UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Gestión centralizada de Logs en clústeres Kubernetes

Trabajo Fin de Máster

Máster Universitario en Ingeniería Informática

AUTOR/A: Farooq Nargis, Tahir

Tutor/a: Acebrón Linuesa, Floreal

CURSO ACADÉMICO: 2021/2022

Agradecimientos

En primer lugar, agradecer a mi tutor Floreal Acebrón Linuesa, que me ha acompañado durante todo el cuatrimestre de la asignatura de Configuración y Optimización de Sistemas de Cómputo y ayudar a hacer realidad este desarrollo.

También me gustaría agradecer a mi familia por ser una base fundamental de mi día a día y por confiar en mí para alcanzar los objetivos personales y profesionales.

Finalmente, me gustaría agradecer a compañeros y profesores de los que he aprendido estos años. Agradecer por la oportunidad de realizar este desarrollo y permitir profundizar los estudios en el campo que me gusta.

Resumen

Hoy en día, la gestión y despliegue de las aplicaciones basadas en microservicios se hace de forma óptima y automática mediante un orquestador de contenedores: Kubernetes (utilizado en este desarrollo). Se encarga de desplegar los contenedores, además del Service Discovery y facilitar el escalamiento de la aplicación entre otros aspectos. Además Kubernetes se puede integrar con herramientas de CI/CD.

Con la aparición de las aplicaciones ejecutando dentro de los contenedores, es indispensable observar logs para verificar el correcto funcionamiento de distintos componentes: sistemas, objetos de Kubernetes, contenedores y almacenar todos sus logs en un disco persistente y poder filtrar por nombre de espacio, fecha o nombre del objeto más tarde.

Para ello, en primer lugar, se ha montado un clúster de Kubernetes en el entorno local utilizando máquinas virtuales. A continuación, se han analizado las posibilidades de obtención de logs de los contenedores. Existen distintas herramientas que permiten analizar los logs, sin embargo, cuando el contenedor se detiene, sus logs desaparecen. La conservación de dichos logs es importante para llegar al origen del problema.

Tras analizar carencias de los comandos y funcionalidades que ofrece Kubernetes de forma nativa, se ha procedido a instalar las herramientas Elasticsearch, Logstash y Kibana sobre el clúster.

Tras solucionar la persistencia de logs en el despliegue local, se ha desplegado la infraestructura de Kubernetes en la nube de distintas formas:

- Instalación manual en máquinas virtuales hospedadas en Google Compute Cloud.
- Uso del clúster ya configurado en Google Kubernetes Engine.

En ambas instalaciones se han analizado funcionalidades de visualización de logs. Al no estar satisfecha la persistencia de logs con la instalación en máquinas virtuales, se ha instalado la pila Elasticsearch, Logstash y Kibana.

Finalmente, sobre los sistemas desplegados, se han realizado pruebas con aplicaciones emitiendo logs para comprobar el correcto funcionamiento de la solución.

Palabras clave: Clústeres, Kubernetes, logging, Elasticsearch, Logstash, Kibana, ELK, Google Cloud, gestión centralizada de logs.

Abstract

Nowadays, the management and deployment of applications based on microservices is performed optimally and automatically through a container orchestrator: Kubernetes (used in this development). Kubernetes is responsible for deploying containers, as well as Service Discovery and facilitating the scaling of the application, among other aspects. Kubernetes can also be integrated with CI/CD tools.

With the emergence of applications running inside containers, it is essentials to observe logs to verify the correct operation of different components: system, Kubernetes objects, containers and to store all their logs in a persistent disk and then be able to filter by namespace, date, or object name later.

To do this, first, a Kubernetes cluster was set up in the local environment using virtual machines. Next, we analysed the possibilities of obtaining logs from the containers. There are several tools available to analyse the logs, however, when the container is stopped, its logs disappear. The preservation of these logs is important to get to the source of the problem.

After analysing the shortcomings of the commands and functionalities offered natively by Kubernetes, we proceeded to install the Elasticsearch, Logstash and Kibana tools on the cluster.

After solving the log persistence in the local deployment, the Kubernetes infrastructure has been deployed in the cloud in different ways:

- Manual installation on virtual machines hosted on Google Compute Engine.
- Using the cluster already configured in Google Kubernetes Engine.

In both installations, log visualization functionalities have been analysed. As the log persistence was not satisfied with the installation on virtual machines, the Elasticsearch, Logstash and Kibana stack were installed.

Finally, on the deployed systems, tests have been performed with applications emitting logs to verify the correct operation of the solution.

Keywords: Clusters, Kubernetes, Logging, Elasticsearch, Logstash, Kibana, ELK, Google Cloud, Centralized log management.

Índice de contenidos

Índice de imágenes

Índice de tablas

1. Introducción y estructura

Actualmente, las aplicaciones se despliegan en microservicios, basado en contenedores. Estos son orquestados por Kubernetes.

Para saber de antemano que ocurre en los servicios, pods o despliegues realizados, es necesario un control de logs. Ya que permite detectar cuellos de botella en aplicaciones, saturación de pods, control de errores o sobrecarga del sistema.

En un orquestador como Kubernetes, la observabilidad sirve para monitorizar y asegurar un sistema distribuido permitiendo analizar el contexto de un servicio para:

- Visualizar el clúster como un gráfico de servicios para ver dependencias entre pods asociados a servicios y flujos de comunicación entre los servicios.
- Analizar metadatos como cantidad de CPU, memoria consumidos por pod en el nodo desplegado.
- Observar métricas relacionadas con operaciones sobre pods como latencia de red, carga de tráfico de peticiones HTTP o tiempos de reinicio de pods.
- Analizar comunicación de red para pods y recursos asociados.
- Analizar logs de pods y recursos asociados a un determinado servicio.

A continuación, se detallan aspectos en los cuales más impacto tiene la observabilidad de logs.

Tráfico de red

Mediante la monitorización de logs se puede controlar la comunicación entre los pods, es decir, los pods etiquetados como “frontend” se comunican con el microservicio ‘finanzas’ a través del puerto 8080, cualquier otro intento de comunicación se puede considerar un ataque. Es útil ya que permite detectar actividades maliciosas o la detección de tráfico procedente de ubicaciones geográficas inesperadas o no autorizadas.

DNS Logs

Es recomendable revisar logs relacionado con la actividad DNS para detectar peticiones con dominios no conocidos o maliciosos.

Tráfico de aplicaciones

Es recomendable revisar flujos de tráfico de aplicaciones ejecutándose en el clúster en busca de llamadas HTTP con cabeceras maliciosas o desconocidas por la aplicación, formato incorrecto de las peticiones (cabeceras, cuerpo, parámetros) que suelen provocar códigos de respuesta erróneos. Además de revisar los flujos a través de Apache Kafka u otros intermediarios.

Registro de actividad de Kubernetes

Revisar logs para detección de actividad anómala dentro del clúster de Kubernetes como acceso denegado a recursos protegidos, creación de usuarios, cuentas, modificación de roles, modificación de permisos de nombre de espacios o acceso a Secretos.

1.1 Objetivos

En presente tesis, el objetivo principal es el estudio de la parte de logging dentro del contexto de observabilidad en un clúster Kubernetes. En primer lugar, se pretende la puesta en marcha de una infraestructura de un clúster Kubernetes en local.

En segundo lugar, se realizará el análisis de herramientas nativas para el control de logs así como comandos que ejecutan contra la API.

En tercer lugar, el objetivo es la instalación de Elasticsearch, Logstash y Kibana dentro del clúster para dotar de persistencia a los logs de las aplicaciones que se ejecutan sobre Kubernetes.

En cuarto lugar, se pretende realizar el despliegue del clúster en Google Cloud y analizar las posibilidades que ofrece las herramientas nativas para análisis, control y visualización de logs. Además se analizarán servicios adicionales que ofrece Google Cloud para dar solución a la persistencia de logs. En caso de no estar resuelta la persistencia de logs, se instalará la pila Elasticsearch, Logstash y Kibana.

Finalmente, se realizarán pruebas sobre el clúster para verificar la correcta instalación de la solución de logging desarrollada.

1.2 Metodologías

Para llevar a cabo los objetivos, se pretende en primer lugar elaborar un marco teórico que sirva de referencia y estudio de las tecnologías utilizadas en este proyecto.

Una vez estudiados los conceptos sobre un clúster Kubernetes y los diferente objetos, se empezará con la implementación práctica.

La implementación se divide en dos grandes bloques:

1. Instalación Local.

Se crearán máquinas virtuales sobre el hipervisor de VirtualBox en Windows. El clúster estará formado por 3 nodos: 1 nodo maestro y 2 nodos trabajadores.

Tras crear las máquinas virtuales, se instalará Kubernetes mediante Ansible. A continuación, se estudiará la posibilidad de analizar logs de aplicaciones sencillas que ejecutarán sobre el clúster, se estudiará la API, como otros recursos como cuadros de mando para obtención de logs.

Posteriormente, sobre el nodo máster, se instalará la pila Elasticsearch, Logstash y Kibana para analizar, almacenar y visualizar logs.

2. Instalación Google Cloud.

Despliegue del clúster en Google Cloud (formado por 3 nodos) y estudio de métodos para la obtención de logs documentando precios de los costes en la nube.

1.3 Estructura

En este apartado, se describe como se estructuran los diferentes secciones de este documento. Este documento está formado por 7 capítulos:

En el primer capítulo (Introducción y estructura) se encuentra la introducción y se detallan los objetivos a alto nivel, así como la estructura de este documento.

En el capítulo segundo (Estado del arte) , se comenta brevemente los antecedentes de las aplicaciones para finalmente introducir el orquestador utilizado: Kubernetes. Con este apartado, se pretende familiarización con conceptos, herramientas de línea de comando y recursos de Kubernetes. También se realizará una breve introducción de logging.

En el capítulo tercero se detalla el problema de logs en un clúster de Kubernetes y se plantea el problema a resolver.

El capítulo cuarto (Solución propuesta) da comienzo al desarrollo de la implementación para satisfacer las necesidades planteadas. Para ello, en primer lugar, se detalla el plan de trabajo para cumplir los diferentes objetivos. A continuación, se encuentra la elaboración del presupuesto con costes fijo y variables. En tercer lugar, se listan las tecnologías utilizadas en el presente desarrollo. Finalmente, en último apartado de este capítulo, se implementa la solución.

En el quinto capítulo (Implantación), se realizan pruebas sobre los clústeres y se analizan los resultados obtenidos.

En el sexto capítulo, se concluye el desarrollo resumiendo los objetivos marcados al inicio de este a nivel de trabajo realizado y personales. Además se comentan los objetivos futuros de este proyecto y la relación con los estudios cursados.

En el último capítulo, se encuentran referencias y bibliografía consultada.

2. Estado del arte

En este apartado, se repasa la evolución de las aplicaciones y en que influye la tecnología de Kubernetes para el despliegues de aplicaciones. Además, se verán ventajas que aporta la gestión de logs en un clúster de Kubernetes.

2.1 Antecedentes

Las aplicaciones monolíticas presentan una gran desventaja respecto aquellas que se han desarrollado con la arquitectura de microservicios. El problema de estas aplicaciones, a menudo, aparece a la hora de despliegue, ya que se despliegan como una unidad. Las aplicaciones monolíticas presentan las siguientes ventajas:

- Desarrollo sencillo y unificado.
- Facilidad para realizar pruebas de integración ya que no se prueban hasta finalizar todo el código.
- Repositorio centralizado que facilita la búsqueda de errores o fallos en un único sitio.

Sin embargo, estas aplicaciones también presentan siguientes desventajas:

- Alto acoplamiento entre las clases que no permite el escalado por componentes provocando altos costes computacionales.
- Velocidad de desarrollo más lento ya que al aumentar el acoplamiento entre clases, se hace poco mantenible el código.
- Ante un fallo, provoca caída de la aplicación completa.
- Falta de flexibilidad, ya que toda la aplicación se escribe en un mismo lenguaje de programación.
- Utilización de lenguajes y versiones obsoletas.
- Código no mantenible a lo largo del tiempo.
- Actualizaciones del software y nuevos requisitos provocan restructuraciones complejas de toda la aplicación.
- Mantenimiento caros y complejos que provocan parones de la aplicación durante largos periodos de tiempo para el cliente.

Para ello, es importante que las aplicaciones se desarrollen utilizando patrones de arquitectura como microservicios para backend y patrón microfrontend para frontend que solucionan la

problemática dividiendo la aplicación en microcomponentes. La comunicación entre diferentes partes se realiza mediante API REST.

Una vez, desarrollado las aplicaciones, entra en contexto los contenedores que son unidades pequeñas que encapsulan la aplicación junto a librerías necesarias y exponen un punto de entrada para que otros componentes puedan interactuar con la aplicación vía API REST.

Además, si se pretende desplegar las aplicaciones en la nube, es imprescindible esta labor de encapsulación utilizando alguna herramienta como Docker.

Docker es una tecnología que permite crear y gestionar contenedores de forma simplificada. En el apartado de contexto tecnológico se comentará más en detalle.

Para tener una aplicación escalable y tolerante a fallos, se necesita un gestor o unidad de administración de contenedores llamado orquestador de contendores.

Los orquestadores de contenedores son herramientas que agrupan sistemas para formar clústeres en los que corren aplicaciones basados en microservicios. Estos orquestadores son encargados de gestionar el ciclo de vida de los contenedores.

Algunos orquestador de contenedores son:

- Kubernetes: Es una herramienta de orquestación de código abierto originalmente desarrollada por Google.
- Open Shift: Se trata de una plataforma de contendores de Kubernetes para empresas.
- Docker Swarm: Es una herramienta integrada en Docker que permite gestionar de forma centralizada contenedores de Docker formando un clúster.
- Amazon Elastic Container Service: En entornos cloud, Amazon ofrece esta herramienta para ejecución y orquestación de contenedores.
- Azure Container Instances: Por otro lado, Azure ofrece su propio servicio de orquestación.

Un orquestador de contenedores presenta siguientes ventajas:

- Gestión de los contenedores de forma sencilla y automática optimizando recursos.
- Permite formar un conjunto de nodos y ofrecer como un único macro servicio.
- Permite gestionar clústeres automáticamente: definición de políticas de acceso, escalado horizontal y vertical, encapsulación de servicios, comunicación entre contenedores dentro del clúster.

2.2 Kubernetes

Hoy en día, las aplicaciones son desarrolladas como microservicios. Estas son encapsuladas dentro de contenedores y lanzadas como componentes mediante un orquestador de contenedores.

Kubernetes es un orquestador de contenedores de código abierto, ligero, portable para administrar cargas de trabajo y servicios en contenedores. Se encarga de gestionar contenedores que ejecutan las aplicaciones y formar un sistema distribuido compacto.

Kubernetes ofrece:

- Abstracción de la infraestructura de hardware exponiendo la funcionalidad de la aplicación como un único recurso.
- Escalabilidad: Aumentar o disminuir contenedores y nodos para satisfacer necesidades de la demanda de forma totalmente automática. Además, optimiza la utilización de recursos sin afectar la disponibilidad.
- Permite realizar despliegues automáticos de aplicaciones integrando con herramienta de despliegue DevOps.
- Aislamiento entre aplicaciones dentro del clúster.
- Remplazo de contenedores caídos de forma automática sin suponer una falta de disponibilidad de la aplicación.
- Orquestación declarativa e imperativa.
- Descubrimiento de servicios y equilibrio de carga de forma automática entre los diferentes componentes.
- Despliegues en local y en remoto.

2.2.1 Arquitectura

En este apartado, se estudiará la arquitectura de Kubernetes.

Un clúster de Kubernetes está formado por cientos o miles de nodos que se pueden clasificar en dos tipos:

- Nodo maestro que contiene el plano de control de Kubernetes que gestiona toda el sistema y arquitectura del clúster.
- Nodos trabajadores sobre los cuales se ejecutan las aplicaciones.

En la figura 1, se puede observar la arquitectura y la relación entre las partes.

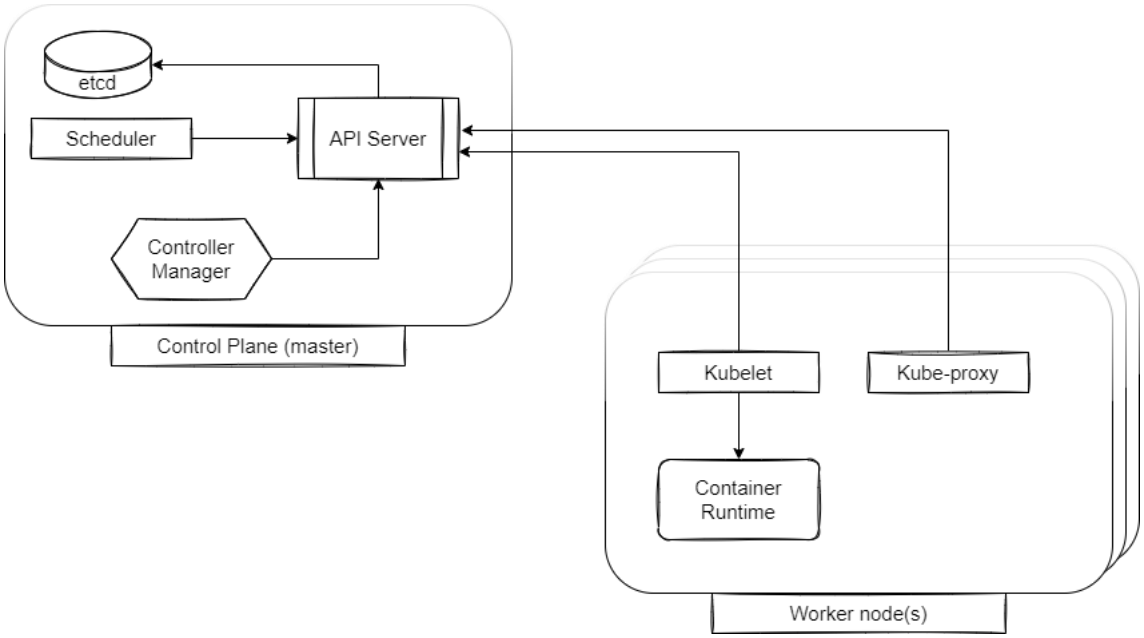

Figura 1: Arquitectura de un clúster de Kubernetes (elaboración propia).

El plano de control es el componente principal y el nodo central del clúster. Se encarga de hacer funcionar todo el sistema. Es muy habitual encontrar más de un nodo maestro para asegurar la alta disponibilidad. Está formado por los siguientes componentes:

- API Server: Es el objeto que expone la funcionalidad del plano de control en un clúster Kubernetes. Se comunican tanto componentes del plano de control como nodos.
- Scheduler: Planificador que se encarga de asignar nodos trabajadores a cada componente ejecutable de la aplicación.
- Controller Manager: Se encarga de gestionar el clúster como la replicación de componentes, seguimiento de nodos trabajadores o gestión de errores en nodos.
- ETCD: Sistema persistente distribuido que se encarga de almacenar la configuración del clúster.

El plano de control no ejecuta las aplicaciones, de esto se encargan los nodos trabajadores.

Los nodos trabajadores ejecutan microservicios encapsulados en contenedores. Está formado por diferentes componentes:

- Container runtime: Tecnología que se encarga de ejecutar contenedores.
- Kubelet: Agente que se comunica con el API Server para gestionar ciclo de vida de contenedores dentro del nodo.

- Kube-proxy: Proxy del servicio que balancea la carga de tráfico de red entre los componentes de la aplicación.

La comunicación entre los diferentes componentes se realiza mediante llamadas API REST mediante el componente API Server, no existiendo comunicación directa.

La comunicación es iniciada por otros componentes hacia API Server, sin embargo, se puede dar el caso de que API Server se comunique con kubelet para visualizar los logs o conectarse a un contenedor en concreto.

Para asegurar la alta disponibilidad, se pueden replicar instancias de nodo maestro, esto no ocurre para todos sus componentes: los componentes ETCD y API Server pueden ejecutar múltiples instancias para ejecutar trabajos en paralelo, sin embargo, para los componentes Scheduler y Controller se debe ejecutar una única instancia en un instante de tiempo, estando en el resto de los nodos, en modo reposo.

2.2.2 Kubectl

Como se ha visto en apartado anterior, el plano de control está formado por API Server que es el encargado de gestionar las comunicaciones a través de API. Este API puede ser consumido por usuarios o diferentes componentes del clúster.

Esta API permite consultar a la información del clúster, así como manipulación del estado de los objetos dentro de un clúster de Kubernetes como pods, ConfigMap, Deployments, nombre de espacios, etc.

Las operaciones al API a menudo se realizan a través de línea de comandos kubectl o Kubeadm.

Kubectl es una herramienta de línea de comandos que permite comunicar con el control de plano de Kubernetes a través de la API. Permite crear recursos para las aplicaciones y el clúster, interactuar con contenedores y gestionar el clúster.

A continuación, se puede consultar algunos comandos más utilizados:

- Kubectl get nodes // Obtiene todos los nodos.
- Kubectl get deployments // Obtiene los objetos Deployments de un clúster.
- Kubectl cluster-info // Ofrece información del clúster.
- Kubectl apply – f file.yml // Aplicar una configuración al clúster descrito dentro del fichero “file.yml”.
- Kubectl delete // Elimina uno o varios recursos.

Se puede consultar la guía completa aquí [1].

2.2.3 Recursos en K8S

En Kubernetes existen diferentes tipos de recursos para la configuración, mantenimiento y despliegue de aplicaciones basadas en microservicios. Por ello, en este apartado, se van a comentar los tipos más importantes que se utilizarán durante el despliegue del desarrollo práctico.

Las aplicaciones se pueden desplegar en uno o varios nodos, además, estas aplicaciones se ejecutan dentro de los contenedores de los pods.

2.2.3.1 Pod

Un pod es la agrupación de uno o varios contenedores, tiene un almacenamiento volátil que perdura la vida útil del pod. Posee un identificador que le permite diferenciar dentro del resto de pods de una red compartida.

Un pod se ejecuta dentro de un nombre de espacio de Kubernetes determinado. Existe el espacio de nombre “kube-system” reservado para ejecutar recursos del sistema. Cuando se realiza un despliegue en la nube, dependiendo de la configuración, el espacio de nombre “kube-system” no es accesible ni permite el despliegue de aplicaciones.

Los pods se ejecutan dentro de los nodos del clúster, cuando un nodo muere, los pods ejecutando dentro del mismo, desaparecen junto a toda la información.

Se puede interactuar con los pods mediante kubectl:

- Kubectl get pods // lista todos los pods.
- Kubectl describe pod podName // revela información relativa al pod.
- Kubectl get pods -o wide // lista con más detalle todos los pods.
- Kubectl logs podName //devuelve pods del log.
- Kubectl exec -it podName /bin/bash // lanza una terminal interactiva dentro del pod.

2.2.3.2 Deployment

En un objeto de tipo Deployment, existen dos estados: estado deseado y estado actual. Este objecto se encarga de alcanzar el estado deseado de forma controlada.

Los casos de uso más comunes para utilizar este controlador son:

- Alcanzar nuevo estado de los pods.

[1] https://kubernetes.io/docs/reference/kubectl/cheatsheet/

- Retroceder a una versión anterior de un Deployment.
- Escalar horizontalmente el Deployment para soportar más carga.

Se puede interactuar con los Deployments mediante kubectl:

- Kubectl get deployments // Lista todos los objetos Deployments.
- Kubectl rollout history deployment/nameDeployment //Retrocede al despliegue anterior.
- Kubectl logs deploy/nameDeployment // Devuelve logs del deployment.
- Kubectl port-forward deploy/nameDeployment hostPort: remotePort // Escucha en el puerto local y reenvía las peticiones en un pod creado mediante deployment.

2.2.3.3 DaemonSet

DaemonSet es un tipo de controlador que garantiza que se ejecuta una copia de pod dentro de cada nodo del clúster. Si se añaden más nodos en el clúster, este objeto se encargará de que se repliquen en cada pod.

Los casos de uso más comunes son:

- Ejecutar un proceso de almacenamiento en el clúster.
- Ejecutar proceso de recolección de logs en cada nodo.

Este tipo de objetos crean pods al iniciar el nodo, al principio de todo. Este tipo de objetos guardan similitud con Deployment, ya que ambos crean pods, sin embargo, un Deployment sirve para definir un despliegue sin estado.

Una vez que se elimina el objeto DaemonSet, se limpian los pods que se hayan creado durante su ejecución.

2.2.3.4 Servicio

Un servicio es un objeto de Kubernetes que describe como se accede a las aplicaciones, como puerto, pods específicos y balanceadores de carga.

