Elastic

De este fichero (figura 7.3), se explicarán los campos marcados en verde.

En el primer apartado se está configurando el namespace donde se desplegará este componente y el tipo de componente que se desplegará, en este caso una Ingress.

En el segundo apartado se configura el certificado que se utilizará para el host definido dentro de la configuración tls. Esta configuración, es la que nos proporciona que los navegadores confíen en el servicio que se expondrá de elastic y que la comunicación sea segura a través de la red.

En el tercer apartado se define el host utilizado para acceder al servicio interno.

Por último, en el cuarto apartado se define el service y el puerto utilizado por elastic (puerto por defecto 9200) al que el ingress controller redirigirá la petición en el momento que llegue una petición dirigida al host configurado en este fichero.

7.1.2. Fichero de configuración de Kibana

Respecto a la configuración de Kibana, también haremos uso de dos ficheros de configuración: kibana.yaml y ingress.yaml.

A continuación se muestran sus configuraciones:

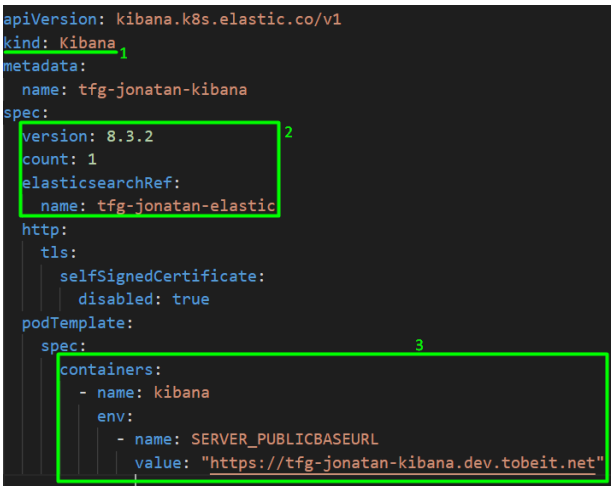

Fig. 7.4 Fichero de configuración kibana.yaml

En la figura 7.4 se muestra la configuración del fichero kibana.yaml.

Si visualizamos el primer rectángulo marcado en verde, vemos el tipo de componente que se está creando, en este caso Kibana.

Si observamos el segundo rectángulo, vemos la cantidad de pods de este tipo que serán desplegados, además de el campo elasticsearchRef donde configuramos el nombre del service de elastic al que estará asociado Kibana.

En el último rectángulo verde observamos la URL utilizada para acceder al servicio de Kibana.

Internamente, el controlador de Elastic, se encarga de hacer las configuraciones necesarias para que Kibana pueda ser accesible desde fuera y a su vez, pueda asociarse a la infraestructura de Elastic.

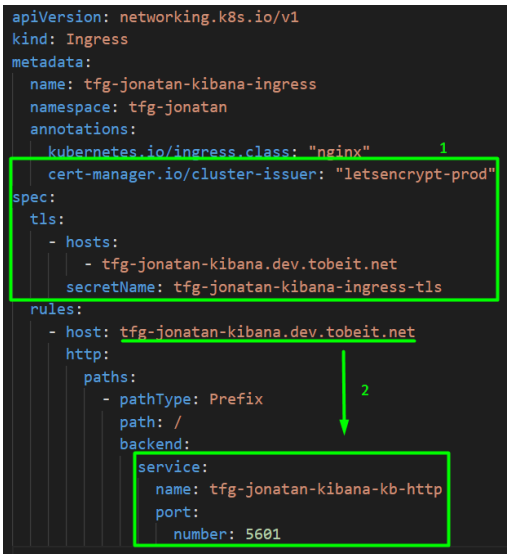

Fig. 7.5 Fichero de configuración ingress.yaml, Kibana

En la figura 7.5, visualizamos el fichero de configuración utilizado para crear el host que será utilizado para acceder desde el exterior al servicio de Kibana. Respecto a este fichero, la configuración es similar a la realizada en el fichero de configuración de ingress.yaml de Elastic.