- kubectl port-forward svc/my-service 5000 // redireccionamiento del servicio en el puerto indicado.
- Kubectl get services // lista todos los servicios en el nombre de espacio por defecto.
- Kubectl delete service serviceName //borra el servicio y no los pods a los que redirecciona.

A continuación se comentan los tipos de servicio utilizados en este desarrollo:

- ClusterIP: Se trata del servicio por defecto, si no se especifica el servicio. Expone el servicio en una dirección IP interna del clúster (únicamente es accesible desde dentro del clúster).
- NodePort: Expone el servicio en la IP del nodo en un puerto estático, permitiendo el acceso al servicio en la dirección: http://IPNodo:puertoServicio. Permite acceder desde fuera del clúster, se considera peligroso ya que no define ninguna política de acceso.
- LoadBalancer: Expone el servicio externamente usando el balanceador de carga. Este tipo de servicio crea un servicio de tipo NodePort y ClusterIP a los cuales apuntará el balanceador externo.

2.3 Logging

Los logs de las aplicaciones pueden ayudar a saber qué ocurre dentro de la aplicación. Los logs son utilizados para depurar la aplicación y para monitorizar la actividad del clúster. La forma más común de registrar logs para las aplicaciones contenerizadas es volcar logs a la salida estándar.

Por defecto, los logs de los pods están almacenados dentro del nodo en el directorio /var/logs/pods. Sin embargo, si el pod cae o se cierra intencionalmente, estos logs desaparecen.

El logging a nivel de nodo consiste en lo siguiente: Exportar logs a una base de datos externa como puedes ser Elastic, StackDriver u otros. Existen agentes como Logstash o fluentd que se encargan de enviar los logs a dicha base de datos. Estos agentes se pueden desplegar mediante el objeto DaemonSet dentro de cada uno de los nodos disponibles o en un nodo en particular.

Sin embargo, si se desea registrar todos los logs de Kubernetes, de forma nativa no existe una solución, pero se puede alcanzar el logging a nivel de clúster mediante distintas aproximaciones:

1. Usar logging a nivel de nodo para todos los nodos del clúster.
2. Utilizar un contenedor sidecar.
3. Emitir logs directamente desde la aplicación a una base de datos.

La primera aproximación es la más común y es la que se va a implementar en este desarrollo para alcanzar el logging a nivel de clúster ya que se puede implementar junto a otras herramientas para filtrar y visualizar logs.

La segunda aproximación consiste en ejecutar un contenedor dentro del mismo pod cuya función es la extracción de logs del pod a un volumen, dependiendo de la configuración del volumen donde se alojen los logs, estos pueden desaparecer tras la muerte del pod. Si se utiliza un volumen persistente, los logs se pueden salvar tras la muerte del pod.

La tercera aproximación consiste en añadir lógica de despliegue a la aplicación, esta solución añade acoplamiento entre el código y la plataforma, siendo poco práctico si se va a desplegar la aplicación en un entorno cloud.

3. Análisis del problema

Kubernetes se trata de una herramienta potente para realizar despliegues de aplicaciones sobre un clúster en un entorno personal o empresarial, en local y en la nube.

En un clúster K8S formado por miles de pods, se hace indispensable la implantación de un sistema que sea capaz de registrar y almacenar logs en un lugar seguro fuera del clúster.

El reto del registro de logs reside en que tras desaparecer el pod, su log deja de estar accesible por el administrador del clúster. La complejidad reside en que los contenedores que se ejecutan dentro de los pods poseen su propio sistema de ficheros, por tanto, cuando se escala el número de pods, es posible que algunos contenedores dejen de existir y por tanto desaparezca su sistema de ficheros (incluyendo logs).

Una de las ventajas que tiene Kubernetes es que las aplicaciones son fácilmente escalables aumentando o reduciendo el número de pods, por tanto, disponer de un sistema de persistencia de logs es fundamental.

Los logs pueden dar información concreta y detallada acerca de los eventos que tanto usuarios individuales, administradores, motor de contenedores (docker), kubelet y otros componentes del sistema ha realizado en el sistema. Además, proporciona un conjunto de registros ordenados de forma cronológica y permite responder a cuestiones como:

- ¿Qué ha pasado? ¿cuándo ha pasado? ¿quién lo ha iniciado? ¿sobre qué ha pasado?¿dónde se ha observado?¿desde dónde se ha iniciado? ¿Por qué un pod muere?

Por tanto, en este desarrollo, se pretende proponer soluciones para la persistencia y filtrado de logs en los entornos local y nube.

4. Solución propuesta

En este apartado se implementará la solución, para ello, en primer lugar, se detalla un plan de trabajo a seguir, a continuación se realizará una estimación del presupuesto, a posteriori se detallarán las tecnologías empleadas durante el desarrollo y finalmente se desarrollará la solución.

4.1 Plan de trabajo

En este apartado se planifican los pasos a seguir para implementar la solución, se distinguirá la solución para el entorno local y para el entorno de nube.

Entorno local

En primer lugar, se crearán tres máquinas virtuales con VirtualBox y se instalará Kubernetes mediante Ansible. Se instalarán dos nodos trabajadores y un nodo máster. Para verificar la correcta instalación se comprobará el estado de los nodos del clúster.

En segundo lugar, se investigará sobre los comandos que se pueden ejecutar con kubectl para obtención de logs de un pod y se investigará la perdurabilidad en el tiempo de dichos logs cuando el pod esté caído o no se encuentre disponible.

Si tras la muerte de un pod, los logs desaparecen del nodo que ejecutaba dicho pod, se analizará la posibilidad de extraer dichos logs fuera del pod. Además, se estudiará la posibilidad de filtrar los logs por cada contenedor.

Tras ello, se estudiará el cuadro de mando ofrecido por la API de Kubernetes como alternativa para obtener logs.

En tercer lugar, tras ver las posibilidades de las soluciones nativas, se instalará la pila de herramientas ELK.

Las tres herramientas se instalarán como pods dentro del nombre de espacio “kube-system”, sobre esos pods, se despliegan servicios que permiten acceder a las interfaces gráficas de dichas herramientas (Elasticsearch y Kibana).

Todas las imágenes se descargarán del repositorio de imágenes Docker de Elastic: “elastic.docker.co”.

Entorno nube

Para la solución en la nube, se utilizará la plataforma de Google Cloud por motivos de accesibilidad frente a AWS y Microsoft Azure.

Por una parte, se crean tres máquinas virtuales que ejecutarán sobre Google Cloud Engine. A continuación, se accederá a ellas vía SSH y se instalará Kubernetes vía Ansible. Estará formado por un nodo máster y dos nodos trabajadores al igual que en el modo local.

A continuación, se instalará la pila ELK sobre el clúster montado y se estudiará la gestión centralizada de logs.

Por otra parte, se desplegará Kubernetes de Google Cloud (funcionalidad nativa) y se estudiará la herramienta nativa para obtención de logs: StackDriver. Esta opción no requiere configuración ni instalación ya que está integrada de forma nativa.

4.2 Presupuesto

En este apartado, se realizará una estimación del coste de este proyecto. Se realizará por una parte, el coste derivado de utilización de recursos hardware y software. A continuación, se realizará una estimación de costes humanos y finalmente se presupuestará el coste del despliegue en Cloud.

- Hardware

El hardware utilizado para el desarrollo de este proyecto ha sido un ordenador portátil de disco SSD de 512Gb y 16Gb de RAM con Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz de un coste de 1000€. Esta equipo es suficiente para ejecutar tres máquinas virtuales que formarán el clúster de Kubernetes.

- Software

En primer lugar, se ha optado por herramientas libres tal que VirtualBox, Google Chrome, Google Drive, nano, Microsoft Office. En segundo lugar, las imágenes Docker utilizadas han sido de libre desarrollo como Alma Linux S.O. En tercer lugar, las herramientas para la gestión centralizada de logs Elasticsearch, Logstash y Kibana se ha utilizado la versión gratuita de las imágenes.

Por tanto, el coste del software es 0€.

- Personal

Este coste hace referencia al coste a percibir por un administrador de clúster. El costo de un administrador junior ronda 25.000€ brutos en España según distintos puestos encontrados en la página de InfoJobs. En base a este sueldo:

25.000€ / 12 meses / 4 semanas / 40 horas/semana = 13€/H. Se calcula una estimación de 400 horas de desarrollo y pruebas de este proyecto.

400horas * 13€/ hora= 5200€.

- Cloud

Para realizar el desarrollo en Cloud se ha utilizado una cuenta gratuita de Google, sin embargo, el uso de Google Cloud no es gratuito. En función de los recursos utilizados como máquinas virtuales, clústeres, discos persistentes, redes con IP fijas, tiene un coste u otro. Google nos ofrece mediante su herramienta de facturación el desglose de la factura por recursos consumidos durante este desarrollo (ver figura 2 y 3).

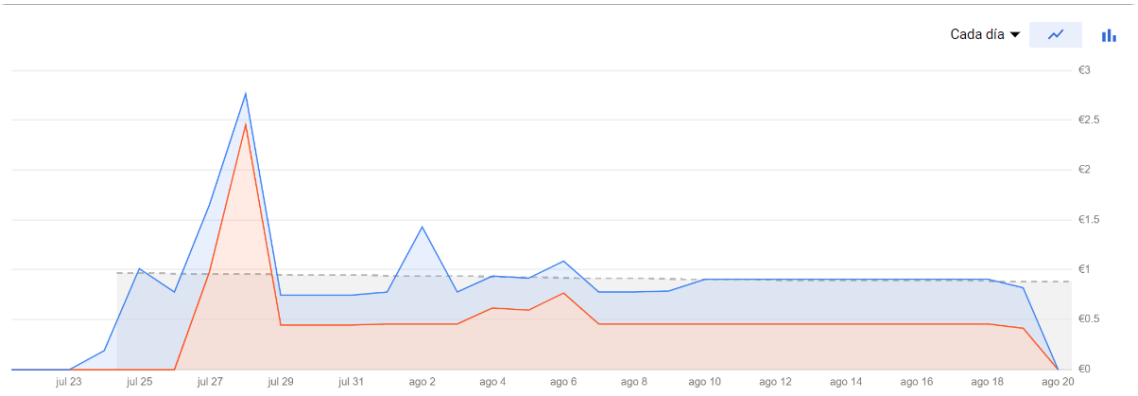

Figura 2: Resumen acumulativo diario, los colores representan distintos proyectos.

Proyecto	ID de proyecto	Número del proyecto	Costo	Descuentos	Promociones y otros créditos	Subtotal
Proyecto cluster Kubernetes	proyecto-cluster-kubernetes	1085788567111	€14.02	–	–	€14.02
My First Project	gold-setup-357312	1091650510287	€11.82	–	–	€11.82
					Subtotal	€25.85
					Impuesto	–
					Total filtrado	€25.85

Figura 3: Resumen de precios totales por proyecto.

Como se aprecia en las figuras 2 y 3, el coste de realizar el proyecto en el entorno Cloud ha sido de 25,85€. El modelo de pago: Uso x horas es realmente muy poco costoso si se desean realizar desarrollos puntuales. Sin embargo, en un entorno empresarial, estos costos incrementarían ya que el clúster ha de estar operativo 24horas*30días* 12 meses al año.

Para realizar la estimación de costes, Google Cloud ofrece una calculadora [2] de precios para estimar el coste de uso de los recursos de antemano por ejemplo si se desea entrenar una red neuronal o se requiere uso de una GPU de alto rendimiento por n días.

- Otros costes

Derivado al desarrollo, existen costes de suministro eléctrico, conexión a Internet, dispositivos adicionales como pantallas, ratón que suman un costo aproximado de 100€.

En la tabla 1 se puede observar el resumen total del desarrollo.

Concepto	Coste (€)
Hardware	1000€

[2] https://cloud.google.com/products/calculator/

Software	0€
Personal	5200€
Cloud	25,85€
Otros costes	100€

Tabla 1: Presupuesto del desarrollo por componentes.

La cantidad asciende a un total de 6325,85€.

4.3 Tecnologías utilizadas

En este apartado, se describirán las tecnologías utilizadas durante el desarrollo.

4.3.1 VirtualBox

VirtualBox[3] se trata de un software de virtualización libre para arquitecturas AMD64 y Intel64. Sirve para generar máquinas virtuales o virtualización de hardware sobre una máquina host.

Figura 4: Logo de VirtualBox

Se trata de una herramienta que permite ejecutar la gran mayoría de distribuciones de Windows, Mac y GNU/Linux y se puede ejecutar sobre Windows, Linux, Mac y Solaris.

VirtualBox se utiliza para crear y ejecutar nodos virtuales del clúster, donde cada nodo será una máquina virtual. VirtualBox permite gestión de la máquina virtual mediante interfaz de forma sencilla, se puede configurar cantidad de CPU, RAM, disco duro, tarjeta de red o tipo de software utilizado.

Además, se opta por esta herramienta ya que es la que se ha utilizado en distintos proyectos durante los años de estudio y no ha presentado limitaciones frente a otras del mercado.

4.3.2 Docker

La herramienta de contenedores Docker [4] es la tecnología de virtualización por contenedores más popular de la industria. Permite crear imágenes y ejecutarlos mediante instancias: contenedores. Comúnmente, se engloba el código de la aplicación junto a las librerías necesarias en una imagen con un S.O instalado ya previamente. A continuación, la imagen creada se sube a un repositorio de imágenes para instanciarlos dentro de un sistema distribuido.

Figura 5: Logo de Docker

[3] https://www.virtualbox.org/
[4] https://www.docker.com/

El despliegue de aplicaciones en Kubernetes se realiza mediante contenedores Docker. Tras realizar el despliegue es necesario gestionar el ciclo de vida estos contenedores mediante un orquestador de contenedores, en este desarrollo, se utilizará Kubernetes.

4.3.3 Ansible

Ansible [5] se trata de una herramienta libre que automatiza procesos para preparación de infraestructura, configuración y organización de los sistemas.

Figura 6: Logo de Ansible

En este desarrollo, se utilizará para instalar Kubernetes sobre las máquinas virtuales: Se conectará a las máquinas vía SSH para instalar todos los demonios y funcionalidades, además de asignar roles de nodo trabajador y máster.

Ansible funciona mediante configuración del inventario y una serie de tareas donde se definen las acciones a realizar en los nodos. Para instalar Kubernetes, se utilizará un proyecto que instala Kubernetes con la versión que se le indique, sin embargo requiere una mínima configuración del inventario (cantidad de nodos, dirección IP de cada nodo y rol dentro del clúster). Tras la configuración del inventario, se lanzará el comando Ansible para que realice la instalación de Kubernetes mediante Kubeadm de forma automática.

4.3.4 Kubeadm

Kubeadm es una herramienta de línea de comandos que permite instalar Kubernetes de forma nativa.


Figura 7: Logo de Kubeadm

Se trata de una herramienta para crear un clúster de Kubernetes de forma manual, aunque se no se encarga de aprovisionar. Algunas de las principales soluciones de Cloud utilizan esta herramienta por debajo para instalar Kubernetes en la nube.

Algunos ejemplos de comandos más utilizado son:

- Kubeadm init // Sirve para iniciar el clúster de Kubernetes.
- Kubeadm join // Sirve para unir nodos al clúster.
- Kubeadm token // Devuelve un token que recibe por parámetro el comando “Kubeadm join” para solicitar unirse al clúster.

[5] https://www.ansible.com/

Se puede consular más información en las referencias de Kubernetes [6].

4.3.5 Elasticsearch, Logstash y Kibana

Elasticsearch, Logstash y Kibana son un conjunto de herramientas de código abierto que combinados ofrecen una solución al problema de persistencia de logs. Permite la monitorización y el análisis de los logs.

Mediante estas herramientas se pretende recopilar toda la información de los logs, procesarla y almacenarla de forma distribuida para posteriormente realizar consultas sobre los logs. A continuación, se comenta que hace cada herramienta de forma individual.

En primer lugar, Elasticsearch se trata de una base de datos distribuida que se instala en todos los nodos, haciendo que sea tolerante a fallos y tenga alta disponibilidad.

En segundo lugar, Logstash realiza el preprocesamiento de los logs y envía la información a la base de datos. Realiza tres etapas: Parte de entrada, filtro y salida.

- Entada: Recoge información de los agentes emisores de beat. Un beat es un elemento que produce información.
- Filtro: Agrega o elimina metadatos a los logs para facilitar posterior filtrado y procesado por Kibana.
- Salida: Envía los logs procesado a Elasticsearch.

En tercer lugar, Kibana se trata de una herramienta visual que realiza consultas a Elasticsearch para filtrar y obtener información en forma de cuadros de mando.

Además de estas tres herramientas, existen otros elementos participantes que son los beats, son definidos como agentes recolectores de información de los pods en distintos formatos: ficheros de logs, salida estándar, métricas del sistema o paquetes de red. Tras recogerlos, se encargan de enviar dicha información a Logstash.

A continuación, se puede observar el diagrama que refleja el flujo de información.

[6] https://kubernetes.io/docs/reference/setup-tools/kubeadm/

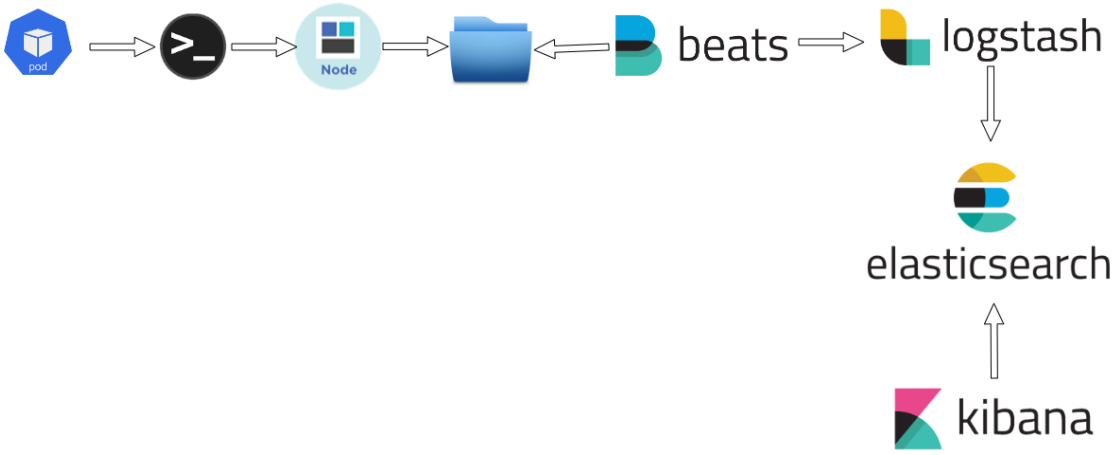

Figura 8: Flujo de información (elaboración propia).

En la figura 8 se puede observar mediante sentido de las flechas el flujo de información entre las distintos entidades y herramientas. En primer lugar, un contenedor que se encuentra dentro del pod produce logs por salida estándar. Dicha información se escribe sobre el sistema de archivos del nodo. Tras esto, el agente beats se encargará de extraer dichos logs y enviarlos a Logstash. Logstash procesará y filtrará los logs para enviarlo a Elasticsearch, que se encarga de almacenarlos. Finalmente, kibana se encarga de previsualizar información realizando consultas a Elasticsearch y generando cuadros de mando.

4.4 Desarrollo e implementación de la solución

En este apartado, se implementará la solución, para ello se divide en siguientes partes: en primer lugar se realizará el estudio de la persistencia de los logs en un clúster en local, en segundo lugar, se desarrollará la solución de persistencia de logs en la nube. En ambas partes, se investigarán alternativas para poder obtener logs tras la muerte del pod.

4.4.1 Despliegue en local

En primer lugar, se crean las máquinas virtuales que formarán el clúster de Kubernetes. Se instalará un total de dos nodos trabajadores y un nodo máster, para ello son necesario tres máquinas virtuales, además de una máquina adicional con ansible que instalará Kubernetes sobre las máquinas virtuales.

Tipo	S.O
Ansible	AlmaLinux-8.6-x86_64 con 1 CPU y 4 Gb RAM y 20Gb ROM dinámico
Master	AlmaLinux-8.6-x86_64 con 1 CPU y 4 Gb RAM y 50Gb ROM dinámico
Worker	AlmaLinux-8.6-x86_64 con 1 CPU y 4 Gb RAM y 50Gb ROM dinámico
Worker2	AlmaLinux-8.6-x86_64 con 1 CPU y 4 Gb RAM y 50Gb ROM dinámico

Tabla 2: Especificación de los nodos en local.

En la tabla 2 , se puede observar los requerimientos de cada máquina virtual. Todas las máquinas van a ejecutar Alma Linux S.O para la arquitectura Intel, contarán con 4Gb de RAM y 20Gb de disco duro, además todas contendrán una red de tipo Adaptador Puente y se ejecutarán en el mismo host.

Una vez creadas todas las máquinas virtuales, se procederá a la instalación del S.O Alma Linux 8.6. La descarga de la imagen se realiza desde este repositorio[7]. Durante la instalación, se ha optado por la versión minimalista de servidor web que incluye interfaz.

Una vez terminada la instalación del sistema operativo en todas las máquinas, se accede a ellas y se comprueba las diferentes redes como se aprecia en la figura 9.

[7] http://repo.ifca.es/almalinux/8.6/isos/x86_64/

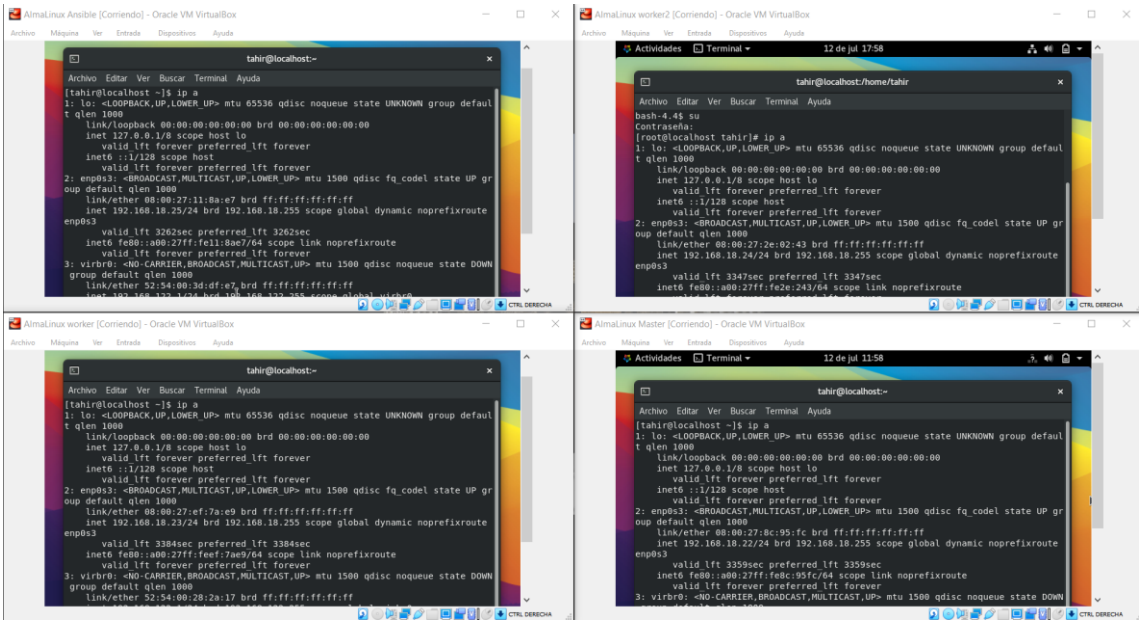

Figura 9: Consulta de IP en las máquinas virtuales.

Como se aprecia en la figura 9, todos las máquinas virtuales poseen una IP además de hallar sobre la misma red (“192.168.18.**”) y conexión a Internet.

En esta fase, se ha terminado de instalar el sistema operativo sobre las máquinas virtuales.

4.4.1.1 Instalación del clúster Kubernetes

En este apartado, se detalla la instalación del clúster Kubernetes. Para ello, son necesario unos pasos de configuración en los nodos trabajadores, máster y nodo ansible. Se distinguen la configuración por un lado de los nodos que formarán el clúster y por otro lado el nodo ansible.

4.4.1.1.1 Configuración de nodos máster y trabajadores

En este apartado, se describe la configuración de los nodos que formarán el clúster.

En primer lugar, se añaden reglas en firewall para la libre comunicación entre los nodos, en concreto se añaden las reglas para los puertos: 6443,10248,10250,10255,10251,10252,10249, 10256,2379,2380. Y se reinicia el firewall. La forma alternativa es apagar y deshabilitar firewall (ver figura 10).

```
[root@localhost tahir]# systemctl stop firewalld | systemctl disable firewalld
Removed /etc/systemd/system/multi-user.target.wants/firewalld.service.
Removed /etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service.
```

Figura 10: Apagando y deshabilitando firewall.