En el primer rectángulo vemos la configuración de conexión segura configurada para el acceso de este servicio. Mientras que en el segundo rectángulo vemos la asociación del host con el service y el puerto por defecto utilizado para acceder a Kibana.

7.1.3. Fichero de configuración de Grafana

Respecto a los ficheros utilizados para el despliegue de Grafana, solo se ha utilizado grafana.yaml.

A continuación se muestra la configuración realizada para esta tecnología:

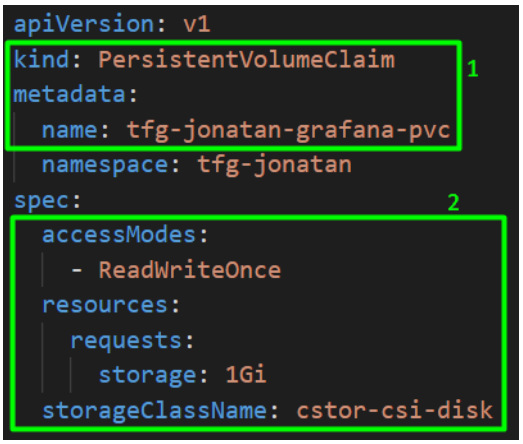

Fig. 7.6 Fichero de configuración grafana.yaml (1/4)

En la figura 7.6 se puede visualizar la configuración del PVC que se utilizará para solicitar un PV mediante el storageClass especificado, que contenga 1Gi de almacenamiento.

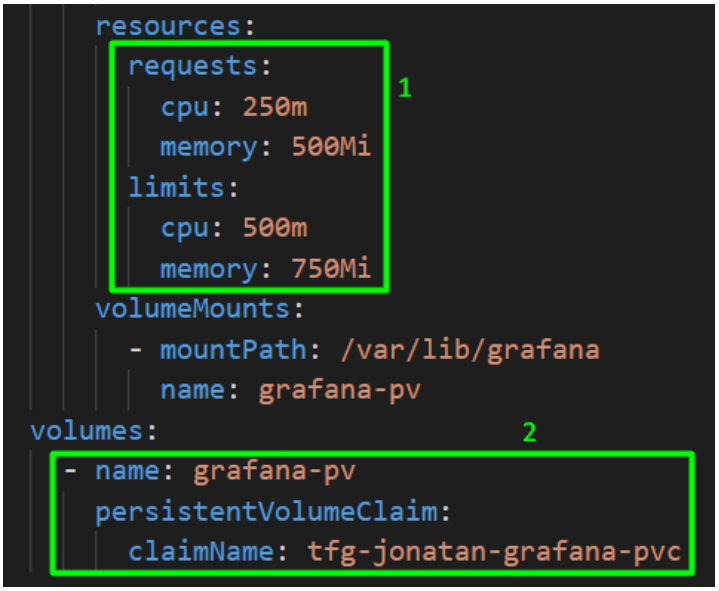

Fig. 7.7 Fichero de configuración grafana.yaml (2/4)

En la figura 7.7, en el primer rectángulo verde, puede ver la configuración de los recursos requeridos para el contenedor de Grafana que se generará.

Mientras que en el segundo rectángulo, se observa la asociación entre el pod de Grafana y el PVC visto en la figura 7.6.

Esta asociación se utiliza para seguidamente, como se ve en la imagen, poder utilizar ese volumen dentro del contenedor donde disponemos de Grafana. De esta manera, el contenedor Grafana podrá almacenar los datos en el

mountPath configurado y a su vez, obtener persistencia de datos. Por lo que si el pod de Grafana muere, los datos permanecen intactos dentro del PVC seleccionado.