A continuación, se deshabilita selinux para desactivar la política de seguridad como se aprecia en la figura 11.

```
[root@localhost tahir]# cat /etc/sysconfig/selinux

# This file controls the state of SELinux on the system.
# SELINUX= can take one of these three values:
#     enforcing - SELinux security policy is enforced.
#     permissive - SELinux prints warnings instead of enforcing.
#     disabled - No SELinux policy is loaded.
SELINUX=disabled
```

Figura 11: Deshabilitando selinux.

A continuación, se deshabilita swap como se aprecia en la figura 12, se comenta la regla de entrada en el fichero /etc/fstab y se reinicia la máquina virtual para desmontar el espacio de intercambio.

```
[root@localhost tahir]# sudo swapoff -a
[root@localhost tahir]# sudo sed -i '/ swap /s/^/#/' /etc/fstab
[root@localhost tahir]# reboot
```

Figura 12: Deshabilitado uso de swap como memoria virtual.

Con estos pasos, los nodos máster y los nodos trabajadores están preparados para recibir la instalación de Kubernetes.

4.4.1.1.2 Preparación de la máquina ansible

En este apartado, se comentan los pasos necesarios para configurar la máquina ansible.

En primer lugar, se generan un par de claves pública-privada mediante la herramienta ssh-keygen como se aprecia en la figura 13.

```
[root@localhost tahir]# ssh-keygen -q
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
[root@localhost tahir]# cat /root/.ssh/id_rsa
id_rsa      id_rsa.pub
```

Figura 13: Generación de par de clave pública-privada ssh-keygen.

Tras generar el par de claves, se copia la clave pública a las máquinas máster y trabajadoras para que el acceso vía SSH desde la máquina ansible al resto sea sin contraseña como se aprecia en la figura 14.

```
Archivo  Editar  Ver  Buscar  Terminal  Ayuda
[root@localhost tahir]# ssh 192.168.18.23
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Tue Jul 12 18:35:18 2022
[root@localhost ~]# exit
logout
Connection to 192.168.18.23 closed.
[root@localhost tahir]# ssh 192.168.18.22
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Tue Jul 12 12:32:07 2022 from 192.168.18.25
[root@localhost ~]# exit
logout
Connection to 192.168.18.22 closed.
[root@localhost tahir]# ssh 192.168.18.24
Activate the web console with: systemctl enable --now cockpit.socket

Last login: Tue Jul 12 18:34:17 2022
[root@localhost ~]# exit
logout
Connection to 192.168.18.24 closed.
```

Figura 14: Acceso vía SSH a los nodos del clúster sin contraseña.

Tras verificar el acceso a las máquinas que formarán el clúster, se instalan los paquetes Python y el gestor pip que permitirán ejecutar ansible, en concreto se usarán las versiones de la figura 15.

```
[root@localhost kubespray]# python3 -V
Python 3.6.8
[root@localhost kubespray]# pip -V
pip 20.3.4 from /root/.local/lib/python3.6/site-packages/pip (python 3.6)
```

Figura 15: Versiones Python y pip.

A continuación, se clona el proyecto de Kubespray de este repositorio[8] (ver figura 16). Este proyecto es una composición de configuración, inventario y metadatos que permiten instalar Kubernetes mediante ejecución de tareas preconfiguradas. Además, se pueden quitar tareas o configurar nuevas.

[8] https://kubernetes.io/docs/setup/production-environment/tools/kubespray/

```
[root@localhost tahir]# git clone https://github.com/kubernetes-sigs/kubespray.g
it
Clonando en 'kubespray'...
remote: Enumerating objects: 62825, done.
remote: Counting objects: 100% (87/87), done.
remote: Compressing objects: 100% (78/78), done.
remote: Total 62825 (delta 22), reused 55 (delta 6), pack-reused 62738
Recibiendo objetos: 100% (62825/62825), 18.26 MiB | 16.05 MiB/s, listo.
Resolviendo deltas: 100% (35350/35350), listo.
```

Figura 16: Clonar Proyecto Kubespray.

Dentro del proyecto, se encuentran distintos ficheros de configuraciones que se han de modificar, en primer lugar el fichero de inventario. Este fichero indica la configuración del clúster relativa a los nodos maestros, nodos trabajadores, direcciones de red donde se alojan los recursos, roles, capa de red que permitirá la comunicación entre los nodos de forma segura.

Se crea nuevo fichero de inventario, que contiene la configuración del propio clúster como se aprecia en la figura 17. Aquí se le indica las IP de las máquinas que forman el clúster así como roles de los nodos.

```
[root@localhost kubespray]# cat inventory/mycluster/inventory.ini
# ## Configure 'ip' variable to bind kubernetes services on a
# ## different ip than the default iface
# ## We should set etcd_member_name for etcd cluster. The node that is not a etc
d member do not need to set the value, or can set the empty string value.
[all]
node_master ansible_host=192.168.18.22  # ip=10.3.0.1 etcd_member_name=etcd1
node_worker ansible_host=192.168.18.23  # ip=10.3.0.2 etcd_member_name=etcd2
node_worker2 ansible_host=192.168.18.24  # ip=10.3.0.3 etcd_member_name=etcd3

# ## configure a bastion host if your nodes are not directly reachable
# [bastion]
# bastion ansible_host=x.x.x.x ansible_user=some_user

[kube_control_plane]
 node_master

[etcd]
 node_master

[kube_node]
 node_master
 node_worker
 node_worker2

[calico_rr]

[k8s_cluster:children]
kube_control_plane
kube_node
calico_rr
```

Figura 17: Configuración del inventario.

A continuación, es necesario instalar ansible ya que mediante ansible se puede ejecutar el proyecto Kubespray para instalar Kubernetes. Para ello, se instala ansible mediante pip como se observa en la figura 18.

```
[root@localhost kubespray]# pip install ansible
Collecting ansible
  Using cached ansible-4.10.0.tar.gz (36.8 MB)
```

Figura 18: Instalación de Ansible.

El proyecto Kubespray trae por defecto, una serie de requerimientos para la correcta creación del clúster, sin embargo, se ha decidido instalar ansible y ansible-core con las últimas versiones mediante el comando de la figura 18 y el resto de los requerimientos se instalan según las versiones del fichero de requerimientos (ver figura 19).

```
[root@localhost kubespray]# cat requirements.txt
#ansible==5.7.1
#ansible-core==2.12.5
cryptography==3.4.8
jinja2==2.11.3
netaddr==0.7.19
pbr==5.4.4
jmespath==0.9.5
ruamel.yaml==0.16.10
ruamel.yaml.clib==0.2.6
```

Figura 19: Versiones de requerimientos.

Mediante el comando pip, se instalan los requerimientos como se aprecia en la figura 20.

```
[root@localhost kubespray]# sudo pip3 install --user -r requirements.txt
```

Figura 20: Instalación de requerimientos con pip.

Finalmente, se verifica la correcta instalación de las herramientas (ver figura 21).

```
[root@localhost kubespray]# ansible --version
ansible [core 2.11.12]
  config file = /home/tahir/kubespray/ansible.cfg
  configured module search path = ['/home/tahir/kubespray/library']
  ansible python module location = /usr/local/lib/python3.6/site-packages/ansible
  ansible collection location = /root/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/local/bin/ansible
  python version = 3.6.8 (default, Apr 29 2022, 13:46:02) [GCC 8.5.0 20210514 (Red Hat 8.5.0-10)]
  jinja version = 2.11.3
  libyaml = True
```

Figura 21: Verificación de las versiones de las herramientas.

En este paso, se ha terminado de configurar todas las herramientas necesarias en la máquina ansible para comenzar la instalación de Kubernetes.

4.4.1.1.3 Instalación de Kubernetes con ansible

En este apartado, se instala Kubernetes con ansible. Para ello, han de estar todas las máquinas virtuales iniciadas ejecutando sobre el mismo host. En primer lugar, se procede a verificar que el nodo ansible tiene acceso al resto de máquinas haciendo ping a ellas como se aprecia en la figura 22.

```
[root@localhost kubespray]# ansible -i inventory/mycluster/inventory.ini -m ping
 all
[WARNING]: Skipping callback plugin 'ara_default', unable to load
node-worker2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node-master | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
node-worker | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Figura 22: Ping a nodos del clúster.

Como se ha recibido una respuesta por parte de los nodos, se lanza el comando playbook que realizará tareas automatizadas para configurar e instalar el clúster Kubernetes como se ve en la figura 23.

```
[root@localhost kubespray]# ansible-playbook -i inventory/mycluster/inventory.ini --b
ecome-user=root cluster.yml
[WARNING]: Skipping callback plugin 'ara_default', unable to load

PLAY [localhost] ***************************************************************
miércoles 13 julio 2022  19:59:19 +0200 (0:00:00.079)       0:00:00.079 *******
```

Figura 23: Lanzamiento del comando ansible-playbook.

Finalmente, se puede observar el resultado del comando anterior en la figura 24.

```
PLAY RECAP *********************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0    skippe
d=0    rescued=0    ignored=0
node-master                : ok=766  changed=141  unreachable=0    failed=0    skippe
d=1251 rescued=0    ignored=9
node-worker                : ok=500  changed=91   unreachable=0    failed=0    skippe
d=725  rescued=0    ignored=2
node-worker2               : ok=500  changed=90   unreachable=0    failed=0    skippe
d=725  rescued=0    ignored=2

miércoles 13 julio 2022  20:41:51 +0200 (0:00:00.389)       0:42:31.735 *******
===============================================================================
```

Figura 24: Resultado del comando ansible-playbook.

En la figura 24, se puede ver resumen de las tareas ejecutadas en cada nodo del clúster, así como la marca de tiempo y la duración del comando. Una vez verificado que todas las tareas se han completado sin errores (ya que la columna de tareas fallidas es 0), se puede comprobar el estado del clúster en el nodo máster como se ve en la figura 25.

```
[root@node-master tahir]# kubectl get pods -n=kube-system
NAME                                  READY   STATUS    RESTARTS        AGE
calico-node-bv5wf                     1/1     Running   1 (10m ago)     13m
calico-node-jl2dj                     1/1     Running   0               13m
calico-node-zdn2x                     1/1     Running   0               13m
coredns-666959ff67-7l5b7              1/1     Running   0               6m42s
coredns-666959ff67-wnpt8              1/1     Running   0               8m37s
dns-autoscaler-59b8867c86-jmrg5       1/1     Running   0               8m17s
kube-apiserver-node-master            1/1     Running   1               18m
kube-controller-manager-node-master   1/1     Running   5 (5m38s ago)   18m
kube-proxy-bb99w                      1/1     Running   0               15m
kube-proxy-r95xn                      1/1     Running   0               15m
kube-proxy-twlpp                      1/1     Running   0               15m
kube-scheduler-node-master            1/1     Running   5 (5m38s ago)   18m
nginx-proxy-node-worker               1/1     Running   0               14m
nginx-proxy-node-worker2              1/1     Running   0               14m
nodelocaldns-hrk6k                    0/1     Pending   0               8m10s
nodelocaldns-ldft5                    1/1     Running   0               8m10s
nodelocaldns-xksbk                    1/1     Running   0               8m10s
[root@node-master tahir]# kubectl get nodes
NAME           STATUS   ROLES           AGE   VERSION
node-master    Ready    control-plane   18m   v1.24.2
node-worker    Ready    <none>          15m   v1.24.2
node-worker2   Ready    <none>          15m   v1.24.2
```

Figura 25: Comandos para ver pods ejecutando en el nombre de espacio kube-system y nodos del clúster.

Se puede observar que todos los componentes del sistema están ejecutándose de forma correcta en el nombre de espacio kube-system: **calico** se encarga de ofrecer redes IP y seguridad a los contenedores y máquinas virtuales del clúster, **coredns** se encarga de servir como DNS del clúster, **dns-autoscaler** determina si el auto escalado horizontal de DNS ya está activado, **api-server** da servicio a las operaciones REST, **controller** y **scheduller** gestionan y planifican los trabajos dentro del clúster, **kube-proxy** balancea la carga de tráfico de red entre los componentes de la aplicación, **nodelocaldns** mejora el rendimiento del DNS del clúster ejecutando un agente de caché DNS en los nodos del clúster. Además se aprecia en la figura 25, el comando "kubectl get nodes" devuelve los nodos que forman el clúster. Se puede observar que el clúster está formado por 3 nodos: uno máster y dos trabajadores. Todos los nodos ejecutan la versión 1.24.2 de Kubernetes.

4.4.1.2 Logging de contenedores.

En este apartado, se procede a verificar las diferentes formas de obtención de logs de los contenedores de forma nativa dentro de un clúster Kubernetes en local.

En primer lugar, la primera aproximación del registro de logs que es ofrecida por Kubernetes es mediante el comando “kubectl logs”. Este comando vuelca los logs de los contenedores ejecutando dentro del pod por salida estándar.

Para realizar la prueba, se despliega una aplicación que imprime logs por consola. Para ello, se puede usar el fichero yaml de la figura 26.

```
[root@node-master logs-yaml]# cat busybox.yaml
apiVersion: v1
kind: Pod
metadata:
 name: muestraHora
spec:
 containers:
 - name: muestraHora
   image: busybox
   args: [/bin/sh, -c,
           'i=0; while true; do echo "$i: Hola"; i=$((i+1)); sleep 1; done']
```

Figura 26: Definición del pod muestra Hora que imprime el mensaje "${i}: Hola" {i>N} cada 1 segundo por consola.

A continuación, se despliega el pod dentro del clúster mediante el siguiente comando:

#kubectl apply -f busybox.yaml –namespace=mensajeconsola .

El comando kubectl logs permite formatear logs y aplicar diferentes parámetros como añadir la hora en la que se ha obtenido dicho log como se ve en la figura 27.

```
[root@node-master logs-yaml]# kubectl logs muestrahora --timestamps=true -n mens
ajesconsola
2022-07-14T20:49:26.467135780+02:00 0: Hola
2022-07-14T20:49:27.466765544+02:00 1: Hola
2022-07-14T20:49:28.467643069+02:00 2: Hola
2022-07-14T20:49:29.468963507+02:00 3: Hola
2022-07-14T20:49:30.470513826+02:00 4: Hola
2022-07-14T20:49:31.471653284+02:00 5: Hola
2022-07-14T20:49:32.473327115+02:00 6: Hola
2022-07-14T20:49:33.473777971+02:00 7: Hola
```

Figura 27: Logs del pod muestra hora que incluye hora y mensaje impreso por el contenedor.

Para ver logs en tiempo real, se puede aplicar la opción “-f”, sin embargo, no permite seguir registros de varios contenedores a la vez.

Para analizar los logs desde un periodo de tiempo concreto, se puede aplicar el parámetro de la figura 28.

```
[root@node-master logs-yaml]# kubectl logs muestrahora --since-time=2020-07-14T2
0:23:00Z -n mensajesconsola
```

Figura 28: Visualización de logs desde una determinada hora.

Además, se puede acceder a los logs de un contenedor específico mediante parámetro "-c".

Por otra parte, la opción "-p" o "--previous" permite visualizar logs de previas instancias de contenedores dentro de un determinado pod como se ve en la figura 29.

```
[root@node-master logs-yaml]# kubectl logs muestrahora -n mensajesconsola --prev
ious
```

Figura 29: Visualización de los logs de las previas instancias de los contenedores de un pod.

Se pueden analizar todos los parámetros del comando kubectl logs en este enlace [9] o mediante comando de ayuda : “kubectl logs -help”.

En un momento determinado, se pueden exportar los logs a través de un pipeline a un fichero, siempre y cuando el contenedor esté vivo. Además con la opción “-f” permite el almacenado constante de los logs, siendo útil para analizar los últimos logs antes de la detención o fallo del contenedor. (ver figura 30).

```
[root@node-master logs-yaml]# kubectl logs muestrahora -f --timestamps=true -n m
ensajesconsola > /home/tahir/logs-yaml/logs.txt
```

Figura 30: Exportación de los logs del pod a un fichero.

Mediante exportación de los logs a un fichero, se pueden consultar logs de toda la vida de un contenedor y ver así la traza de peticiones incluso el error que provocó que el contenedor deje de existir.

Si se elimina el pod, el comando kubectl logs devuelve error ya que no encuentra un pod activo (ver figura 31).

```
[root@node-master logs-yaml]# kubectl logs muestrahora --since-time=2020-07-14T2
0:23:00Z -n mensajesconsola
Error from server (NotFound): pods "muestrahora" not found
```

Figura 31: Error al visualizar kubectl logs del pod tras su detención.

[9] https://kubernetes.io/docs/reference/generated/kubectl/kubectl-commands#logs

Además, tras la muerte del pod, no se puede usar el comando describe para acceder a la información del contexto del pod (ver figura 32). El comando describe proporciona información relativa al nodo donde se alejó, recursos consumidos por el pod, nombre de espacios, logs, etc.

```
[root@node-master logs-yaml]# kubectl describe pod muestrahora -n mensajesconsol
a
Error from server (NotFound): pods "muestrahora" not found
```

Figura 32: Error al visualizar kubectl describe del pod tras su detención.

En definitiva, tras la muerte del pod y por tanto de los contenedores, los logs dejan de estar accesibles, esto es la volatibilidad del registro de los logs presente en un clúster Kubernetes.

A continuación, se estudiará el cuadro de mando nativo de Kubernetes para ver si ocurre lo mismo que con los comandos ejecutados contra API Server.

Para ello, en primer lugar, se configura dashboard siguiendo este repositorio [10] y se realiza el balanceado de la carga mediante un proxy (ver figura 33).

```
[root@node-master dashboard]# kubectl apply -f clusterrolebinding.yaml
clusterrolebinding.rbac.authorization.k8s.io/admin-user created
[root@node-master dashboard]# kubectl -n kubernetes-dashboard create token admin-user
eyJhbGciOiJSUzI1NiIsImtpZCI6IjNYS1RtSXc5NGRwbFFlbUlZd1V4ZzFuOUcwN1hFSjZJcFlTcmFuRHlUTjAifQ.eyJhdWQiOlsiaHR0cHM6Ly9rdWJlc
m5ldGVzLmRlZmF1bHQuc3ZjLmNsdXN0ZXIubG9jYWwiXSwiZXhwIjoxNjU3OTA3NzgzLCJpYXQiOjE2NTc5MDQxODMsImlzcyI6Imh0dHBzOi8va3ViZXJuZ
XRlcy5kZWZhdWx0LnN2Yy5jbHVzdGVyLmxvY2FsIiwia3ViZXJuZXRlcy5pbyI6eyJuYW1lc3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsInN1cnZpv
2VhY2NvdW50Ijp7Im5hbWUiOiJhZG1pbi11c2VyIiwidWlkIjoiMDc1NjB1YjktOTNkMC00ZTkzLWEwYmUtMWR1MmQ4MzBiZjc0In19LCJuYmYiOjE2NTc5M
DQxODMsInN1YiI6InN5c3R1bTpzZXJ2aWNlYWNjb3VudDprdWJlcm5ldGVzLWRhc2hib2FyZDphZG1pbi11c2VyIn0.NK2yAVQEHXhN0Kloo9mkuBFFY9hiz
SHa1kLlnbMwdtmXYP_8XJbLBkk8_PFmM8Yd10faUayIh0yGKhwrG13mmGCY4aKP16Aj1AB7XXkXhDngGofP3DSbdT9HeKU4Ktuc8T7wLyw3MZC4SuXaBn321
tJRBGIZDL69ZXR5rGZykkMbDCmmpwv7hYgSD_VU9ocdjSEbMGNOzAdUjJQV7i7Gcu_R2L77Aw0AimnGsIDq4gh46nvGPC4fbwQtlPIujQGBKXgzRE4jO08DF
RwlvLIXGSB39yZB8UQFa6ThQQ0NggMfBaD5lrEnZXJ9jPK8JxQe82Bh6AJ3p6zJNIWIlxVJWg
[root@node-master dashboard]# kubectl proxy --address 0.0.0.0 --accept-hosts '.*'
Starting to serve on [::]:8001
```

Figura 33: Creación del objeto ClusterRole, obtención del token y apertura de conexiones.

Para ver los logs mediante dashboard, se crea un deployment y se configuran 3 réplicas como se puede apreciar en la figura 34.

```
[root@node-master tahir]# kubectl create deployment my-first-nginx --image=nginx
 --port=80 --replicas=3
deployment.apps/my-first-nginx created
[root@node-master tahir]# kubectl get pods
NAME                              READY   STATUS             RESTARTS      AGE
my-first-nginx-68454f4649-bqx7g   1/1     Running            0             3m12s
my-first-nginx-68454f4649-lcvhn   1/1     Running            0             3m12s
my-first-nginx-68454f4649-llwbq   0/1     CrashLoopBackOff   2 (35s ago)   3m12s
```

Figura 34: Creación de una aplicación nginx con 3 réplicas: 1 de ellos da error.

En la dirección localhost:8001 se accede al dashboard. En concreto si se va al apartado de los pods, se puede apreciar el despliegue realizado previamente (ver figura 35).

[10] https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

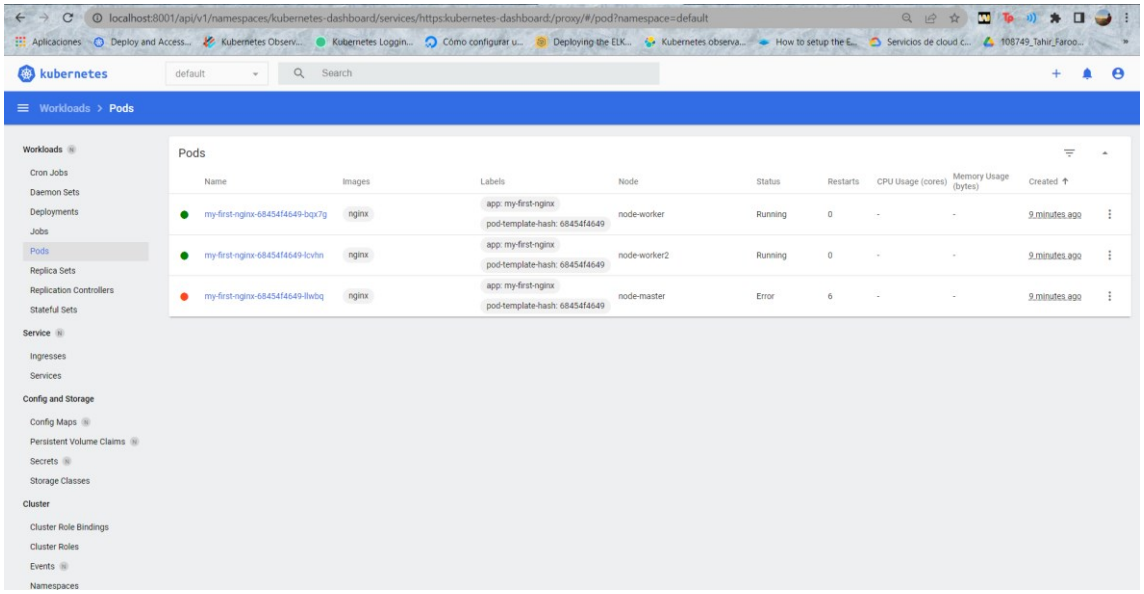

Figura 35: Dashboard accesible en localhost.

Se procede a ver logs del pod vivo haciendo clic sobre los puntos y seleccionando logs. El resultado se puede ver en la figura 36.

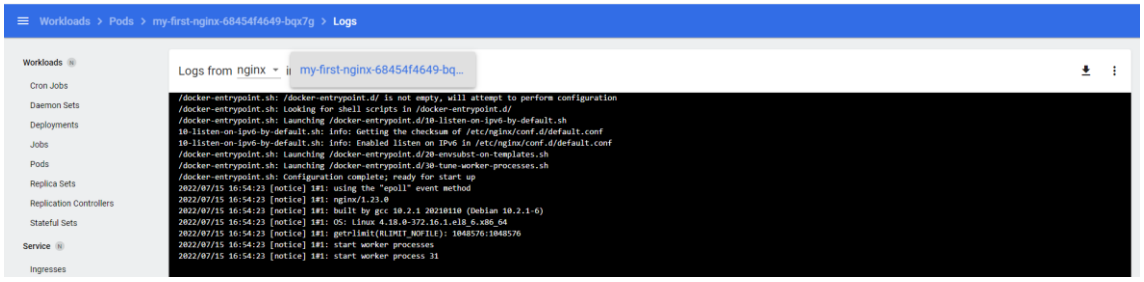

Figura 36: Logs del pod “my-first-nginx”.

A continuación en la figura 37 se puede ver logs del pod que todavía no está en ejecución, sin embargo, existe el pod.

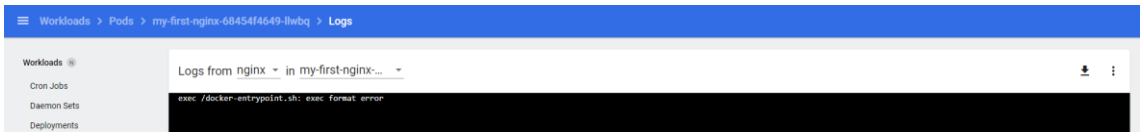

Figura 37: Logs del pod con estado erróneo.

A continuación, se eliminan los pods (ver figura 38).