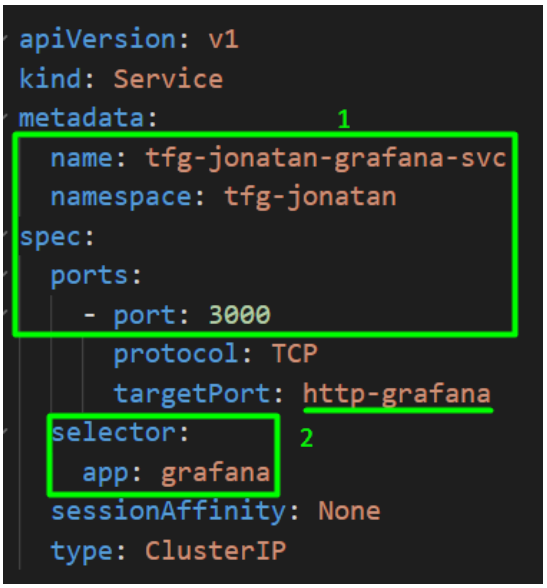

Fig. 7.8 Fichero de configuración grafana.yaml (3/4)

En el primer rectángulo marcado de color verde de la figura 7.8, podemos observar el nombre del servicio, el namespace donde se generará este servicio y el puerto de acceso configurado para ese servicio en concreto.

Mientras que en el segundo rectángulo, observamos el campo targetPort, que nos índica el puerto al que se redirigirá la petición cuando se haga una solicitud de servicio a Grafana.

A su vez, en este rectángulo, se observa el campo selector donde se establecen etiquetas. Estas etiquetas se definen cuando se crean los pods de Grafana y se utilizan para hacer conexiones entre Services y pods. En este caso la etiqueta es *app: grafana*. Por lo que todos los pods que tengan esta etiqueta, recibirán todas las peticiones reenviadas por el service de Grafana.

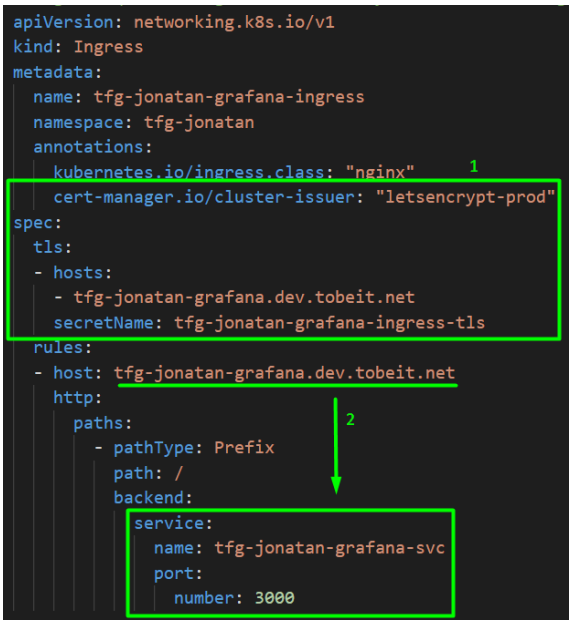

Fig. 7.9 Fichero de configuración grafana.yaml (4/4)

Por último, en la figura 7.9, visualizamos la ingress generada para acceder a este servicio desde el exterior.

La configuración es muy similar a la realizada para Elastic y Kibana.
Con la diferencia del nombre del host utilizado y la asociación del servicio proporcionado por Grafana junto al host generado.

El puerto configurado para acceder al servicio en este caso es el puerto 3000, puerto por defecto utilizado para esta tecnología.

Una vez vistas las partes de las que se componen los ficheros de configuración utilizados para realizar el despliegue del sistema de monitorización completo, daremos paso al capítulo siguiente donde se realizarán las pruebas de evaluación.

8. Pruebas de evaluación

Dado que ya tengo creadas la infraestructura de Kubernetes y del sistema de monitorización completo, vamos a utilizar Metricbeat como operación para la obtención de métricas del sistema operativo del ordenador portátil de la empresa ToBeIT para posteriormente, ver el rendimiento de los dispositivos internos de la empresa.

8.1 Verificación de los despliegues mediante Metricbeat

Se procede a instalar Metricbeat [32]: agente que nos proporciona el Stack de Elastic y que utilizaremos para monitorizar servicios, métricas de recursos de un ordenador, entre otras cosas.

La instalación se realizará en un ordenador portátil de empresa. Con sistema operativo Windows 11 y modelo de portátil ASUS F515J.

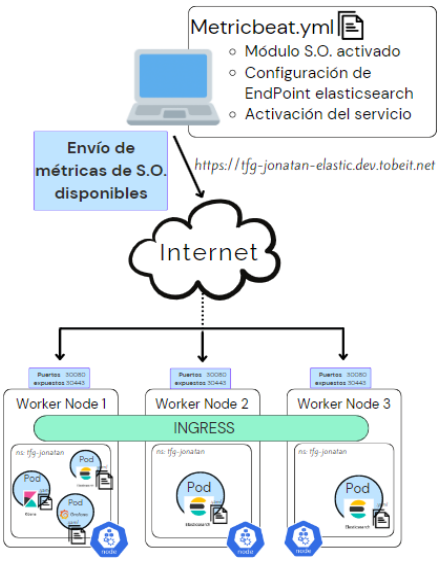

Fig. 8.1 Envío de datos desde Agente Metricbeat hacía Elastic

Como se observa en la figura anterior (figura 8.1), nuestro dispositivo enviará datos hacia el EndPoint de elastic configurado previamente a través del agente instalado de Metricbeat. Además, podemos observar los puntos configurados dentro de Metricbeat para poder realizar esta ingesta de datos. Donde procedemos a activar el módulo de sistema, configurar la ruta de salida de los datos generados por Metricbeat, y a activar el propio servicio de Metricbeat para la puesta en marcha de este Agente. Los datos que se reciban de Metricbeat, serán almacenados y visualizados en Kibana.

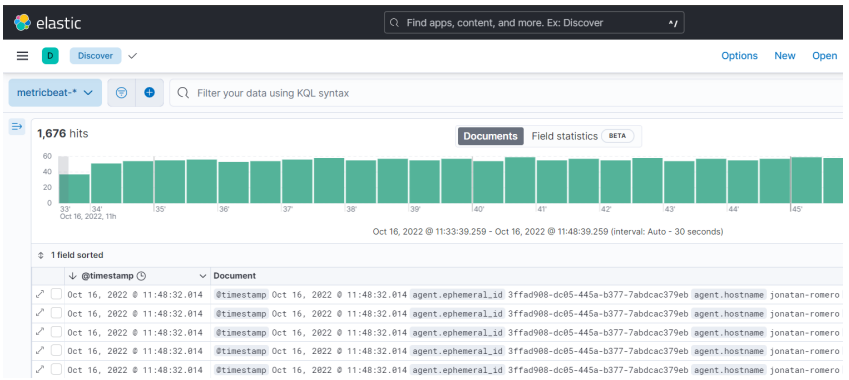

Fig. 8.2 Visualización de datos en Kibana ingestados por Metricbeat

Una vez instalado y arrancado el servicio de Metricbeat, procedemos a visualizar datos en Kibana (figura 8.2).

Al llegar datos desde Metricbeat a la infraestructura de Elastic, aparte de corroborar que el despliegue de la infraestructura de Elastic está bien realizado, a su vez se corrobora que el despliegue de Kubernetes también ha sido un éxito. Ya que el despliegue de Elastic se realiza dentro de la propia infraestructura de Kubernetes.