```
[root@node-master tahir]# kubectl delete deployment my-first-nginx
deployment.apps "my-first-nginx" deleted
[root@node-master tahir]# kubectl get pods
NAME                              READY   STATUS        RESTARTS   AGE
my-first-nginx-68454f4649-bqx7g   1/1     Terminating   0          12m
my-first-nginx-68454f4649-lcvhn   1/1     Terminating   0          12m
my-first-nginx-68454f4649-llwbq   0/1     Terminating   6          12m
```

Figura 38: Se elimina deployment.

Al igual que ocurría con los comandos kubectl, el dashboard deja de mostrar los logs, ya que los pods así como información relativa a ellos, se ha destruido completamente (ver figura 39).

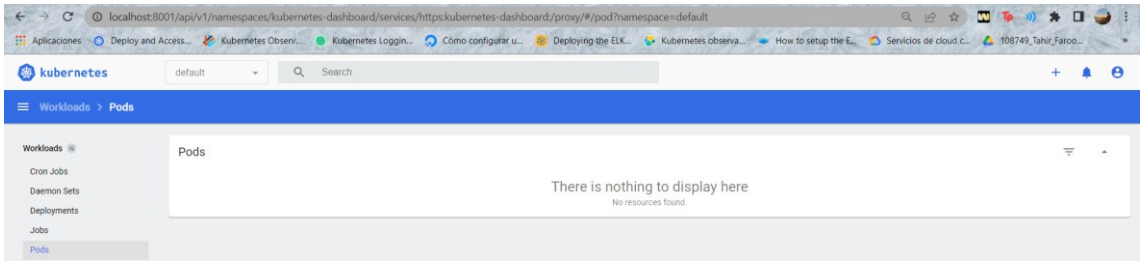

Figura 39: Dashboard no muestra logs de pods que no están vivos.

Como extra, mediante dashboard, se puede ver logs de pods que se encuentran dentro de espacio de nombre kube-system (ver figura 40).

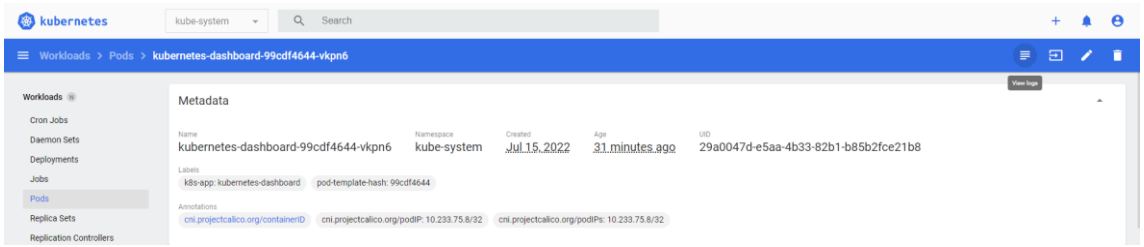

Figura 40: Obtención de logs de pods que se ejecutan en el espacio de nombre kube-system.

Actualmente, no se encuentra una solución nativa para el correcto registro de todos los pods incluso para logs del sistema. En un entorno real, en el cual se tiene que monitorizar miles de nodos y sus respectivos pods, se hace indispensable la gestión de logs mediante herramientas no nativas. Tras ver limitaciones de la herramientas nativas así como comandos, se instala ELK.

4.4.1.3 ELK

En este apartado, se instalará la pila de herramientas Elasticsearch, Logstash y Kibana sobre el clúster Kubernetes instalado en el apartado anterior.

En este desarrollo se utilizarán imágenes del repositorio de Docker de Elasticsearch: docker.elastic.co. Todas las herramientas serán instaladas dentro del clúster de Kubernetes.

4.4.1.3.1 Instalación Elasticsearch

En este apartado, se detalla la instalación de Elasticsearch.

En primer lugar, se configuran permisos, para ello, se crea una cuenta de servicio ("Service Account [11]"). Una cuenta de servicio da identidad a los pods para permitir autenticar contra el API Server. Se crea la cuenta de servicio que se aprecia en la figura 41.

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: elasticsearch-logging
  namespace: kube-system
  labels:
    k8s-app: elasticsearch-logging
---
```

Figura 41: Definición de Service Account.

A continuación, se define un objeto de tipo ClusterRole [12]. Se trata de un objeto que especifica unos permisos a determinados recursos, se define a nivel de clúster. En este caso, se conceden permisos de lectura de servicios, nombre de espacios y los diferentes puntos de entrada (ver figura 42).

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: elasticsearch-logging
  labels:
    k8s-app: elasticsearch-logging
rules:
- apiGroups:
  - ""
  resources:
  - "services"
  - "namespaces"
  - "endpoints"
  verbs:
  - "get"
```

Figura 42: Definición de ClusterRole.

Posteriormente, se define el objeto ClusterRoleBinding [13] que va a permitir aplicar los permisos definidos al clúster (ver figura 43).

[11] https://kubernetes-on-aws.readthedocs.io/en/latest/user-guide/service-accounts.html
[12] https://kubernetes.io/docs/reference/access-authn-authz/rbac/
[13] https://docs.openshift.com/container-platform/3.11/rest_api/rbac_authorization_k8s_io/clusterrolebinding-rbac-authorization-k8s-io-v1.html

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: kube-system
  name: elasticsearch-logging
  labels:
    k8s-app: elasticsearch-logging
subjects:
- kind: ServiceAccount
  name: elasticsearch-logging
  namespace: kube-system
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: elasticsearch-logging
  apiGroup: ""
```

Figura 43: Definición de ClusterRoleBinding.

Después, mediante el comando “kubectl apply” se crean las objetos (ver figura 44).

```
[root@node-master permisos]# kubectl apply -f rbac.yaml
serviceaccount/elasticsearch-logging created
clusterrole.rbac.authorization.k8s.io/elasticsearch-logging created
clusterrolebinding.rbac.authorization.k8s.io/elasticsearch-logging created
```

Figura 44: Aplicación del comando kubectl.

En segundo lugar, se prepara el despliegue de Elasticsearch. Para ello, se prepara un fichero “yaml” que contendrá la definición de un StatefulSet. El objeto Statefulset garantiza la persistencia identificando los volúmenes, de esta forma si los pods de una aplicación StatefulSet caen, el nuevo pod que remplace utilizará el volumen del anterior. Se especifican un total de 3 réplicas que levantan contra el mismo volumen persistente para ofrecer alta disponibilidad.

```
        - containerPort: 9200
          name: rest
          protocol: TCP
        - containerPort: 9300
          name: inter-node
          protocol: TCP
        volumeMounts:
        - name: elasticsearch-data
          mountPath: /usr/share/elasticsearch/data
        env:
          - name: cluster.name
            value: k8s-logs
          - name: node.name
            valueFrom:
              fieldRef:
                fieldPath: metadata.name
          - name: discovery.seed_hosts #list of master eligible nodes that will seed node discovery process
            value: "es-cluster-0.elasticsearch,es-cluster-1.elasticsearch,es-cluster-2.elasticsearch"
          - name: cluster.initial_master_nodes #list of master eligible nodes that will participate in master election
            value: "es-cluster-0,es-cluster-1,es-cluster-2"
          - name: ES_JAVA_OPTS
            value: "-Xms512m -Xmx512m"
      initContainers:
      - name: fix-permissions #change owner of group to elasticsearch, defaults to root
        image: busybox
        command: ["sh", "-c", "chown -R 1000:1000 /usr/share/elasticsearch/data"]
        securityContext:
          privileged: true
        volumeMounts:
        - name: elasticsearch-data
          mountPath: /usr/share/elasticsearch/data
  volumeClaimTemplates:
  - metadata:
      name: elasticsearch-data
      labels:
        app: elasticsearch
    spec:
      accessModes: [ "ReadWriteOnce" ]
      resources:
        requests:
          storage: 100Gi
```

Figura 45: Definición de un objeto StatefulSet.

En la figura 45 se puede observar que para el despliegue de Elasticsearch se utilizará un total de 3 réplicas que ejecutarán la versión de 7.2.0 de Elasticsearch. Cada nodo utiliza el volumen persistente que se montará sobre el fichero de sistema interno del contenedor en la ruta: /usr/share/Elasticsearch/data.

Se crea el objeto mediante el comando kubectl de la figura 46.

```
[root@node-master elastic]# kubectl apply -f estados.yaml
statefulset.apps/elasticsearch-logging created
```

Figura 46: Creación del objeto StatefulSet.

En tercer lugar, se define un servicio que va a escuchar en el puerto 9200 para que Logstash pueda comunicar con Elastic (ver figura 47).

```
kind: Service
apiVersion: v1
metadata:
  name: elasticsearch
  # namespace: efk-logging
  labels:
    app: elasticsearch
spec:
  selector:
    app: elasticsearch
  clusterIP: None
  ports:
    - port: 9200
      name: rest
    - port: 9300
      name: inter-node
```

Figura 47: Definición de un servicio en el puerto 9200.

Se aplica el servicio mediante el comando “kubectl” (véase figura 48).

```
[root@node-master servicio]# kubectl apply -f service.yaml
service/elasticsearch-logging created
```

Figura 48: Aplicación del servicio mediante kubectl.

Este servicio no será accesible desde fuera del clúster. Para permitir el acceso, se usará un proxy que redirigirá todo el tráfico proveniente al clúster en el puerto 9200 al puerto 9200 del servicio que se acaba de desplegar (ver figura 49).

```
[root@node-master servicio]# kubectl port-forward -n kube-system svc/elasticsearch
-logging 9200:9200
Forwarding from 127.0.0.1:9200 -> 9200
Forwarding from [::1]:9200 -> 9200
Handling connection for 9200
Handling connection for 9200
Handling connection for 9200
Handling connection for 9200
```

Figura 49: Redireccionamiento del puerto 9200 al 9200 del servicio Elasticsearch.

Si se realiza una petición a localhost:9200, se obtiene la respuesta por parte del servicio (ver figura 50).

```
[root@node-master logstash]# curl localhost:9200
{
  "name" : "vPv8Jch",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "2RX1u3L7SRmZtdDY7_01mQ",
  "version" : {
    "number" : "6.8.4",
    "build_flavor" : "default",
    "build_type" : "docker",
    "build_hash" : "bca0c8d",
    "build_date" : "2019-10-16T06:19:49.319352Z",
    "build_snapshot" : false,
    "lucene_version" : "7.7.2",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Figura 50: Respuesta del servicio en el puerto 9200.

4.4.1.3.2 Instalación Logstash

En primer lugar, se configura un configMap [14]. Se trata de un objeto que sirve para almacenar datos en el formato clave-valor. En ella, se pueden definir variables a incluir dentro de un fichero.

En la figura 51 se configuran un par de ficheros, Logstash.yml y Logstash.conf. El primero de ellos almacena configuración de Logstash tanto las interfaces como la ruta de la tubería que ejecutará Logstash. El segundo fichero de configuración indica la entrada de información que se realizará en el puerto 5044, además se indica la transformación de datos mediante los diferentes filtros (grok, date, geoip), finalmente se configura la salida a la base de datos Elasticsearch en el puerto 9200.

```
[root@node-master logstash]# cat logstash-config.yml
apiVersion: v1
kind: ConfigMap
metadata:
  name: logstash-configmap
  namespace: kube-system
data:
  logstash.yml: |
    http.host: "0.0.0.0"
    path.config: /usr/share/logstash/pipeline
  logstash.conf: |
    input {
      beats {
        port => 5044
      }
    }
    filter {
      grok {
          match => { "message" => "%{COMBINEDAPACHELOG}" }
      }
      date {
        match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]
      }
      geoip {
          source => "clientip"
        }
      }
      output {
        elasticsearch {
          hosts => ["elasticsearch-logging:9200"]
      }
    }
```

Figura 51: ConfigMap para Logstash

[14] https://kubernetes.io/es/docs/concepts/configuration/configmap/

A continuación, se aplica el objeto configMap (ver figura 52).

```
[root@node-master logstash]# kubectl apply -f configmap.yaml
configmap/logstash-configmap created
```

Figura 52: Aplicación del configMap mediante kubectl.

En segundo lugar, se crea un objeto deployment que utiliza el anterior configMap, la imagen oficial de Logstash y escuchará en el puerto 5044 (en la figura 53 se puede ver parte de configuración de la instalación de Logstash).

En la figura 53 se puede observar que se crea un objeto de tipo Deployment que cuenta con una única réplica. Se hace uso de la imagen oficial de Logstash con la versión 6.3.0. El contenedor escuchará en el puerto de entrada 5044 para permitir llegar la información a través de los agentes. Además, dentro del contenedor se monta el ConfigMap como los volúmenes "config-volume" y "logstash-pipline-volume" en la ruta /usr/share/Logstash.

```
[root@node-master logstash]# cat logstash-deployment.yml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: logstash-deployment
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: logstash
  replicas: 1
  template:
    metadata:
      labels:
        app: logstash
    spec:
      containers:
      - name: logstash
        image: docker.elastic.co/logstash/logstash:6.3.0
        ports:
        - containerPort: 5044
        volumeMounts:
          - name: config-volume
            mountPath: /usr/share/logstash/config
          - name: logstash-pipeline-volume
            mountPath: /usr/share/logstash/pipeline
      volumes:
      - name: config-volume
        configMap:
          name: logstash-configmap
          items:
            - key: logstash.yml
              path: logstash.yml
      - name: logstash-pipeline-volume
        configMap:
          name: logstash-configmap
          items:
            - key: logstash.conf
              path: logstash.conf
```

Figura 53:Deployment para crear Logstash.

Se aplica el deployment mediante el comando "kubectl" de la figura 54 que da como resultado la creación correcta de los pods.

```
[root@node-master logstash]# kubectl apply -f logstash-deployment.yml --validate
=false
deployment.apps/logstash-deployment created
```

Figura 54: Aplicación del comando kubectl.

Finalmente, se crea un servicio para Logstash en el puerto 5044 de tipo ClusterIP (ver figura 55).

```
[root@node-master logstash]# cat servicio.yaml
kind: Service
apiVersion: v1
metadata:
  name: logstash-service
  namespace: kube-system
spec:
  selector:
    app: logstash
  ports:
  - protocol: TCP
    port: 5044
    targetPort: 5044
```

Figura 55: Creación del servicio de Logstash.

Y se crea el servicio mediante el comando de la figura 56.

```
[root@node-master logstash]# kubectl apply -f servicio.yaml
service/logstash-service created
```

Figura 56: Aplicación del servicio mediante comando kubectl.

Al final de la instalación, se puede comprobar pods de Elasticsearch y Logstash ejecutando en el nombre de espacio “kube-system” (ver figura 57). Se puede observar que se han creado dos pods de Elasticsearch “Elasticsearch-logging-0” y “elasticsearch-logging-1” y un pod de Logstash “logstash-deployment-*”.

```
[root@node-master logstash]# kubectl get pods -n kube-system
NAME                                   READY   STATUS    RESTARTS         AGE
calico-node-bv5wf                      1/1     Running   11 (16m ago)     3d1h
calico-node-jl2dj                      1/1     Running   8 (80m ago)      3d1h
calico-node-zdn2x                      1/1     Running   7 (16m ago)      3d1h
coredns-666959ff67-rnwm8               1/1     Running   7 (5m35s ago)    95m
coredns-666959ff67-wnpt8               1/1     Running   58 (6m40s ago)   3d1h
dns-autoscaler-59b8867c86-hkt7q        1/1     Running   1 (16m ago)      95m
elasticsearch-logging-0                1/1     Running   0                14m
elasticsearch-logging-1                1/1     Running   19 (5m18s ago)   62m
kube-apiserver-node-master             1/1     Running   8 (80m ago)      3d1h
kube-controller-manager-node-master    1/1     Running   29 (80m ago)     3d1h
kube-proxy-bb99w                       1/1     Running   7 (16m ago)      3d1h
kube-proxy-r95xn                       1/1     Running   7 (16m ago)      3d1h
kube-proxy-twlpp                       1/1     Running   5 (80m ago)      3d1h
kube-scheduler-node-master             1/1     Running   19 (80m ago)     3d1h
logstash-deployment-8ffbcc994-l4rbb    1/1     Running   0                8m1s
```

Figura 57: Obtención de pods en el nombre de espacio kube-system.

En este paso de instalación queda pendiente la instalación de un agente que recoja logs y envíe a Logstash y la herramienta Kibana para poder acceder a los logs de forma visual. En los apartados siguientes, se instalan el resto de las herramientas.

4.4.1.3.3 Instalación Filebeat

En este apartado, se configura Filebeat. Se trata del agente que se va a utilizar para enviar registros a Logstash. Para su implementación, se utilizará un objeto de tipo DaemonSet ya que se desea que haya una instancia en cada nodo del clúster. Además, este objeto contendrá permisos para acceder, monitorizar y listar pods dentro de cada nodo (ver figura 58).

```
kind: DaemonSet
metadata:
  name: filebeat
  namespace: kube-system
  labels:
    k8s-app: filebeat
spec:
  selector:
    matchLabels:
      k8s-app: filebeat
  template:
    metadata:
      labels:
        k8s-app: filebeat

    spec:
      serviceAccountName: filebeat
      terminationGracePeriodSeconds: 30
      containers:
      - name: filebeat
        image: docker.elastic.co/beats/filebeat:6.8.4
        args: [
          "-c", "/etc/filebeat.yml",
          "-e",
        ]
```

Figura 58:DaemonSet de Filebeat.

Se puede observar el uso de la imagen oficial de Filebeat con la versión 6.8.4 que recibe como argumento el fichero de configuración "/etc/filebeat.yml".

A continuación, se crea la cuenta de servicio, se aplican los permisos y los pods definidos en DaemonSet.

4.4.1.3.4 Instalación Kibana

En este apartado, se describe la instalación de Kibana.

Kibana es la interfaz de usuario para visualizar logs. Los logs se obtienen realizando consultas al servicio de Elasticsearch. En la figura 59 se puede ver la configuración de Kibana: En primer lugar, se aprecia que se usará una única réplica que ejecutará la imagen de kibana con la versión 7.2.0. Además, se indica el límite de los recursos para permitir escalar el pod de kibana (en caso de ser necesario). También se indica el puerto de escucha de Elasticsearch por medio de variable de entorno "ELASTICSEARCH_URL". Los pods de este despliegue escucharán en el puerto 5601.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kibana
  # namespace: efk-logging
  labels:
    app: kibana
spec:
  replicas: 1
  selector:
    matchLabels:
      app: kibana
  template:
    metadata:
      labels:
        app: kibana
    spec:
      containers:
      - name: kibana
        image: docker.elastic.co/kibana/kibana:7.2.0
        resources:
          limits:
            cpu: 1000m
          requests:
            cpu: 100m
        env:
          - name: ELASTICSEARCH_URL
            value: http://elasticsearch:9200
        ports:
        - containerPort: 5601
```

Figura 59: Declaración de Deployment para Kibana.

A continuación, se declara el servicio de tipo NodePort para permitir el acceso a la interfaz gráfica de Kibana desde fuera del clúster en el puerto 32010 (ver figura 60).

```
---
apiVersion: v1
kind: Service
metadata:
  name: kibana-logging
  namespace: kube-system
  labels:
    k8s-app: kibana-logging
    kubernetes.io/name: "Kibana"
spec:
  type: NodePort
  ports:
  - port: 5601
    protocol: TCP
    targetPort: ui
    nodePort: 32010
  selector:
    k8s-app: kibana-logging
```

Figura 60: Creación de servicio para Kibana.

Finalmente, se aplica el comando para crear la aplicación y el servicio (ver figura 61).

```
[root@node-master kibana]# kubectl apply -f deployment.yaml
deployment.apps/kibana-logging created
service/kibana-logging created
```

Figura 61: Aplicación del comando kubectl.

4.4.1.3.5 Resultado final

Finalmente, tras haber desplegado todos las herramientas, se puede consultar los pods corriendo en el nombre de espacio del sistema como se aprecia en la figura 62.

```
[root@node-master kibana]# kubectl get pods -n kube-system
NAME                                READY   STATUS             RESTARTS         AGE
calico-node-bv5wf                   1/1     Running            13 (28m ago)     3d19h
calico-node-jl2dj                   1/1     Running            9 (55m ago)      3d19h
calico-node-zdn2x                   1/1     Running            8 (18h ago)      3d19h
coredns-666959ff67-t4cqt            1/1     Running            7 (2m33s ago)    48m
coredns-666959ff67-wnpt8            1/1     Running            68 (16m ago)     3d19h
dns-autoscaler-59b8867c86-cr5ds     1/1     Running            0                48m
elasticsearch-logging-0             1/1     Running            0                27m
elasticsearch-logging-1             0/1     CrashLoopBackOff   3 (23s ago)      104s
filebeat-7xbhr                      1/1     Running            1 (18h ago)      18h
filebeat-s6rfr                      1/1     Running            1 (55m ago)      18h
filebeat-xt7k7                      1/1     Running            1 (28m ago)      18h
kibana-logging-7b6d8cbc6b-r226q     1/1     Running            0                52s
kube-apiserver-node-master          1/1     Running            12 (42m ago)     3d20h
kube-controller-manager-node-master 1/1     Running            36 (44m ago)     3d20h
kube-proxy-bb99w                    1/1     Running            9 (37m ago)      3d20h
kube-proxy-r95xn                    1/1     Running            9 (28m ago)      3d19h
kube-proxy-twlpp                    1/1     Running            6 (55m ago)      3d20h
kube-scheduler-node-master          1/1     Running            24 (44m ago)     3d20h
logstash-deployment-8ffbcc994-vsfjz 1/1     Running            0                48m
```

Figura 62: Ejecución de los pods en el nombre de espacio kube-system.

Aparte de los componentes del sistema explicados en los anteriores apartados, se puede ver la aparición de nuevos pods ejecutando sobre el nombre de espacio “kube-system”: pods de Filebeat: “**filebeat-***” que se ejecutan en cada nodo del clúster que van a recoger métricas y enviarlas a Logstash. Por otra parte, se puede observar el pod de Kibana: “**kibana-logging-***” que permitirá acceder a los logs y realizar filtrados y consultas sobre los mismos.

Tras ello, se accede a la dirección del nodo que está ejecutando Kibana en el puerto 32010 y se puede observar la interfaz gráfica de Kibana (ver figura 63).

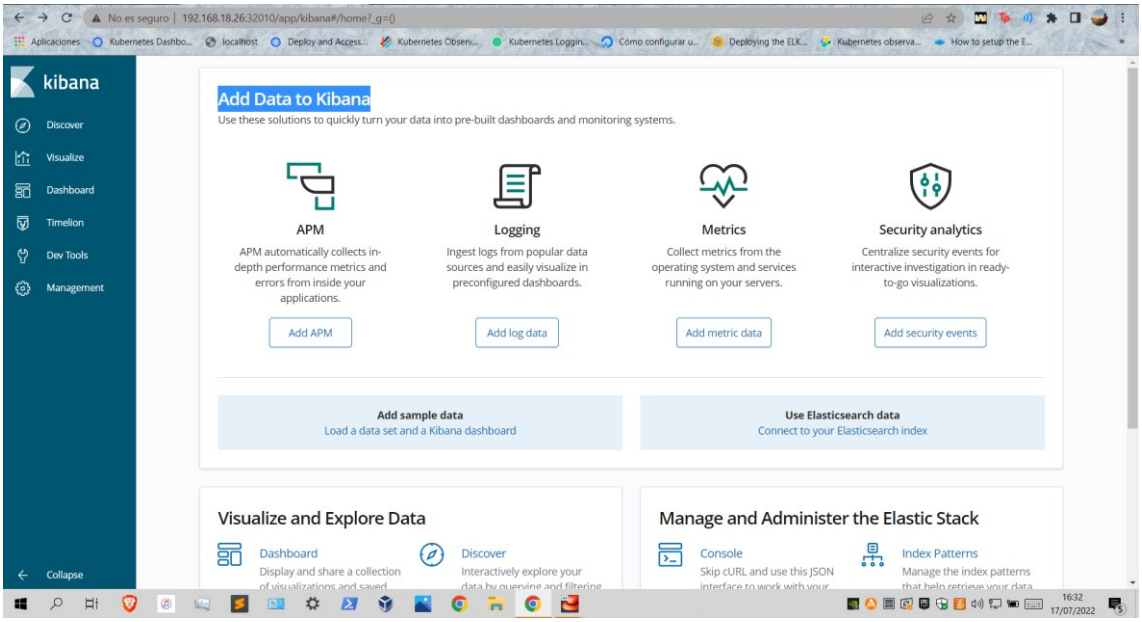

Figura 63: Cuadro de mando principal de Kibana.