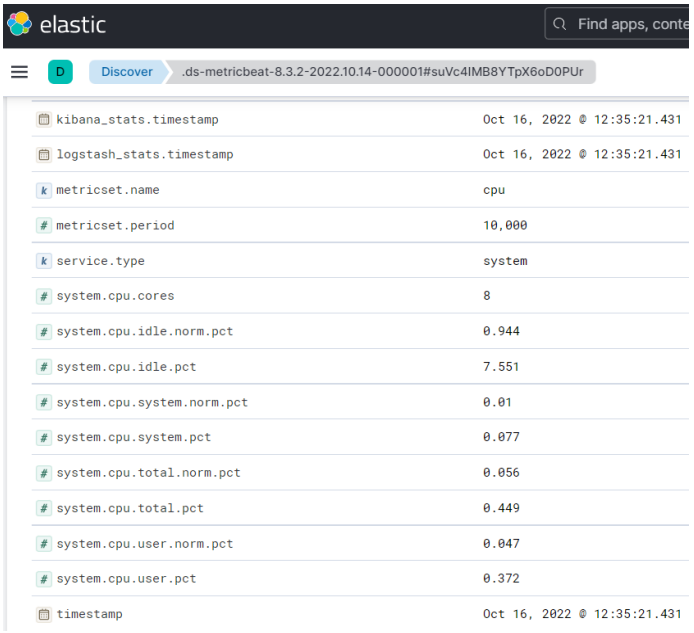

Fig. 8.3 Obtención de campos predefinidos de S.O. por Metricbeat

Como Metricbeat nos proporciona una plantilla de módulo de Sistema Operativo para Windows de manera integrada, podemos visualizar los campos predefinidos por defecto para Windows junto a los valores correspondientes de cada campo sin necesidad de procesar los datos para su futura visualización (figura 8.3) en Grafana.

Ahora es el momento de comprobar y visualizar los datos en Grafana:

Fig. 8.4 Visualización de métricas de Sistema Operativo mediante Grafana

Como se observa en la figura 8.4, el despliegue de Grafana también ha sido un éxito. En esta imagen se puede observar algunas de las métricas del portátil que se está monitorizando. Donde se encuentran métricas como: tiempo de actividad de uso del portátil, número de cores de los que dispone, memoria RAM disponible, uso de CPU y de Memoria RAM…

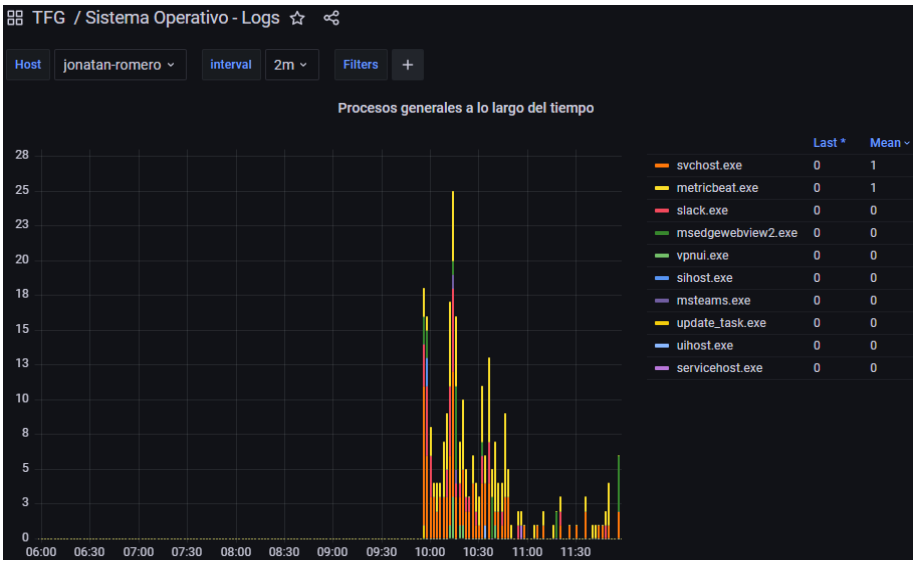

Fig. 8.5 Visualización de registros de Sistema Operativo mediante Grafana

En la figura 8.5, en cambio, se puede visualizar los registros de los procesos que se han ido ejecutando a lo largo del tiempo junto al nombre del proceso en específico.