En el marco izquierdo de la figura 64 se pueden ver las funcionalidades disponibles como ver logs: cuadro de mando (Dashboard) para realizar consultas y aplicar filtros , consultar a la api de Elasticsearch (Dev Tools) y un apartado para gestionar configuración sobre los índices de búsqueda de los logs (Management)

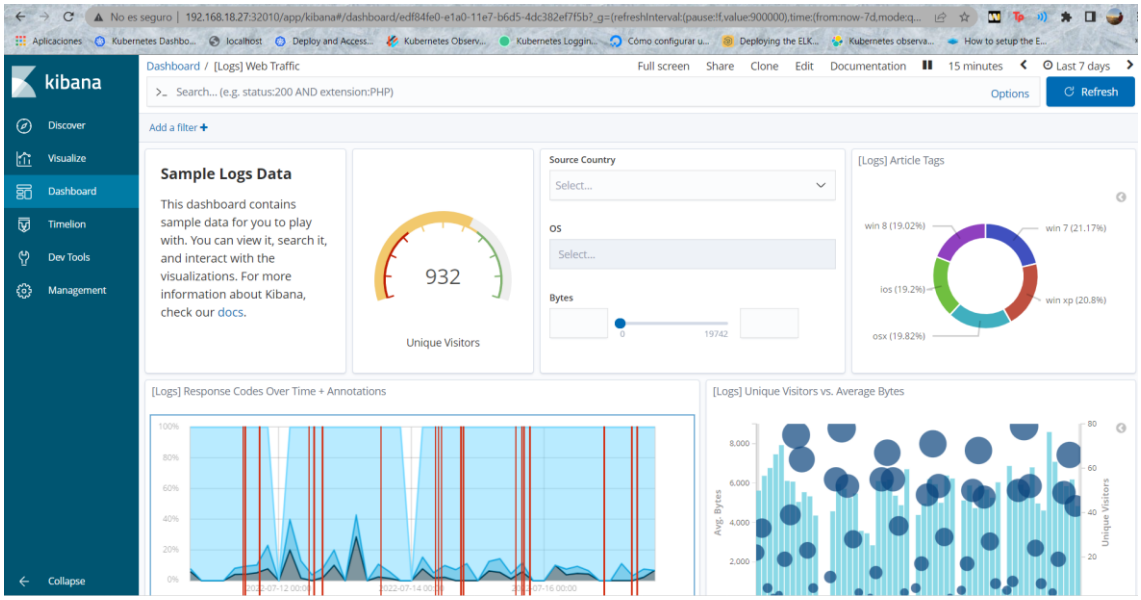

Figura 64: Cuadro de mando para visualizar logs de Kibana.

Tras la generación de un proyecto de ejemplo, se aprecia en el apartado de cuadro de mando la posibilidad de aplicar filtrar y seleccionar logs por diferentes métricas como agente, clienteip, contenido del mensaje entre otros. Los logs se pueden organizar por diferentes tipos de diagramas: circulares, barras, histogramas como se ve en la figura 65.

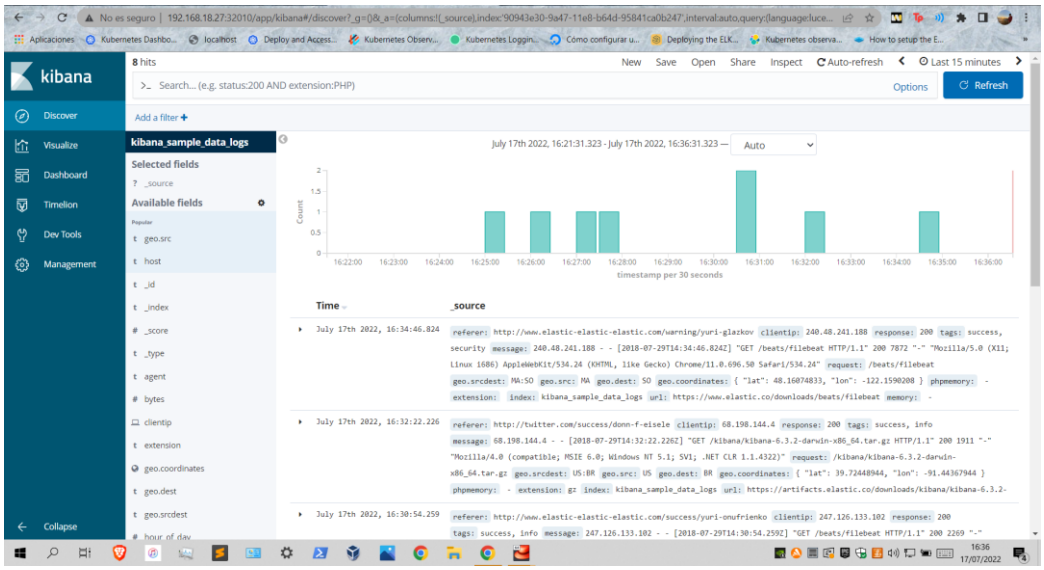

Figura 65: Obtención de logs de un proyecto de ejemplo.

Si se desea analizar logs en el tiempo, se puede aplicar filtros de fecha, horas, minutos o desde una fecha concreta.

Además, se puede acceder al apartado de Dev Tools para realizar consultas a la base de datos de Elasticsearch como se ve en la figura 66.

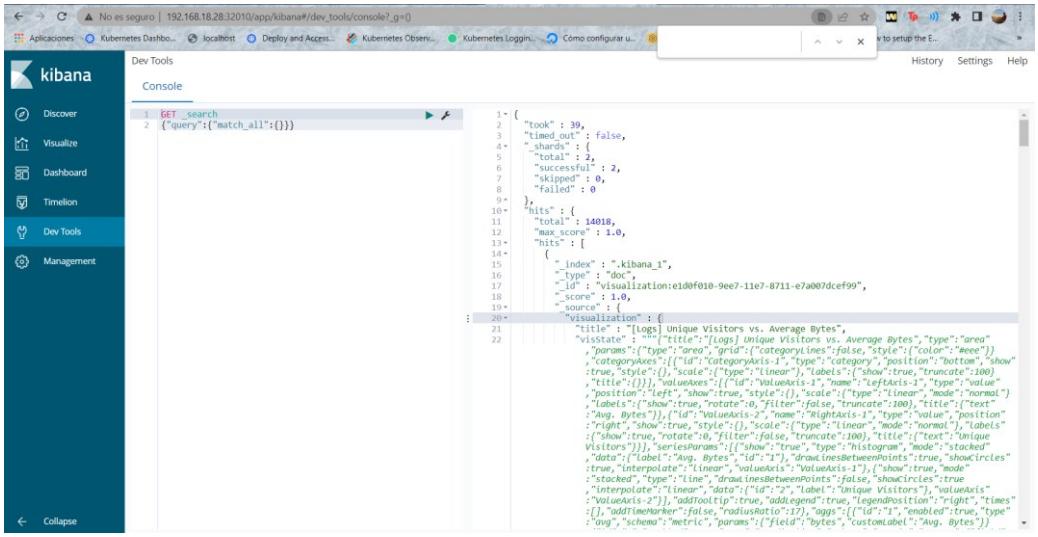

Figura 66: Consulta a la base de datos de Elasticsearch desde Kibana.

Finalmente, también se pueden escuchar logs de la propia interfaz de Kibana. Para ello, en el apartado de gestión, se selecciona el índice de "kibana" para monitorizar sus logs (ver figura 67).

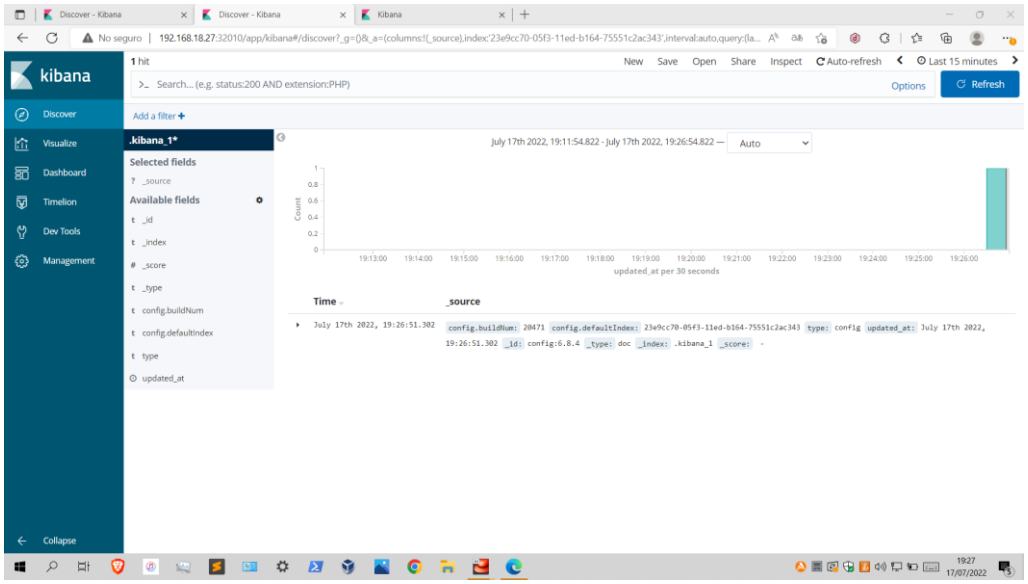

Figura 67: Obtención de logs de Kibana.

4.4.2 Despliegue en Google Cloud

En este apartado, se realizará el despliegue de un clúster Kubernetes en Google Cloud y se estudiará la monitorización de logs.

4.4.2.1 Configuración de los nodos del clúster

En este apartado, se configurarán las máquinas virtuales que formarán el clúster de Kubernetes.

Se necesita al igual que en el despliegue local, un total de 4 nodos: dos nodos trabajadores, un nodo máster y una máquina adicional con ansible que instalará Kubernetes (ver tabla 3 para las especificaciones).

Tipo	S.O
Nodo-ansible-cloud	Rocky Linux 8 x86-64, x86_64 con 2 CPU y 8 Gb RAM y 50Gb ROM persistente.
Nodo-master-cloud	Rocky Linux 8 x86-64, x86_64 con 2 CPU y 8 Gb RAM y 50Gb ROM persistente.
Nodo-worker-cloud	Rocky Linux 8 x86-64, x86_64 con 2 CPU y 8 Gb RAM y 50Gb ROM persistente.
Nodo-worker2-cloud	Rocky Linux 8 x86-64, x86_64 con 2 CPU y 8 Gb RAM y 50Gb ROM persistente.

Tabla 3: Configuración de los nodos que formarán el clúster.

En la tabla 3 se puede observar que se trabajará sobre el sistema operativo Rocky Linux (disponible en Google Cloud) , contarán con 2 CPU, 8 Gb de RAM y 50 Gb de disco duro.

En primer lugar, se accederá a la consola web [15] para crear las máquinas virtuales y se hace clic sobre crear una VM (Virtual Machine), como se aprecia en la figura 68.

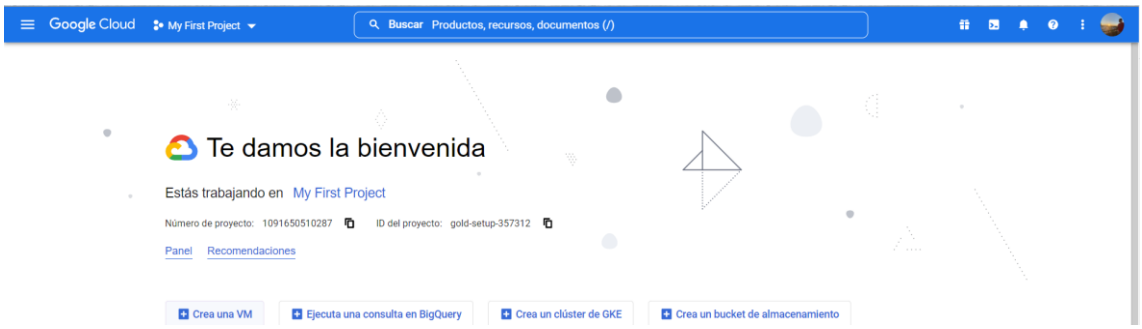

Figura 68: Consola principal.

Se seleccionará el nombre así como la región donde se alojará la máquina virtual como se aprecia en la figura 69.

[15] https://console.cloud.google.com/welcome

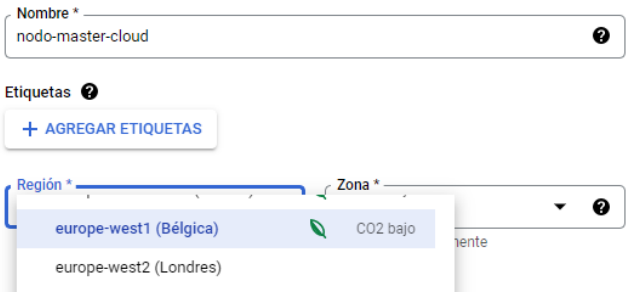

Figura 69: Metadatos de creación de máquina virtual en la nube.

Se ha seleccionado una zona europea con índices bajos de contaminación de dióxido de carbono (Bélgica).

Se pueden configurar la especificación físicas del nodo, como Cores, memoria RAM, si es necesario optimización de memoria para cálculos computacionales complejos, así como el uso de una GPU. Estos aspectos se configuran en el apartado de configuración en la figura 70.

Figura 70: Configuración de la máquina virtual.

Se ha optado por la serie N2 de 2 CPU y 8Gb de RAM, que se ajustan perfectamente al desarrollo de este proyecto.

En la configuración de la figura 71, se selecciona el disco de arranque, así como el sistema operativo.

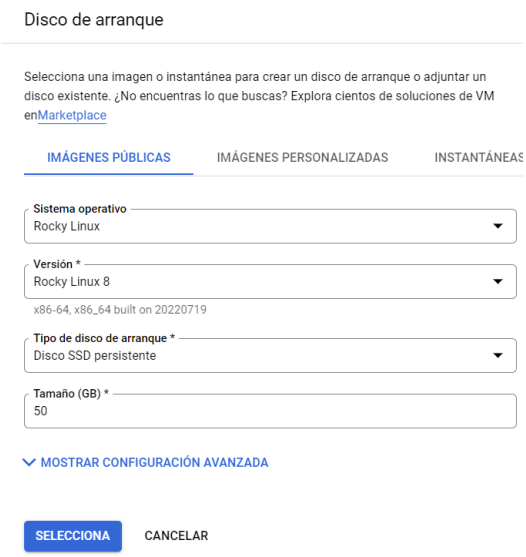

Figura 71: Selección de S.O y disco.

Se elige el sistema operativo Rocky Linux 8 con un disco persistente SSD de 50Gb. A continuación, en la figura 72, se configura el acceso y las cuentas de servicio de los nodos.

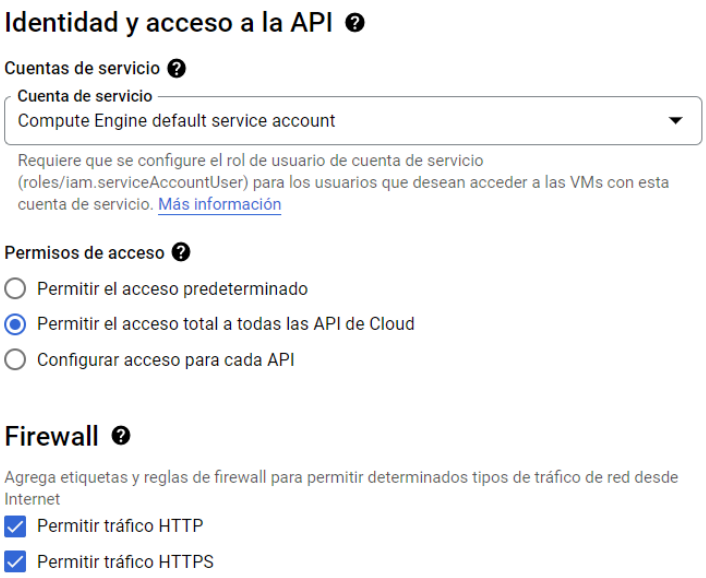

Figura 72: Configuración de las cuentas de servicio.

Se permite el acceso a toda la API de Cloud mediante la cuenta de servicio por defecto, además de permitir el tráfico HTTP y HTTPS, con esto se habilita el tráfico en el puerto 80 y 443,sin embargo, el clúster funcionará en otros puertos, por lo que se tendrán que añadir nuevas reglas para comunicar los nodos en diferentes puertos.

Una vez que se ha configurado el nodo, Google ofrece la posibilidad de crear máquinas virtuales mediante línea de comandos en la Shell de Google Cloud (ver figura 73).

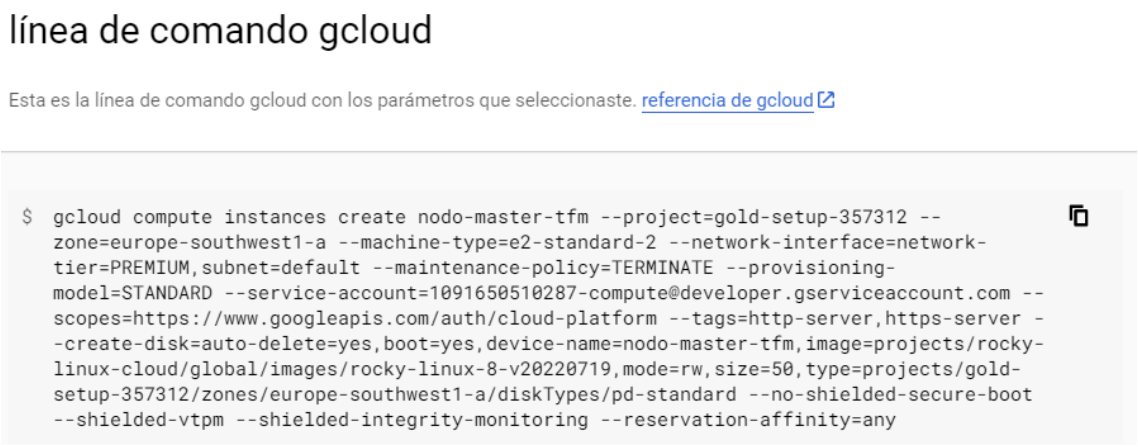

Figura 73: Configuración del nodo-master vía línea de comandos.

Se utiliza el comando de la figura 73 para crear la máquina virtual. Finalmente, se puede ver en la figura 74 la máquina virtual creada correctamente.

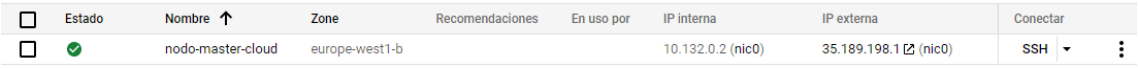

Figura 74: Máquina virtual creada correctamente.

Una vez creada la máquina virtual, esta ofrece una IP externa al cual se puede conectar vía SSH desde cualquier host para interoperar con ella (penúltima columna en la figura 74).

Se repite la misma configuración para el resto de los nodos, finalmente se quedan creados todos los máquinas virtuales (ver figura 75).

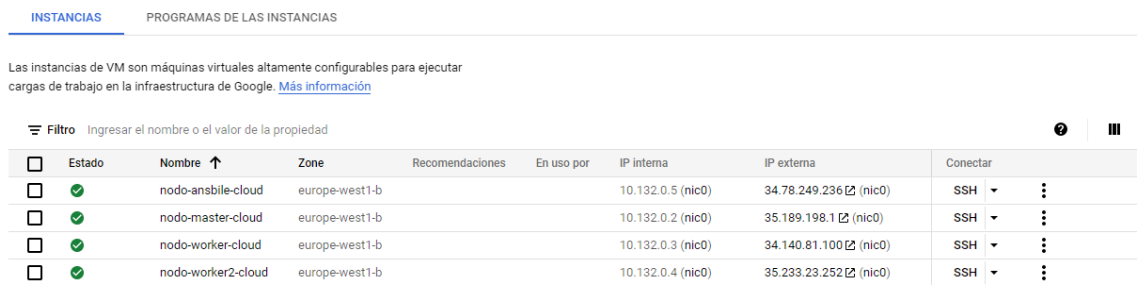

Figura 75: Creación correcta de las máquinas virtuales del clúster en Google.

Se puede observar que todas las máquinas virtuales ofrecen una IP externa volátil, esto quiere decir que si se apaga una máquina virtual, la próxima vez no se asigna la misma IP. Esto es un problema ya que es necesario definir una IP y que no se modifique en el tiempo para que el nodo

máster pueda encontrar los nodos trabajadores en la red. En los apartados siguientes, se procede a resolver los problemas de acceso y la persistencia de IP.

4.4.2.2 Acceso las máquinas virtuales vía SSH

En este apartado, se describe la forma de acceder a las máquinas virtuales en la nube.

En primer lugar, para acceder, se usará par de claves pública/privada en el host y se conectará vía SSH mediante la clave privada (sin usar contraseña).

Se crean las claves en el host, que en este caso se trate de Windows 10. Tras generar las claves, se copiará la clave pública en los metadatos del proyecto en Google Cloud, tal y como se aprecia en la figura 76.

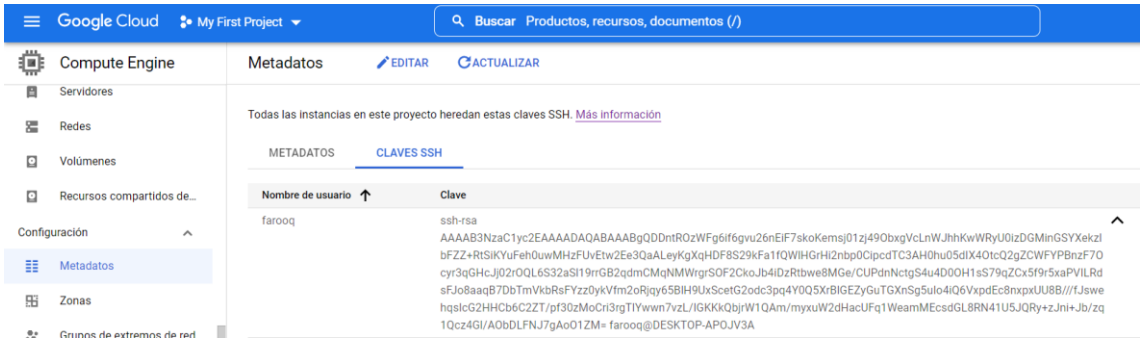

Figura 76: Copiando clave pública en metadatos del proyecto.

De tal forma que mediante la clave privada almacenada en el host, se puede acceder a las máquinas virtuales configuradas bajo el mismo proyecto de la manera siguiente (ver figura 77).

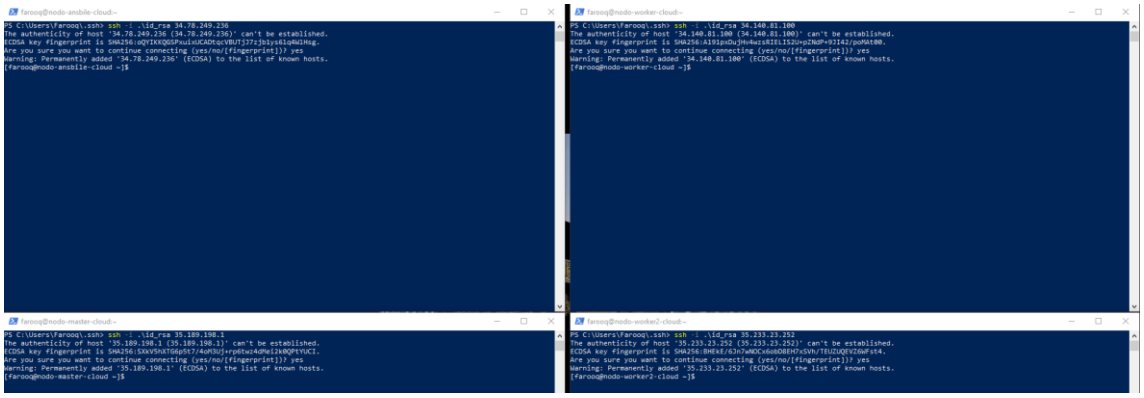

Figura 77: Acceso a las máquinas virtuales vía SSH.

Por defecto, no viene configurada la contraseña para el usuario root, se configura mediante los comandos de la figura 78.

```
[farooq@nodo-worker-cloud ~]$ sudo passwd
Changing password for user root.
New password:
BAD PASSWORD: The password is shorter than 8 characters
Retype new password:
passwd: all authentication tokens updated successfully.
[farooq@nodo-worker-cloud ~]$ su
Password:
[root@nodo-worker-cloud farooq]# clear
```

Figura 78: Configuración de la contraseña para el usuario root.

4.4.2.3 Persistencia de IP en Google Cloud

En este apartado, se configuran las redes para que no muten cada vez que se apaga la máquina virtual. Se reservan las direcciones IP de las máquinas virtuales, ya que al apagar las máquinas, se resetearán y cada vez que se inicie la máquina virtual, Google asignará una IP nueva. Para evitar esto, se accede a la sección de redes en este enlace[16].

A continuación se reservan IP cambiando el nivel de red (ver figura 79).

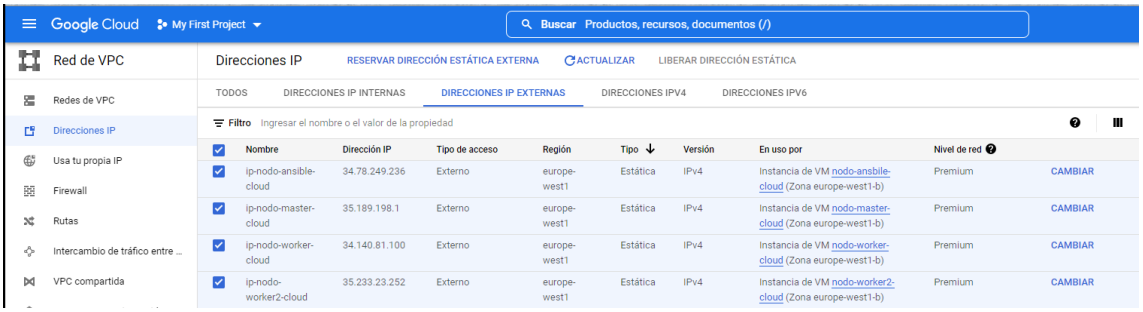

Figura 79: Reserva de IP.

Como se puede apreciar en la figura 79, tras hace clic sobre la última columna de cada IP se puede cambiar a nivel de red premium. Ahora, las direcciones IP son de tipo IPV4 estáticas.

4.4.2.4 Instalación del clúster Kubernetes

Una vez, creadas las máquinas virtuales y configurado el accesos a estas, en este apartado, se configuran para recibir la instalación de Kubernetes. Para ello, se siguen los mismos pasos que para el despliegue local: se deshabilita firewall, el espacio de intercambio y se reinicia el nodo para los nodos máster, worker y worker2. Se puede observar la pila de comandos a realizar en la figura 80.

[16] https://console.cloud.google.com/networking/addresses/

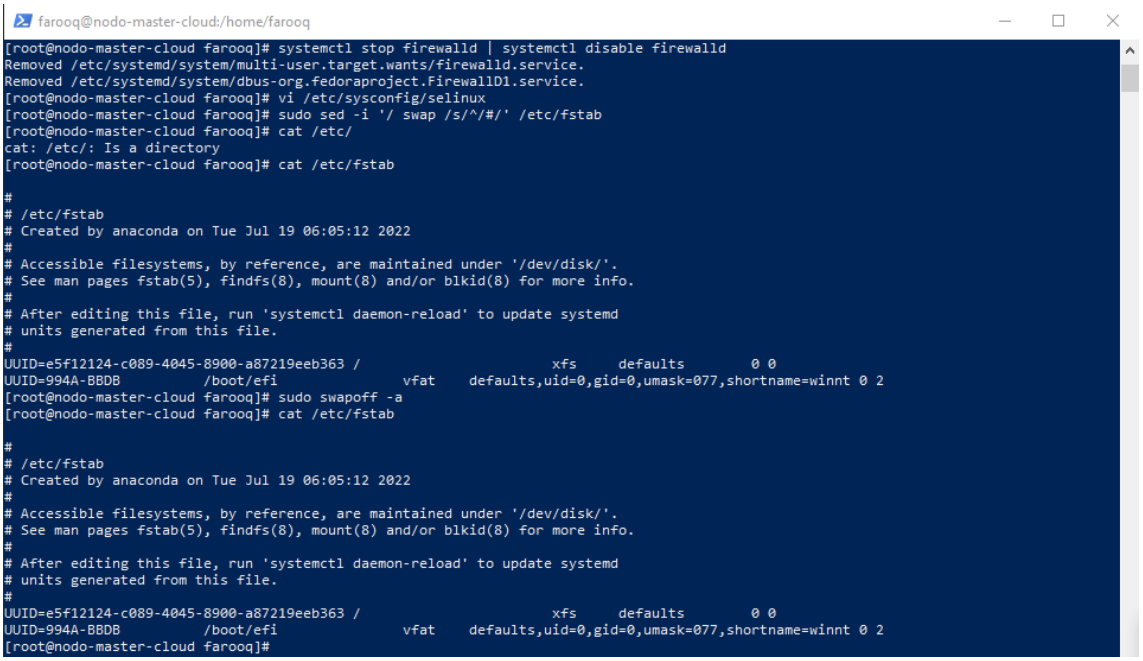

Figura 80: Preparación de los nodos máster, worker y worker2.

En futuros pasos, se prepara la máquina ansible.

En primer lugar, se crea un par de claves pública/privada (ver figura 81).

```
[root@nodo-ansbile-cloud farooq]# ssh-keygen -q
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

Figura 81: Creación de clave pública/privada.

En segundo lugar, se copia la clave pública a los nodos máster, worker y worker2 (ver figura 82).

```
[root@nodo-worker-cloud farooq]# mkdir /root/.ssh
[root@nodo-worker-cloud farooq]# vi /root/.ssh/authorized_keys
[root@nodo-worker-cloud farooq]# chmod 600 /root/.ssh/authorized_keys
[root@nodo-worker-cloud farooq]# cat /root/.ssh/authorized_keys
sh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQC+JTeIiO7DVFFg3Oe/Og1uBr1ke2d8RUfZaU191b6ddi7QCXJD87misIV6qVF1f5wGJrRuWZWmoFOPURPOa2r/B5Rw0vWC5BE
Ydg3U+5xJnw8ApXNxtOGDCVy25DkBw7z6dLyt09kPUa7gEpMT+DL4BOJQUZXTfbihErz3afihAww8xZgypyv2pzf6u3kuu5GOu4qC5I2pdLuYqNfUUWLGg19xGsKqeP4SdCEPD
5COn4ow9eaEgTp4EKH5ptt8SkxxYtzCSS1uKyeveo3zSgcFprL2L31rKDPloaVFZspg8qWXvB6wSxqvMY4Y+E+xXT+0HYJEY/24EbWmaUYi8b05G7QxqV0cn2IqUb2eIgyDd/d
G6GjPZyqtkoHV+Y5dCIEsxZPGRK+bA2BXm1JngrJVIuQiE2TpSJR8Q1x1d+WZdj3cH+3aGUlYLLDoJUCoO1LMqO7dDL/55/xwGs9ACRgPdnx491tuvGjEYBBmvD9shahkHc81g
J2lFkjBFaLl5/c= root@nodo-ansbile-cloud
[root@nodo-worker-cloud farooq]#
```

Figura 82: Copiando clave pública en los nodos del clúster.

Por defecto, la configuración del SSH de las máquinas virtuales alojadas en la nube no permiten el acceso como usuario root, por tanto, se modifican los permisos en el fichero de configuración de SSH: /etc/ssh/sshd_config (ver figura 83).

```
# Authentication:

#LoginGraceTime 2m
PermitRootLogin yes
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

PubkeyAuthentication yes
```

Figura 83: Alterando permisos con usuario root.

Posteriormente, se reinicia el servicio SSH (ver figura 84).

```
[root@nodo-worker2-cloud farooq]# systemctl restart sshd
```

Figura 84: Reiniciando servicio SSH.

Ahora, se añaden reglas en el fichero /etc/hosts y se prueba que se tiene acceso vía SSH sin contraseña a las máquinas que formarán el clúster (ver figura 85).

```
[root@nodo-ansbile-cloud .ssh]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
10.132.0.5 nodo-ansbile-cloud.europe-west1-b.c.gold-setup-357312.internal nodo-ansbile-cloud  # Added by Google
169.254.169.254 metadata.google.internal  # Added by Google
35.189.198.1 nodo-master-cloud
34.140.81.100 nodo-worker-cloud
35.233.23.252 nodo-worker2-cloud
[root@nodo-ansbile-cloud .ssh]# ssh nodo-master-cloud
Last login: Mon Jul 25 20:36:42 2022 from 34.78.249.236
[root@nodo-master-cloud ~]# exit
logout
Connection to nodo-master-cloud closed.
[root@nodo-ansbile-cloud .ssh]# ssh nodo-worker-cloud
Last login: Mon Jul 25 20:37:10 2022 from 34.78.249.236
[root@nodo-worker-cloud ~]# exit
logout
Connection to nodo-worker-cloud closed.
[root@nodo-ansbile-cloud .ssh]# ssh nodo-worker2-cloud
The authenticity of host 'nodo-worker2-cloud (35.233.23.252)' can't be established.
ECDSA key fingerprint is SHA256:BHEkE/6Jn7wNOCx6obD8EH7xSVh/TEUZUQEVZ6WFst4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'nodo-worker2-cloud,35.233.23.252' (ECDSA) to the list of known hosts.
Last login: Mon Jul 25 19:55:05 2022
[root@nodo-worker2-cloud ~]# exit
logout
Connection to nodo-worker2-cloud closed.
[root@nodo-ansbile-cloud .ssh]#
```

Figura 85: Comprobación de acceso a los nodos del clúster desde el nodo ansible.

Posteriormente, se realiza la instalación de pip y pyhton3 en el nodo ansible (ver figura 86).

```
[root@nodo-ansbile-cloud farooq]# python3 -V
Python 3.6.8
[root@nodo-ansbile-cloud farooq]# pip -V
pip 21.3.1 from /usr/local/lib/python3.6/site-packages/pip (python 3.6)
[root@nodo-ansbile-cloud farooq]#
```

Figura 86: Comprobación de versiones de pip y Python.

A continuación, se configura el fichero del inventario donde se le indica las direcciones IP de las máquinas que formarán el clúster, roles de los nodos y la función de cada uno de los nodos dentro del clúster (ver figura 87).

```
[root@nodo-ansbile-cloud kubespray]# cat inventory/cloudcluster/inventory.ini
# ## Configure 'ip' variable to bind kubernetes services on a
# ## different ip than the default iface
# ## We should set etcd_member_name for etcd cluster. The node that is not a etcd member do not need to set the value, or can set the
 empty string value.
[all]
nodo-master-cloud ansible_host=35.189.198.1
nodo-worker-cloud ansible_host=34.140.81.100
nodo-worker2-cloud ansible_host=35.233.23.252


# ## configure a bastion host if your nodes are not directly reachable
# [bastion]
# bastion ansible_host=x.x.x.x ansible_user=some_user

[kube_control_plane]
nodo-master-cloud

[etcd]
nodo-master-cloud

[kube_node]
nodo-master-cloud
nodo-worker-cloud
nodo-worker2-cloud

[calico_rr]

[k8s_cluster:children]
kube_control_plane
kube_node
calico_rr
```

Figura 87: Creación del fichero del inventario.

La versión de ansible instalada es 2.11.12 como se aprecia en la figura 88.

```
[root@nodo-ansbile-cloud cloudcluster]# ansible --version



/usr/local/lib/python3.6/site-packages/ansible/parsing/vault/__init__.py:44: CryptographyDeprecationWarning: Python 3.6 is no longer s
upported by the Python core team. Therefore, support for it is deprecated in cryptography and will be removed in a future release.
  from cryptography.exceptions import InvalidSignature
ansible [core 2.11.12]
```

Figura 88: Comprobación de la versión de ansible.

Se lanza el comando ping a los nodos que formarán el clúster (ver figura 89).

```
[root@nodo-ansbile-cloud kubespray]# ansible -i inventory/cloudcluster/inventory.ini -m ping all
[WARNING]: Skipping callback plugin 'ara_default', unable to load
nodo-worker2-cloud | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
nodo-worker-cloud | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
nodo-master-cloud | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Figura 89: Respuesta correcta del ping a los nodos del clúster.

Finalmente, se lanza el comando playbook para instalar el clúster Kubernetes sobre los nodos (ver figura 90).

```
[root@nodo-ansbile-cloud kubespray]# ansible-playbook -i inventory/cloudcluster/inventory.ini --become-user=root cluster.yml
```

Figura 90: Lanzamiento del comando ansible-playbook.

El resumen del comando anterior se puede observar en la figura 91.

```
PLAY RECAP *********************************************************************************************************************
localhost                  : ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
nodo-master-cloud          : ok=778  changed=147  unreachable=0    failed=0    skipped=1252 rescued=0    ignored=9
nodo-worker-cloud          : ok=508  changed=95   unreachable=0    failed=0    skipped=730  rescued=0    ignored=2
nodo-worker2-cloud         : ok=508  changed=95   unreachable=0    failed=0    skipped=730  rescued=0    ignored=2

Monday 25 July 2022  21:15:30 +0000 (0:00:00.132)       0:12:55.953 **********
===============================================================================
kubernetes/preinstall : Install packages requirements ------------------------------------------------------------------- 62.37s
kubernetes/kubeadm : Join to cluster ------------------------------------------------------------------------------------ 38.41s
download : download_file | Validate mirrors ----------------------------------------------------------------------------- 23.57s
kubernetes/control-plane : kubeadm | Initialize first master ----------------------------------------------------------- 14.22s
kubernetes-apps/ansible : Kubernetes Apps | Start Resources ------------------------------------------------------------- 9.73s
etcd : reload etcd ------------------------------------------------------------------------------------------------------ 8.64s
kubernetes/preinstall : Preinstall | wait for the apiserver to be running ----------------------------------------------- 8.27s
kubernetes-apps/ansible : Kubernetes Apps | Lay Down CoreDNS templates -------------------------------------------------- 8.18s
download : download_container | Download image if required -------------------------------------------------------------- 7.56s
download : download_container | Download image if required -------------------------------------------------------------- 7.29s
network_plugin/calico : Wait for calico kubeconfig to be created -------------------------------------------------------- 6.61s
download : download_container | Download image if required -------------------------------------------------------------- 5.50s
etcd : Configure | Check if etcd cluster is healthy --------------------------------------------------------------------- 5.49s
network_plugin/calico : Start Calico resources -------------------------------------------------------------------------- 5.08s
network_plugin/calico : Calico | Create calico manifests ---------------------------------------------------------------- 4.91s
download : download_container | Download image if required -------------------------------------------------------------- 4.89s
container-engine/containerd : containerd | Unpack containerd archive ---------------------------------------------------- 4.73s
download : download_container | Download image if required -------------------------------------------------------------- 4.62s
download : download_container | Download image if required -------------------------------------------------------------- 4.35s
network_plugin/calico : Get current calico cluster version -------------------------------------------------------------- 4.35s
```

Figura 91: Resumen de la instalación del clúster.

Debido a 2 CPU y más memoria RAM, el tiempo de instalación ha sido muy inferior a la solución en local. Se ha tardado un total de 12m55s. Como se apreciar, todas las tareas han ido bien, ninguna ha provocado error. Ahora, se procede a comprobar la correcta instalación mediante obtención de nodos y pods del nombre de espacio “kube-system” del clúster (ver figura 92 y 93).

```
[root@nodo-master-cloud farooq]# kubectl get nodes
NAME                STATUS   ROLES           AGE     VERSION
nodo-master-cloud   Ready    control-plane   6m32s   v1.24.3
nodo-worker-cloud   Ready    <none>          5m16s   v1.24.3
nodo-worker2-cloud  Ready    <none>          5m16s   v1.24.3
[root@nodo-master-cloud farooq]#
```

Figura 92: Comprobando nodos del clúster.

```
[root@nodo-master-cloud farooq]# kubectl get pods -n kube-system
NAME                                        READY   STATUS    RESTARTS   AGE
calico-node-472dj                           1/1     Running   0          5m11s
calico-node-618vm                           1/1     Running   0          5m11s
calico-node-nbfkr                           1/1     Running   0          5m11s
coredns-74d6c5659f-4gdwv                    1/1     Running   0          4m27s
coredns-74d6c5659f-xc4kn                    1/1     Running   0          4m20s
dns-autoscaler-59b8867c86-h6jj2             1/1     Running   0          4m22s
kube-apiserver-nodo-master-cloud            1/1     Running   1          6m56s
kube-controller-manager-nodo-master-cloud   1/1     Running   1          6m54s
kube-proxy-6sdz4                            1/1     Running   0          5m38s
kube-proxy-9zplh                            1/1     Running   0          5m38s
kube-proxy-bft5m                            1/1     Running   0          5m38s
kube-scheduler-nodo-master-cloud            1/1     Running   1          6m54s
nginx-proxy-nodo-worker-cloud               1/1     Running   0          4m52s
nginx-proxy-nodo-worker2-cloud              1/1     Running   0          4m51s
nodelocaldns-kjt48                          1/1     Running   0          4m20s
nodelocaldns-qcmhl                          1/1     Running   0          4m20s
nodelocaldns-trsf7                          1/1     Running   0          4m20s
```

Figura 93: Comprobando pods del clúster.

4.4.2.5 Obtención de logs de las máquinas virtuales en Google

En este apartado, se analizarán formas de obtener logs de un clúster ejecutándose dentro de las máquinas virtuales de Google.

Por una parte, se puede acceder a los logs de los nodos en el apartado métricas y filtrar logs de cada nodo (véase figura 94).

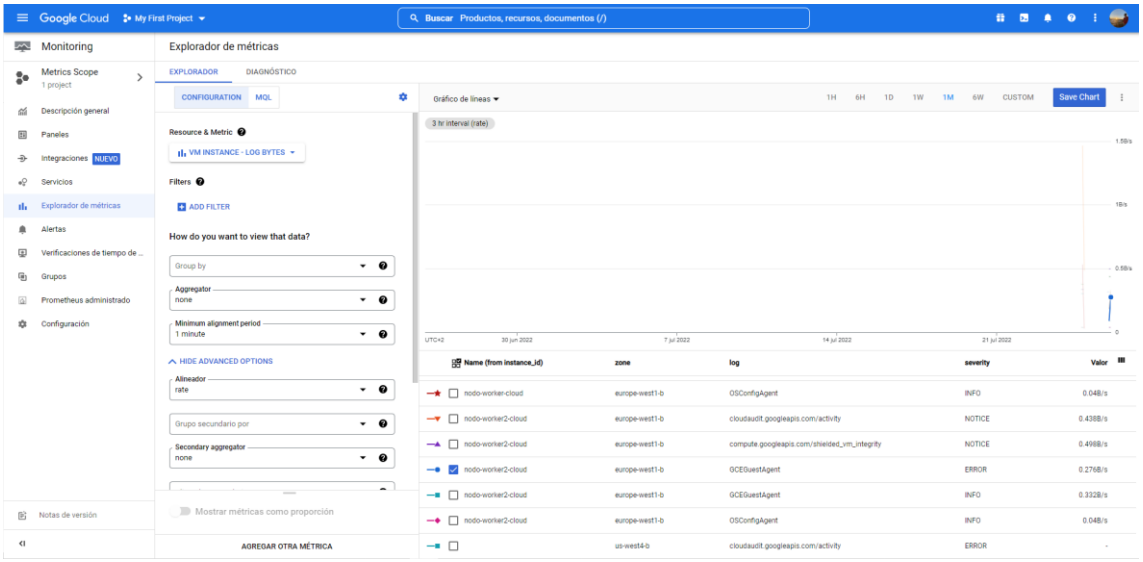

Figura 94: Observación de logs.

Todos los logs de las máquinas virtuales se registran de forma automática en el registro por defecto, sin embargo, los logs de los contenedores dentro de las máquinas virtuales no son reflejados en este apartado.

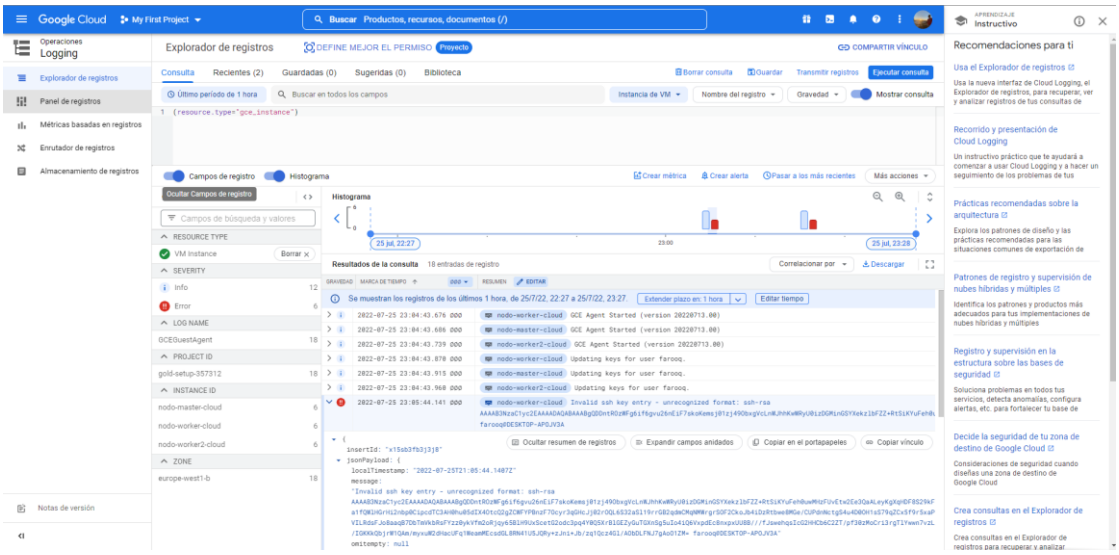

Figura 95: Fallo al realizar comunicación SSH a la máquina virtual.

Como se aprecia en la figura 95, se pueden ver logs relativos a nivel de nodo como accesos erróneos al realizar SSH a la máquina virtual.

Sin embargo, no es posible detectar la actividad que se está ejecutando dentro del clúster, analizada la limitación, se procede a instalar la pila EFK.

4.4.2.6 EFK

La instalación de la pila EFK es idéntica a la realizada en el apartado local excepto que en lugar de Logstash se utilizará Fluentd. A continuación, se analizan los resultados tras crear los despliegues de Elasticsearch, Fluentd, Filebeat y Kibana.

Para esta instalación, todos los objetos se han creado en el nombre de espacio por defecto. En la figura 96, se pueden observar el objeto deployment necesario para crear el pod de kibana.

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl get deployment
NAME     READY   UP-TO-DATE   AVAILABLE   AGE
kibana   1/1     1            1           27m
```

Figura 96: Obtención de deployment ejecutando en el clúster.

En la figura 97, se pueden observar el objeto StatefulSet necesario para crear los pods de Elasticsearch, uno por cada nodo para dotar de alta disponibilidad antes fallos de alguno de los pods. Los 3 pods de Elasticsearch se levantarán sobre el mismo volumen persistente.

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl get statefulset
NAME         READY   AGE
es-cluster   3/3     28m
```

Figura 97: Obtención de StatefulSet ejecutando en el clúster.

En la figura 98, se puede observar el objeto DaemonSet necesario para crear pods de fluentd.

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl get daemonset
NAME      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
fluentd   3         3         3       3            3           <none>          29m
```

Figura 98: Obtención de DaemonSet ejecutando en el clúster.

A continuación, se ven los pods ejecutando dentro del clúster (ver figura 99).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
counter-es                1/1     Running   0          25m
es-cluster-0              1/1     Running   0          26m
es-cluster-1              1/1     Running   0          25m
es-cluster-2              1/1     Running   0          24m
fluentd-v5bjn             1/1     Running   0          25m
fluentd-wdrfb             1/1     Running   0          25m
fluentd-wr2s9             1/1     Running   0          25m
kibana-84cf7f59c-kjk5j    1/1     Running   0          26m
```

Figura 99: Obtención de pods ejecutando en el clúster.

En la figura 99, se puede observar el pod “counter-es” que imprime un mensaje en español que servirá para realizar pruebas. Los pods “es-cluster-x” (0,1,2) son aplicaciones de Elasticsearch “**es-cluster-***” ejecutándose en cada nodo del clúster. Además, también se encuentran pods de Fluentd “**fluentd-***” ejecutando en cada nodo. En el último lugar, se puede observar un pod de kibana “**kibana-***”.