8.2 Validación de servicios mediante CLI de Kubernetes

Para saber si Metricbeat se ha configurado adecuadamente, se ha realizado la siguiente prueba (figura 8.6):

```
PS C:\Program Files\Metricbeat> .\metricbeat.exe test output
elasticsearch: https://tfg-jonatan-elastic.dev.tobeit.net:443...
  parse url... OK
  connection...
    parse host... OK
    dns lookup... OK
    addresses: 82.223.6.122
    dial up... OK
  TLS...
    security: server's certificate chain verification is enabled
    handshake... OK
    TLS version: TLSv1.3
    dial up... OK
  talk to server... OK
  version: 8.3.2
```

Fig. 8.6 Comprobación de configuración de Metricbeat

8.3 Ficheros de configuración de Metricbeat

En este subapartado se explicará la configuración realizada en el dispositivo portátil para poder obtener métricas del sistema operativo, y enviarlas a Elastic desde el exterior.

```
- module: system
  period: 10s
  metricsets:
    - cpu
    - memory
    - network
    - process
    - process_summary
    - socket_summary
  process.include_top_n:
    by_cpu: 5      # include top 5 processes by CPU
    by_memory: 5   # include top 5 processes by memory

- module: system
  period: 1m
  metricsets:
    - filesystem
    - fsstat
  processors:
  - drop_event.when.regexp:
      system.filesystem.mount_point: '^/(sys|cgroup|proc|dev|etc|host|lib|snap)($|/)'

- module: system
  period: 15m
  metricsets:
    - uptime
```

Fig. 8.7 Fichero de configuración system.yml

En la figura 8.7, se puede observar la configuración realizada en el módulo de system para poder obtener métricas de nuestro sistema operativo.

En el primer rectangulo marcado de color verde vemos la perioricidad en la que se obtendran estas metricas junto las métricas que se obtendran.

En el segundo y tercer rectángulo visualizamos algo similar pero con la diferencia del periodo en el que se recogen estas métricas.

Cada una de las métricas nos aportarán información de la CPU, RAM, disco, procesos internos que realiza nuestro sistema operativo, cuánto tiempo lleva activo nuestro dispositivo desde que se ha empezado a monitorizar, métricas de red, sockets entre otros.

Este fichero será leído automáticamente por metricbeat en el momento que se inicie el servicio en el ordenador.

```
# ---------------------------- Elasticsearch Output ----------------------------
output.elasticsearch:
  hosts: ["https://tfg-jonatan-elastic.dev.tobeit.net:443"]
  username: "xxxxxxxxxxxxxxxxxxx"
  password: "xxxxxxxxxxxxxxxxxxxx"
```

Fig. 8.8 Fichero de configuración metricbeat.yml

Respecto a la configuración del fichero Metricbeat.yml, únicamente se ha tenido que configurar el punto de salida donde se enviarán las métricas obtenidas de nuestro dispositivo (figura 8.8).

Finalmente, ha sido gratificante, tal y como hemos podido comprobar, que la prueba de evaluación y por consiguiente la configuración de Metricbeat ha sido satisfactoria, realizándose con el objetivo de obtener las métricas del sistema operativo para poder ver el rendimiento de los dispositivos internos de la empresa.

El capítulo 9 que se presenta a continuación resume las conclusiones técnicas y personales que abarcan todo el proyecto.

9. Conclusiones

Una vez realizadas las pruebas de evaluación, en este capítulo se explicarán las conclusiones a las que se ha llegado respecto a niveles técnicos, personales y de mejoría.

9.1. Conclusiones técnicas

En cuanto a las conclusiones, podemos afirmar que tanto el despliegue de la infraestructura de Kubernetes como el despliegue de sistema de monitorización completo, han resultado efectivos y satisfactorios.

Cabe destacar que los requisitos especificados por la empresa han sido más que un inconveniente, una favorable ayuda para poder realizar las pruebas y los despliegues con éxito.

Aunque por parte de la empresa se haya puesto como requisito realizar el despliegue mediante Kubernetes y utilizando el cloud de Ionos, esto me ha hecho ver una visión real de lo que pueden llegar a solicitar o proporcionar empresas para la investigación.

La combinación de Elastic y Grafana para realizar el despliegue de monitorización completo ha tenido mucha sinergía. Se ha podido realizar el despliegue juntando estas dos tecnologías de manera sencilla.

Finalmente y para concluir este apartado los objetivos específicos planteados al inicio del proyecto, se han optimizado y llevado a cabo con éxito:

- Se han validado las diferentes tecnologías de monitorización escogidas por la empresa.

- Se ha diseñado la infraestructura con Kubernetes de contenedores por un clúster proporcionado por la empresa en Ionos.

- Se han desarrollado los diferentes ficheros de configuración de las tecnologías de monitorización escogidas por la empresa.

- Se han realizado pruebas a través de servicios para obtener métricas del sistema operativo y posteriormente poder analizar el rendimiento de los servidores.

9.1.1. Incidencias

Uno de los inconvenientes más importantes que he encontrado durante el transcurso de este proyecto ha sido el tiempo. El objetivo era finalizar el proyecto a finales de junio pero se ha extendido hasta principios de octubre.

La búsqueda de información y entendimiento de las tecnologías ha requerido un coste de tiempo adicional que no estaba previsto.

También he tenido bastantes incidencias a la hora de configurar la infraestructura de Kubernetes.

Estas incidencias se nombran a continuación:

- Conectividad Pod-to-Pod. Se ha tenido que reinstalar la infraestructura múltiples veces para el correcto funcionamiento de la comunicación entre los pods, CNI (Container Network Interface). La causa de la mala conectividad ha sido el orden de instalación de los componentes de la infraestructura.

- Problemas en los worker nodes con los discos incorporados y con las particiones de los mismos.

- No se conseguía obtener visibilidad de los servicios expuestos fuera de la infraestructura. Esto era debido a la mala configuración de los selectores a los que apuntaba el servicio en los archivos de configuración .yaml.

9.2. Conclusiones personales

Desde mi punto de vista, Kubernetes es una gran herramienta. Y se está utilizando cada vez más en esta industria. Por lo que es una tecnología que si bien no se conoce, es necesaria conocerla para no quedarse obsoleto ni atrás en el avance del campo de las tecnologías de la información.

Respecto a Elastic, es una tecnología muy potente y amplia de la que sin duda con un año de experiencia o siete años, no llegas a conocer realmente todo lo que nos ofrece esta gran tecnología.
Es una de las mejores tecnologías que he visto para la realización de búsquedas de datos.

La curva de aprendizaje ha sido muy dura, ya que Kubernetes es totalmente nuevo para mi de la misma manera que realizar todas las configuraciones y procesos necesarios para que la infraestructura de Kubernetes se pueda crear con éxito.

Si que es verdad que en asignaturas como EA o IOT se han explicado conceptos y realizado trabajos que me han hecho entender más rápido conceptos que para mi son totalmente nuevos.

Ha sido un poco complicado realizar el TFG con empresa ya que independientemente de tener tiempo para dedicar al TFG, he tenido que estar en constante aprendizaje de todas las tecnologías que usa la empresa para poder realizar tareas solicitadas por su parte.

9.3. Futuras mejoras

Hay diferentes mejoras que se podrían realizar en el proyecto:

- Automatización de despliegues de las tecnologías utilizadas mediante Jenkins.

- La seguridad de Kubernetes mediante configuración de roles dentro de la infraestructura.

- Asociación de despliegue de pods con nodos. Esto permitiría que ciertos pods se pudiesen desplegar únicamente en ‘x’ nodos de la infraestructura.