A continuación, se procede a obtener servicios ejecutando dentro del clúster (ver figura 100).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl get svc
NAME            TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)              AGE
elasticsearch   ClusterIP   None           <none>        9200/TCP,9300/TCP    27m
kibana          ClusterIP   10.68.10.167   <none>        5601/TCP             27m
kubernetes      ClusterIP   10.68.0.1      <none>        443/TCP              6h5m
```

Figura 100: Obtención de servicios ejecutando en el clúster.

En la figura 100, se puede observar un servicio de Elasticsearch para que Fluentd pueda comunicar y volcar los logs en el volumen persistente y que posteriormente se puedan consumir por Kibana. Además, es necesario un servicio de Kibana para poder acceder a la interfaz.

Tras la configuración de EFK, se hace un redireccionamiento de los servicios de Elasticsearch y Kibana en los puertos 9200 y 5601, respectivamente (ver figura 101 y 102). Esto es necesario ya que el tipo de servicio no es NodePort y no se dispone de un Ingress para controlar el tráfico.

```
tahirfarooq000@cloudshell:~/elastic (proyecto-cluster-kubernetes)$ kubectl port-forward -n kube-system svc/elasticsearch-logging 9200:9200
Forwarding from 127.0.0.1:9200 -> 9200
```

Figura 101: Redireccionamiento del servicio para permitir tráfico desde exterior.

```
tahirfarooq000@cloudshell:~/kibana (proyecto-cluster-kubernetes)$ kubectl port-forward -n kube-system svc/kibana-logging 32010:5601
Forwarding from 127.0.0.1:32010 -> 5601
Handling connection for 32010
```

Figura 102: Redireccionamiento del servicio para permitir tráfico desde exterior.

Finalmente, se puede acceder a la dirección del nodo en el puerto 9200 para observar la respuesta correcta de Elasticsearch (ver figura 103).

Figura 103: Respuesta de Elasticsearch.

4.4.2.7 Instalación de Kubernetes de forma nativa en Google Cloud

En este apartado, se realizará la instalación del clúster de Kubernetes que ofrece Google (sin necesidad de configurar máquinas virtuales, redes, direcciones IP, comunicación o Ansible).

En primer lugar, se crea un nuevo proyecto para almacenar todos los datos generados del clúster bajo el nombre de "Proyecto clúster Kubernetes" (ver figura 104).

Figura 104: Creación de nuevo proyecto en Google Cloud.

A continuación, se desplaza hacia la sección de Kubernetes Engine, en concreto, en la sección Clústeres (ver figura 105).

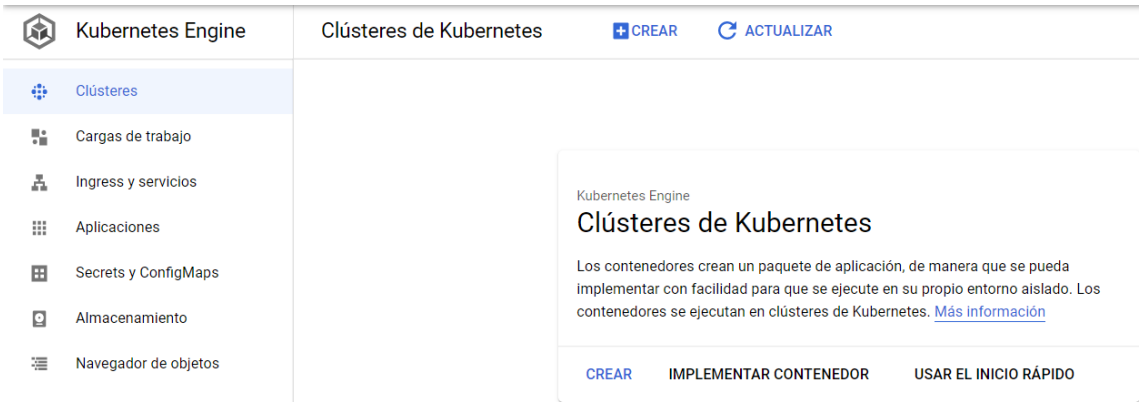

Figura 105: Sección de Kubernetes Engine: Clústeres.

Google ofrece dos modos para crear el clúster: Modo estándar o Autopilot.

La diferencia radica en que el modo de operación Autopilot es un clúster cuya configuración está automatizada por Google y no permite administración ni configuración de nodos. En cambio, el modo estándar permite configurar todo el clúster: configurar escalado horizontal, nodos, cargas de trabajo además de tener permisos de ejecución sobre el nombre de espacio "kube-system".

Para este proyecto, se escoge el modo estándar ya que se requiere el acceso al nombre de espacio "kube-system" para crear pods. En primer lugar, se indica el nombre del clúster ("cluster-cloud"), ubicación del clúster ("europe-west1-b"), cantidad de nodos (3), versión de K8S ("1.22.8"), el resto de configuración se deja por defecto, finalmente se crea el clúster mediante línea de comando de la figura 106.

```
gcloud beta container --project "proyecto-cluster-kubernetes" clusters create "cluster-
cloud" --zone "europe-west1-b" --no-enable-basic-auth --cluster-version "1.22.8-
gke.202" --release-channel "regular" --machine-type "e2-medium" --image-type
"COS_CONTAINERD" --disk-type "pd-standard" --disk-size "100" --metadata disable-legacy-
endpoints=true --scopes "https://www.googleapis.com/auth/cloud-platform" --max-pods-
per-node "110" --num-nodes "3" --logging=SYSTEM,WORKLOAD --monitoring=SYSTEM --enable-
ip-alias --network "projects/proyecto-cluster-kubernetes/global/networks/default" --
subnetwork "projects/proyecto-cluster-kubernetes/regions/europe-
west1/subnetworks/default" --no-enable-intra-node-visibility --default-max-pods-per-
node "110" --no-enable-master-authorized-networks --addons
HorizontalPodAutoscaling,HttpLoadBalancing,GcePersistentDiskCsiDriver --enable-
autoupgrade --enable-autorepair --max-surge-upgrade 1 --max-unavailable-upgrade 0 --
enable-shielded-nodes --node-locations "europe-west1-b"
```

Figura 106: Creación del clúster mediante línea de comando.

Al cabo de unos minutos, el clúster se ha terminado de aprovisionar (ver figura 107).

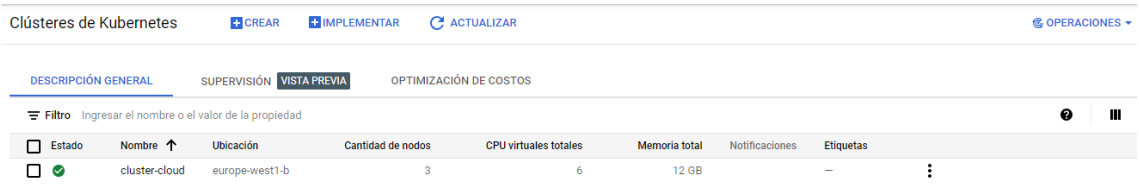

Figura 107: Aprovisionamiento completo del clúster.

En el clúster generado por Google con la opción estándar se pueden analizar los logs mediante StackDriver. De forma nativa Google ofrece integración nativa de Fluentd dentro de los pods. Fluentd se instala como sidecar dentro de los pods que se despliegan, es decir, se trata de un contenedor ejecutando dentro del pod. Por tanto, todos los pods están preparados para enviar logs a Fluentd que los va a almacenar en un volumen persistente fuera del clúster. Los logs de dicho disco son utilizados por la herramienta StackDriver para visualización de logs.

Como conclusión de esta instalación, no es necesario la instalación de herramientas ELK o EFK para la monitorización de registros de los contenedores ya que Google ofrece dicha funcionalidad de forma nativa a través de su herramienta: StackDriver.

5. Implantación

En este apartado, se van a realizar pruebas para medir el alcance de los objetivos propuestos.

Por una parte, se verifica la solución implantada mediante las herramientas ELK y por otra parte, se analizará la opción de monitorización de logs mediante StackDriver.

Finalmente, se realizará el despliegue de una aplicación basada en microservicios y se monitorizará sus logs.

5.1 Pruebas y resultados ELK

En primer lugar, tras realizar el despliegue, se accede a la dirección que aloja Kibana y se va al apartado de índices. En dicho apartado, se busca el índice de nombre Logstash.

Para encontrar índices, se puede realizar la siguiente consulta a Elasticsearch (ver figura 108).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ curl  http://localhost:9200/_cat/indices?v
health status index                uuid                   pri rep docs.count docs.deleted store.size pri.store.size
green  open   .kibana_1            VYVmlyltRCKfdqjfpwoLRg   1   1          5            0     89.3kb           46kb
green  open   logstash-2022.07.28 tiBrLo9ATJq8isf8XiPncg   1   1      11959            0     12.1mb          6.5mb
```

Figura 108: Obtención de índices almacenados en Elasticsearch.

Una vez, se tiene el índice localizado, se va al dashboard de Kibana para comenzar a filtrar logs por dicho índice (ver figura 109).

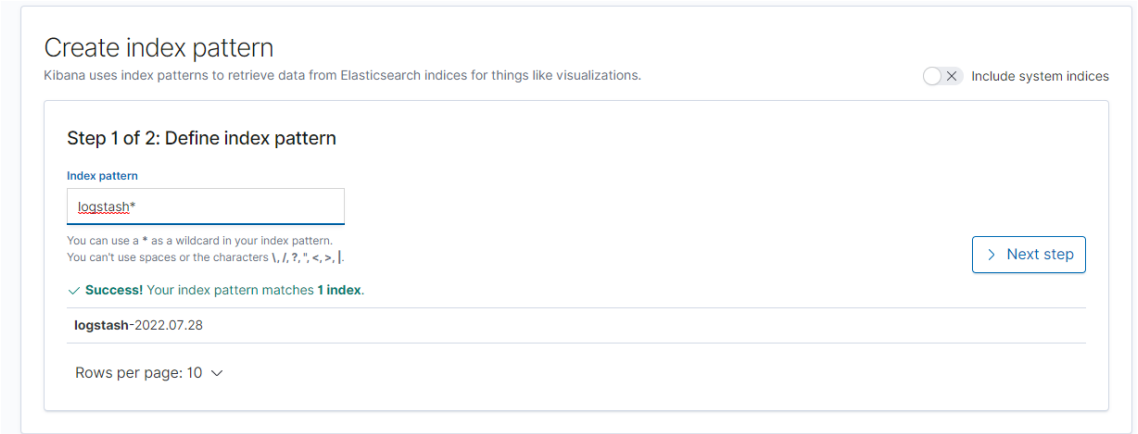

Figura 109: Búsqueda de índice “Logstash*”.

En la figura 109 se selecciona el índice “logstash*”, mediante esto se indica a Kibana que consulte logs almacenados en Elasticsearch que coincidan con el índice “logstash*”.

Tras seleccionar el índice, en el apartado “Discover” se puede empezar a monitorizar logs (ver figura 110).

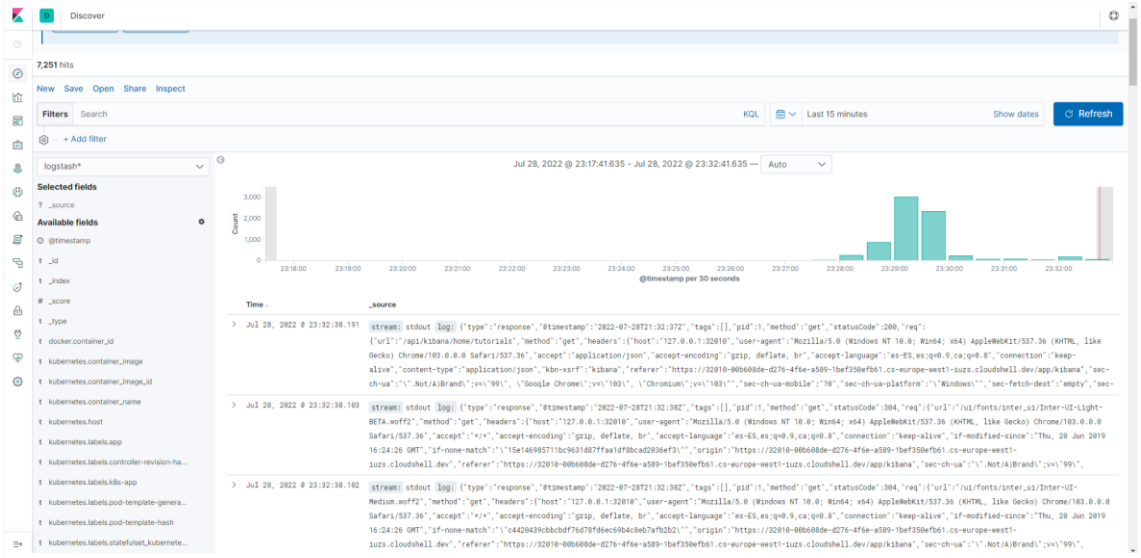

Figura 110: Apartado Discover en Kibana.

Como se aprecia en la figura 110, aparecen logs de Kibana únicamente ya que no hay ninguna aplicación ejecutando. En primer lugar, se van a crear pods que imprimen mensajes en los idiomas inglés (pod "**counter-en**") y español (pod "**counter-es**"). En la figura 111 se puede apreciar el formato de los logs emitidos por el pod "counter-es".

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl logs counter-es | tail -n 5
97: Buenos dias, donde estan mis mensajes?? Thu Jul 28 21:44:31 UTC 2022
98: Buenos dias, donde estan mis mensajes?? Thu Jul 28 21:44:41 UTC 2022
99: Buenos dias, donde estan mis mensajes?? Thu Jul 28 21:44:51 UTC 2022
100: Buenos dias, donde estan mis mensajes?? Thu Jul 28 21:45:01 UTC 2022
101: Buenos dias, donde estan mis mensajes?? Thu Jul 28 21:45:11 UTC 2022
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl get pods
NAME                    READY   STATUS    RESTARTS   AGE
counter-en              1/1     Running   0          17m
counter-es              1/1     Running   0          17m
```

Figura 111: Obtención de logs del pod counter-es y consulta de pods ejecutando dentro del clúster.

Los logs emitidos por los pods "counter-es" y "counter-en" por terminal van a ser capturados por el agente Filebeat y enviados a Logstash que se encargará de agregar la marca de tiempo así como añadir metadatos (etiquetas, contenedor emisor, pod, nombre de espacio, IP cliente, etc.) enviándolos a Elasticsearch. Finalmente, el usuario mediante la interfaz gráfica de Kibana puede filtrar por diferentes metadatos.

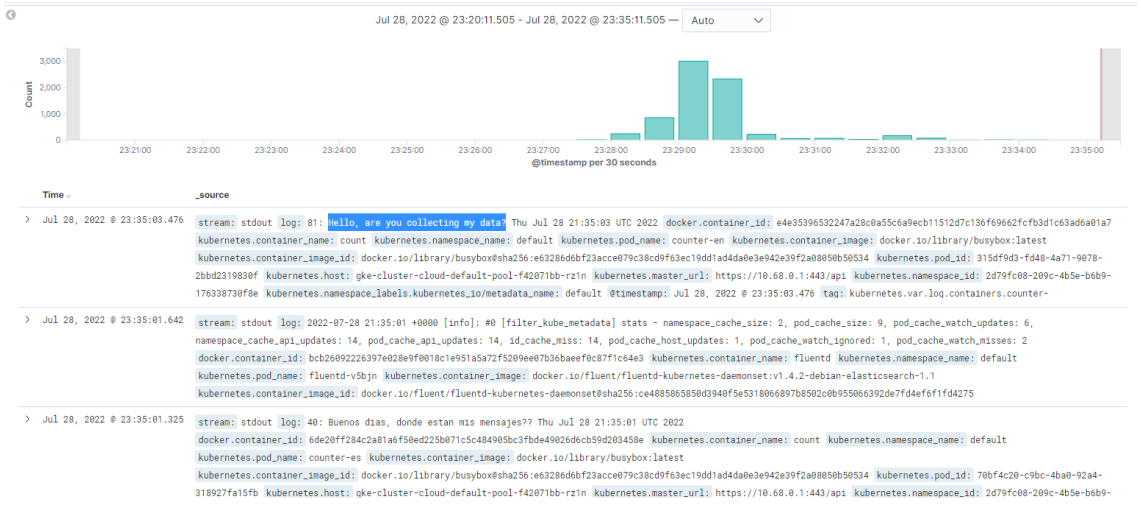

Figura 112: Observación de logs en Kibana.

En la figura 112 se puede observar que se ha recibido el mensaje: “Hello, are you collecting my data?” a las 23:35:03 de 28 Julio del pod “counter-en” ejecutando en el nombre de espacio por defecto, además se ven identificadores y etiquetas que dan más información de contexto del log.

Como se ha comentado previamente, se puede filtrar los logs por diferentes metadatos, a continuación, se va a filtrar por nombre del pod.

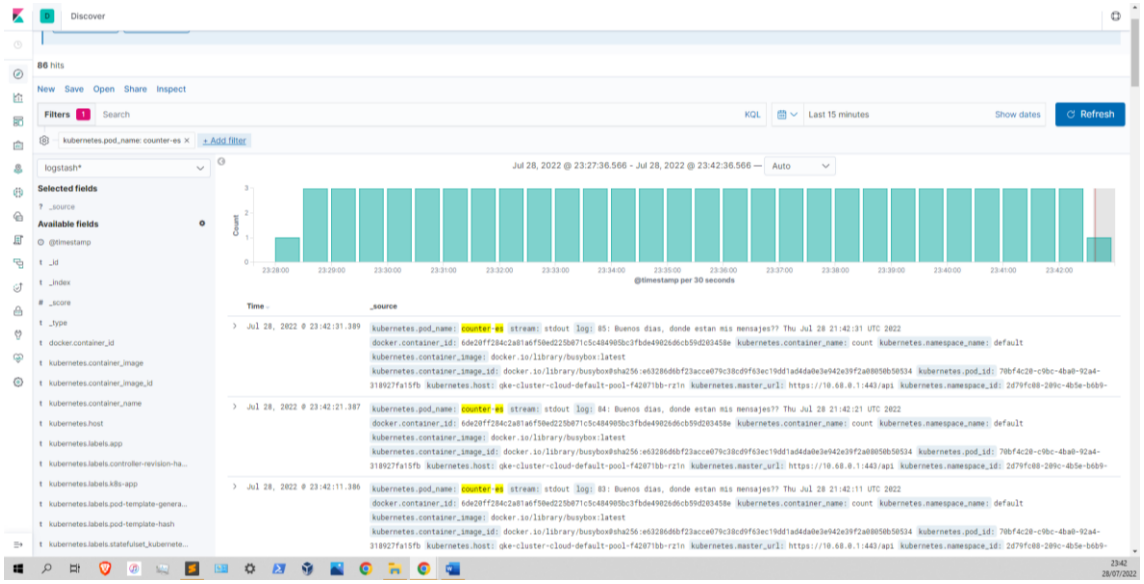

Figura 113: Filtración de logs.

En la figura 113 se puede observar únicamente los logs del pod “counter-es”. El diagrama representa gráficamente los logs emitidos, se pueden buscar por marca de tiempo haciendo clic sobre un determinado instante en el diagrama de barras superior.

En este instante, hay pods emitiendo logs dentro del clúster, para asegurar la persistencia de dichos logs, se va a detener el pod “counter-en” y se va a intentar acceder a sus logs tras tumbar el pod (ver figura 114).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl delete pod counter-en
pod "counter-en" deleted
```

Figura 114: Eliminación del pod "counter-en".

Se observa que mediante el comando “kubectl logs” ya no se pueden acceder a los logs (figura 115).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl logs counter-en
Error from server (NotFound): pods "counter-en" not found
```

Figura 115: Acceso erróneo a logs del pod inexistente.

Tras unos minutos, se comprueba los registros a través de la interfaz de Kibana.

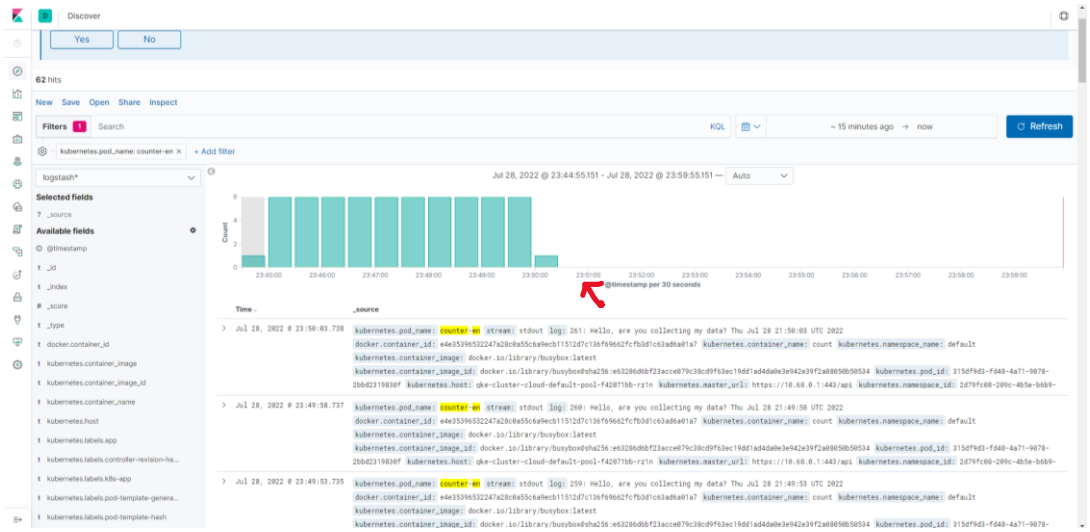

Figura 116: Historial del pod “counter-en”.

En la figura 116 se pueden observar los logs emitidos hasta la detención del pod “counter-en” (ver flecha roja en la figura). Por tanto, se puede concluir que mediante la pila ELK es posible observar logs tras la muerte del pod. Esto es posible ya que los logs han sido almacenados en el volumen persistente que monta Elasticsearch y Kibana tiene acceso al servicio de Elasticsearch para consumir dichos logs. De esta forma, se puede consultar el último log con su marca de tiempo junto al mensaje de error permitiendo saber en qué momento la aplicación ha dejado de funcionar y su causa exacta.

Si se consulta el pod que emite mensajes en español (pod "**counter-es**") , se puede apreciar que los agentes siguen recolectando logs de otros pods de forma totalmente independiente al pod "**counter-en**" detenido (ver figura 117).

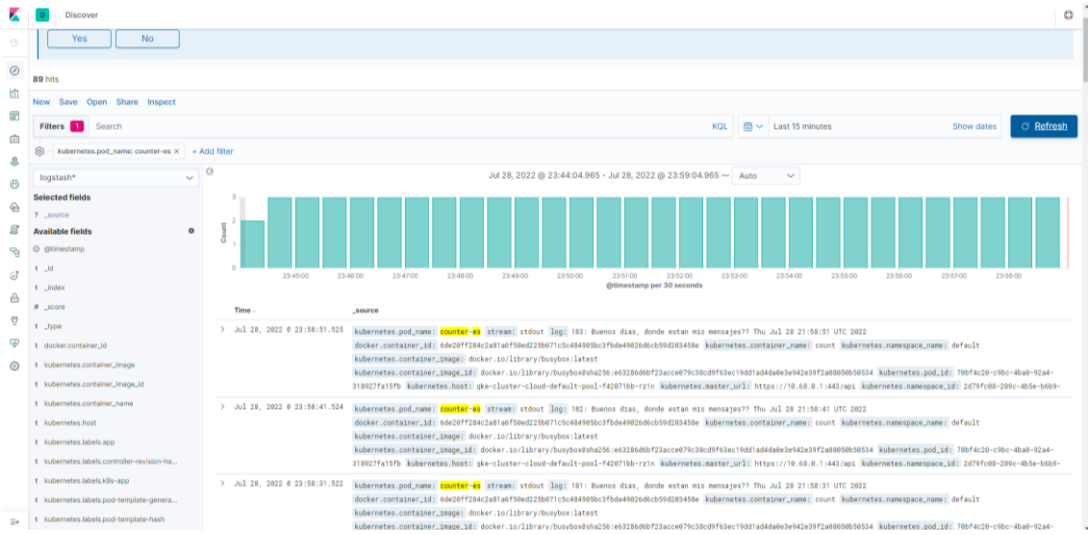

Figura 117: Historial del pod "counter-es".

5.2 Pruebas y resultados StackDriver

En este apartado, se estudiarán métodos de obtención de logs mediante la herramienta nativa de Google Kubernetes Engine: StackDriver.

Para ello, en primer lugar, se habilita la opción de logging durante la creación del clúster con la opción: "- -logging" y el resto de configuración se deja por defecto ya que Google se encargará automáticamente de ella.

Para observar el funcionamiento, se va a desplegar una aplicación en el nombre de espacio "tfmcloud" (ver figura 118).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl create namespace tfmcloud
namespace/tfmcloud created
```

Figura 118: Creación de nombre de espacio "tfmcloud".