- Monitorizar el estado de la infraestructura de Kubernetes.

- Monitorizar los recursos de los nodos disponibles de la infraestructura.

- Hacer uso de Docker Hub u otras tecnologías que nos permitan descargar y crear imágenes necesarias para la creación de nuestros contenedores dentro de Kubernetes.

10. Referencias

[1] ¿Qué es el cloud? Introducción al cloud computing - IONOS.

URL: https://www.ionos.es/digitalguide/servidores/know-how/que-es-el-cloud/

[2] What is a Container? | Docker.

URL: https://www.ionos.es/digitalguide/servidores/know-how/que-es-el-cloud/

[3] ¿Qué es Docker?

URL: https://www.redhat.com/es/topics/containers/what-is-docker

[4] ¿Qué es Kubernetes? | Kubernetes.

URL: https://kubernetes.io/es/docs/concepts/overview/what-is-kubernetes/

[5] ¿What is a Docker Swarm? | Docker Swarm.

URL: https://www.sumologic.com/glossary/docker-swarm/

[6] ¿Qué es Elasticsearch? | Elastic.

URL: https://www.elastic.co/es/what-is/elasticsearch

[7] Grafana - Wikipedia, la enciclopedia libre.

URL: https://es.wikipedia.org/wiki/Grafana

[8] Prometheus - Monitoring system time series database.

URL: https://prometheus.io/

[9] Dynatrace: Software Intelligence for the Cloud | Dynatrace.

URL: https://www.dynatrace.com/company/

[10]. Evernote - Wikipedia, la enciclopedia libre.

URL: https://es.wikipedia.org/wiki/Evernote

[11]. Todoist. URL: https://todoist.com/home

[12]. Focus To-Do. URL: https://www.focustodo.cn/

[13] Git. URL: https://git-scm.com/

[14] GitHub - Wikipedia, la enciclopedia libre.

URL: https://es.wikipedia.org/wiki/GitHub

[15] Visual Studio Code - Code Editing. Redefined.

URL: https://code.visualstudio.com/

[16] Desarrollo en cascada - Wikipedia, la enciclopedia libre.

URL: https://es.wikipedia.org/wiki/Desarrollo_en_cascada

[17] Prerrequisitos de infraestructura.

URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/

[18] Documentación de Kubernetes. URL: https://kubernetes.io/es/docs/home/

[19] Install Calico networking.

URL: https://projectcalico.docs.tigera.io/getting-started/kubernetes/self-managed-onprem/onpremises

[20] Ingress. URL: https://kubernetes.io/docs/concepts/services-networking/ingress/

[21] Lista de Ingress Controller.

URL: https://kubernetes.io/docs/concepts/services-networking/ingress-controllers/

[22] Installing Helm. URL: https://helm.sh/docs/intro/install/

[23] NGINX Ingress Controller. URL: https://kubernetes.github.io/ingress-nginx/

[24] No-IP - Wikipedia, la enciclopedia libre. URL:https://en.wikipedia.org/wiki/No-IP

[25] Let's Encrypt. URL: https://letsencrypt.org/es/

[26] OpenEBS. URL: https://openebs.io/docs

[27] Installing OpenEBS. URL: https://openebs.io/docs/user-guides/installation

[28] Local or Distributed Persistent Volumes, OpenEBS.

URL: https://openebs.io/docs/#what-does-openebs-do

[29] LVM-Localpv, OpenEBS. URL: https://github.com/openebs/lvm-localpv

[30] cStor, OpenEBS. URL: https://openebs.io/docs/concepts/cstor

[31] Elastic Cloud on Kubernetes.

URL: https://www.elastic.co/guide/en/beats/metricbeat/8.3/metricbeat-installation-configuration.html

[32] Metricbeat quick start: installation and configuration.

URL: https://www.elastic.co/guide/en/beats/metricbeat/8.3/metricbeat-installation-configuration.html