Se prepara un pod que imprime la fecha junto al mensaje " INFO hello" cada 1 segundo.

En el fichero de configuración de la figura 119 se puede observar que se crea un pod en el nombre de espacio por defecto y toma por parámetros un bucle infinito que se ejecutará en la consola del contenedor. De esta forma se consigue una aplicación emitiendo logs.

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ cat myapp.yaml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
spec:
  containers:
  - image: busybox
    name: application
    args: [/bin/sh, -c,
           'while true; do echo "$(date) INFO hello"; sleep 1; done']
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl create -f myapp.yaml --namespace=tfmcloud
pod/myapp created
```

Figura 119: Creación y despliegue del pod.

Para analizar los logs, Google ofrece el apartado de operaciones de logging operado por StackDriver. En el índice de explorador de registros, se puede obtener todos los logs de todos los componentes del clúster (ver figura 120).

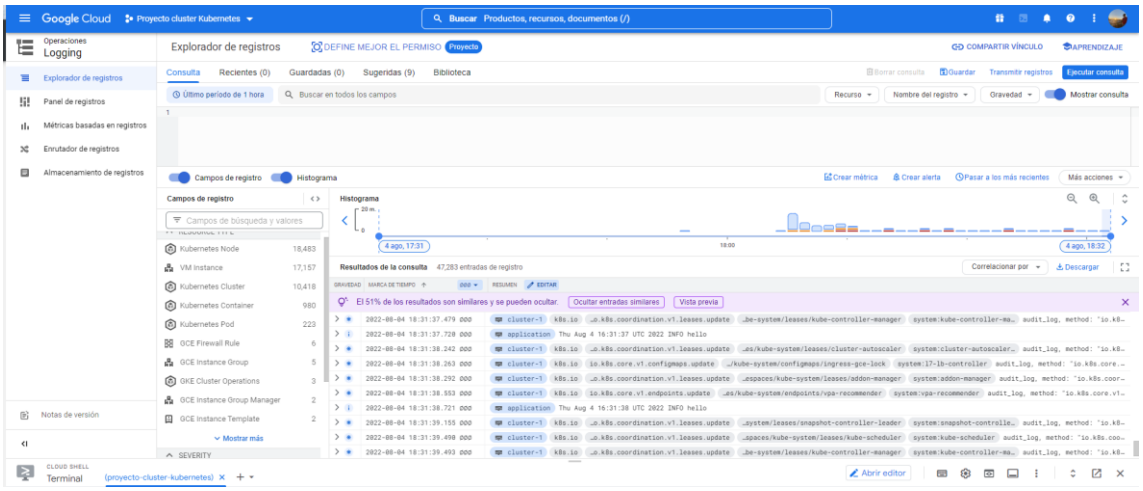

Figura 120: Obtención de logs del clúster.

Además, al igual que Kibana, existe la posibilidad de filtrar logs por nombre de pod (ver figura 121).

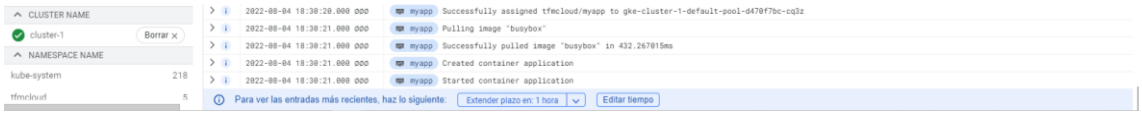

Figura 121: Obtención de logs del pod.

Se puede observar en la figura 121, los logs que hacen referencia a la descarga de la imagen “busybox”, creación y despliegue del contendor: se tratan del ciclo de eventos desde la lectura del fichero “yaml” hasta el despliegue del contenedor.

Además, se puede filtrar por contenedores dentro de un pod (ver figura 122).

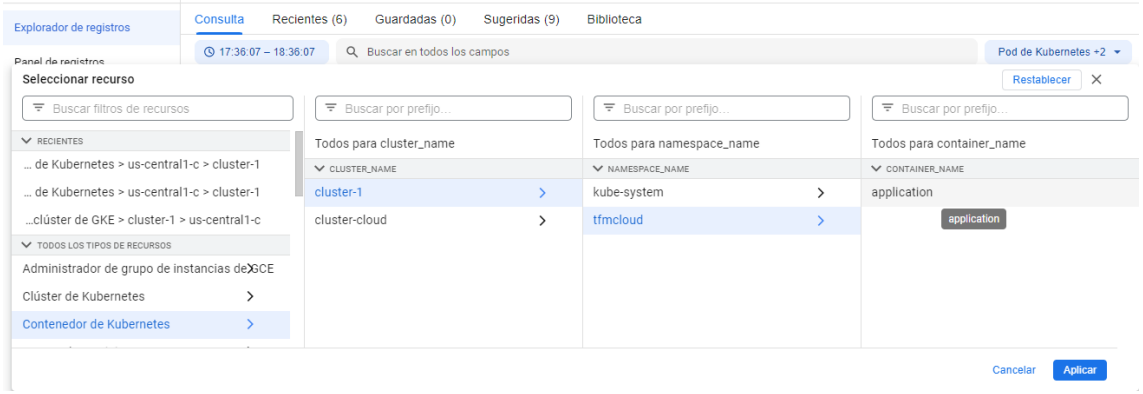

Figura 122: Aplicación del filtro para obtener logs del contendor.

En la figura 122, se están filtrando los logs de la aplicación “application” en el nombre de espacio “tfmcloud” ejecutando dentro del clúster “cluster-1”.

Tras hacer clic en aplicar, se obtienen todos los logs filtrados (ver figura 123).

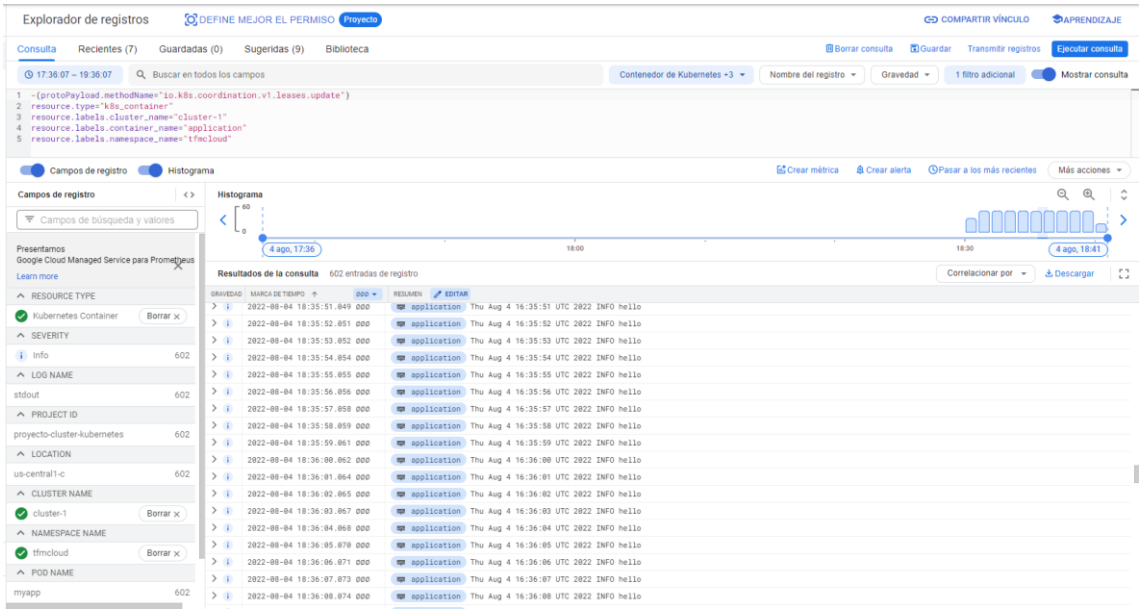

Figura 123: Obtención de logs de la aplicación.

Ahora se simula la caída del contenedor borrando el pod (ver figura 124).

```
tahirfarooq000@cloudshell:~ (proyecto-cluster-kubernetes)$ kubectl delete pod myapp -n tfmcloud
pod "myapp" deleted
```

Figura 124: Se mata el pod y los contenedores que ejecutaba.

Tras unos instantes, se puede apreciar que el pod ha dejado de emitir logs (ver histograma en la figura 125). Sin embargo, se sigue teniendo acceso a todos los logs del contendor, esto ocurre ya

que StackDriver (al igual que ocurría con Kibana) está recuperando información de disco persistente desacoplado de la vida útil de cualquier pod.

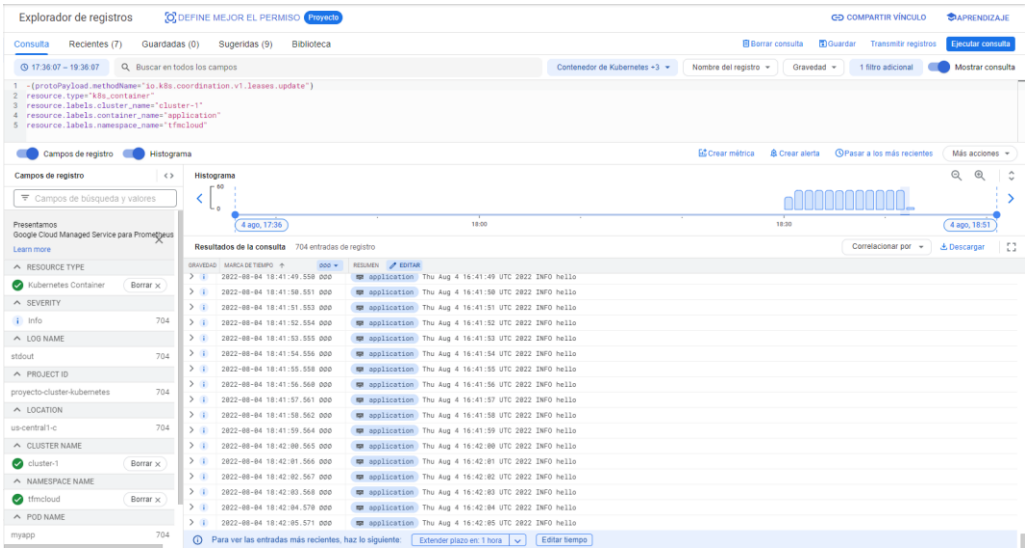

Figura 125: Acceso a los logs de la aplicación tras la caída del pod.

5.3 Pruebas y resultados de aplicación de prueba desplegado en Google Cloud

En este apartado, se despliega una aplicación basada en microservicios real y se analizarán sus logs.

En primer lugar, se dirige a la consola web colocada en la barra de navegación (ver figura 126).

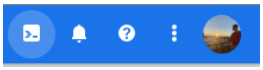

Figura 126: Consola de navegación.

Se descarga la aplicación de ejemplo disponible en este repositorio de Google Cloud Platform [17]. Se trata de una aplicación de comercio electrónico en la que los usuarios pueden buscar artículos, añadir al carrito y comprar.

En la figura 127 se puede observar un diagrama de las relaciones entre los diferentes microservicios.

[17] https://github.com/GoogleCloudPlatform/microservices-demo

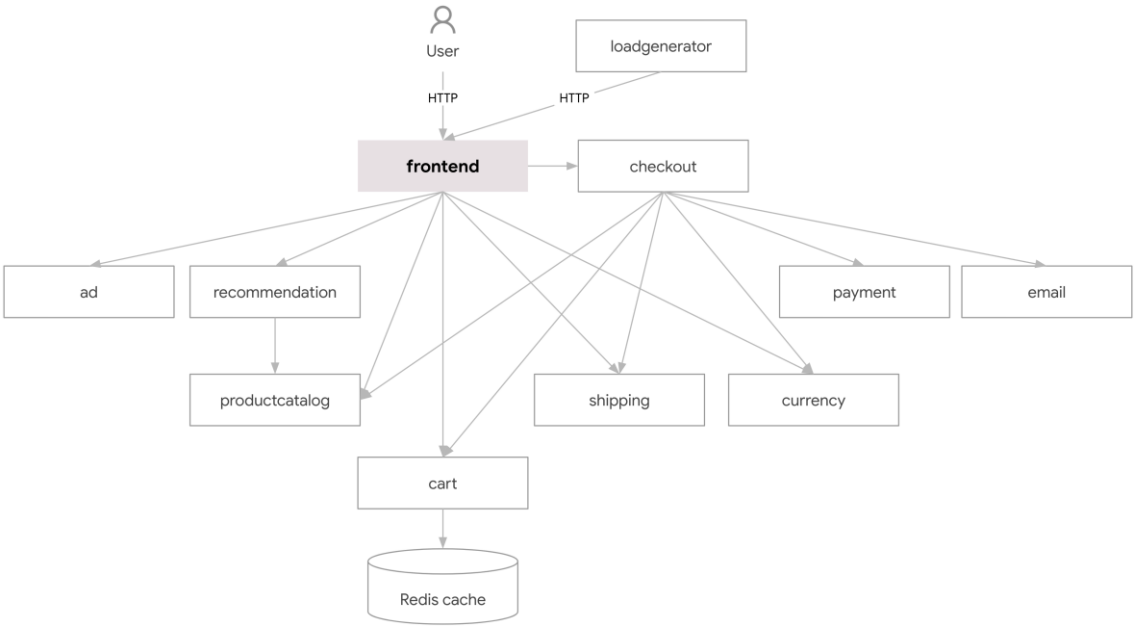

Figura 127: Fuente repositorio: https://github.com/GoogleCloudPlatform/microservices-demo

En la figura 127 se observa que las peticiones del navegador son redirigidas a las réplicas de frontend tras el balanceado de carga. Existen los diferentes microservicios que ofrecen funcionalidades: ad, recommendation, catalog, shipping, payment, currency, email, cart. Además, el microservicio de carrito guarda sus datos en redis (caché temporal).

Todos los microservicios se ejecutan dentro de los pods y se comunican entre ellos vía servicio (objeto de Kubernetes).

La organización de la aplicación se realiza por carpetas (ver figura 128).

Figura 128: Despliegue de la aplicación.

Dentro de la carpeta src, se pueden encontrar submódulos, uno por cada microservicio. Cada microservicio contiene un fichero Dockerfile que engloba la aplicación para su despliegue dentro de un contenedor.

Para realizar el despliegue de la aplicación completa con Kubernetes, existe el fichero "Kubernetes-manifests.yaml" dentro de la carpeta "release" del proyecto (ver figura 128).

Pero antes, se ha de configurar el clúster en esta aplicación, para ello, se pulsa en el editor sobre el índice Google Kubernetes Engine Explorer para seleccionar el clúster creado como el activo por defecto (figura 129).

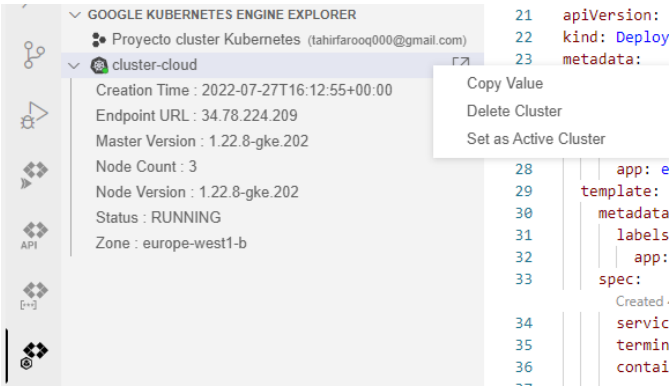

Figura 129: Configurando clúster como activo.

Se puede observar la creación de los objetos necesarios para el funcionamiento de la aplicación tras la ejecución del manifiesto (ver figura 130).

```
tahirfarooq000@cloudshell:~/microservices-demo (proyecto-cluster-kubernetes)$ kubectl apply -f ./release/kubernetes-manifests.
yaml
deployment.apps/emailservice created
service/emailservice created
deployment.apps/checkoutservice created
service/checkoutservice created
deployment.apps/recommendationservice created
service/recommendationservice created
deployment.apps/frontend created
service/frontend created
service/frontend-external created
deployment.apps/paymentservice created
service/paymentservice created
deployment.apps/productcatalogservice created
service/productcatalogservice created
deployment.apps/cartservice created
service/cartservice created
deployment.apps/loadgenerator created
deployment.apps/currencyservice created
service/currencyservice created
deployment.apps/shippingservice created
service/shippingservice created
deployment.apps/redis-cart created
service/redis-cart created
deployment.apps/adservice created
service/adservice created
```

Figura 130: Lanzamiento de la aplicación.

Para realizar el seguimiento del despliegue, se puede clicar en el subíndice “Kubernetes Explorer” de la barra lateral (ver figura 131). A medida que se inician los pods, se mostrará sobre ellos un estado verde “Running”.

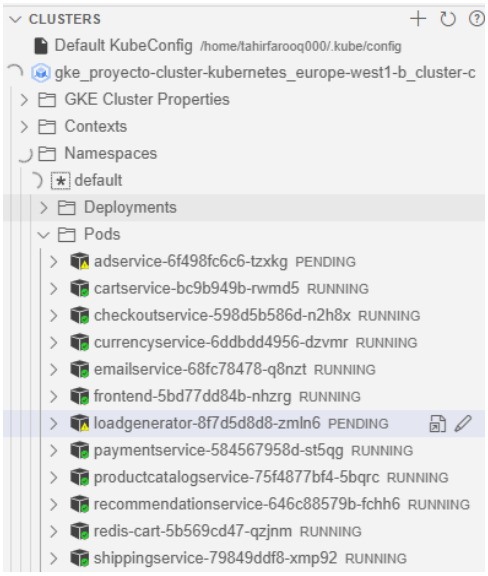

Figura 131: Estado del despliegue.

Existen varias formas de observar logs, en primer lugar, dando clic derecho sobre el pod desplegado, y ver logs (ver figura 132).

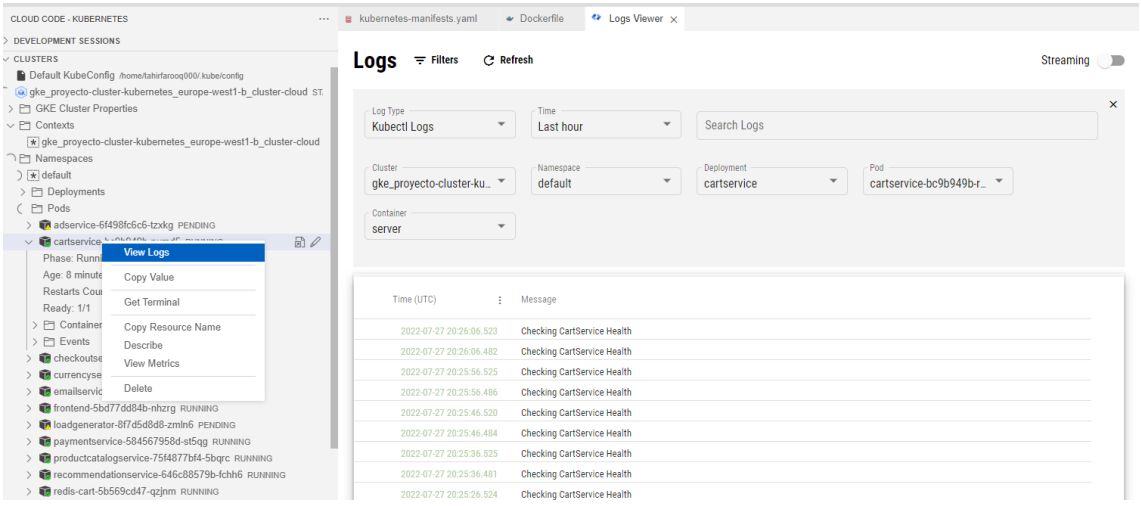

Figura 132: Obtención de logs del microservicio cartservice.

Además, también se puede acceder a la funcionalidad “describe” de la aplicación para observar causas del error: Insuficiente CPU (ver figura 133).

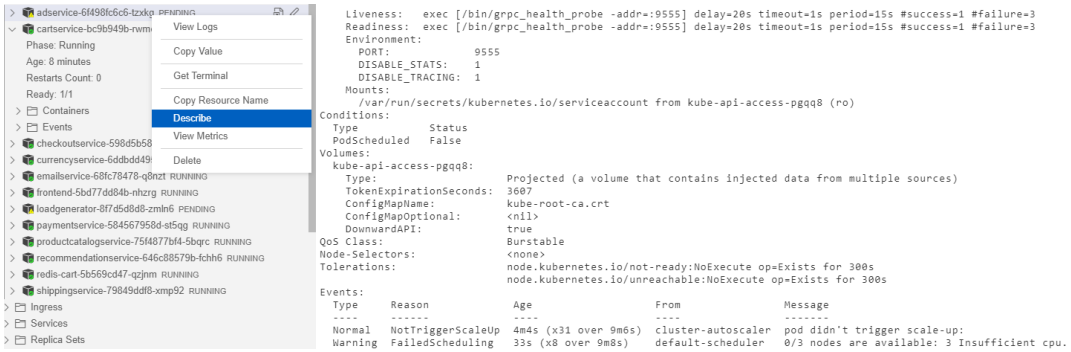

Figura 133: Descripción del pod adservice.

Por otra parte, se pueden observar logs en el registro del clúster (figura 134) o filtrar los logs por pod o servicios en el apartado cargas de trabajo (figura 135).

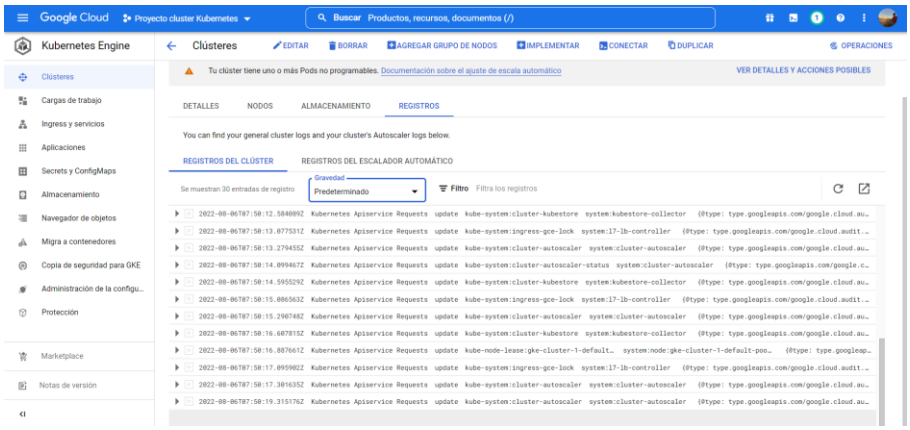

Figura 134: Logs del clúster.

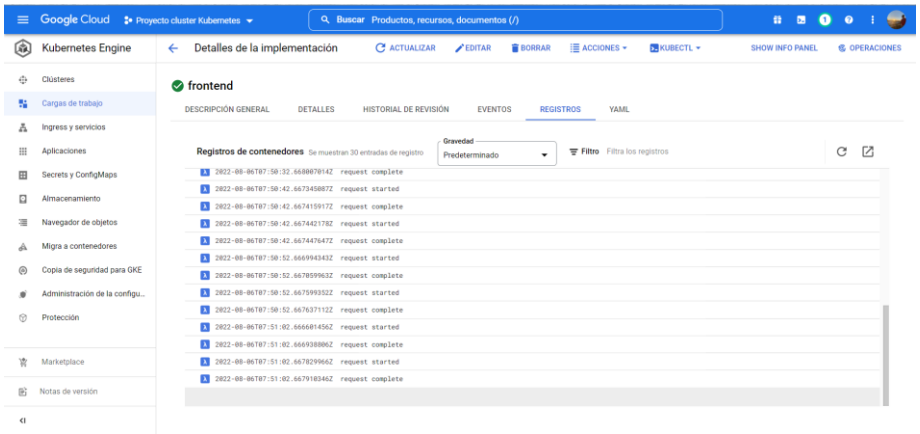

Figura 135: Logs del servicio frontend.

En el figura 135, se pueden observar logs del servicio “frontend” que recibe las solicitudes del usuario y se encarga de servir las peticiones.

Se puede acceder al despliegue realizado consultando la IP del servicio “frontend” (ver figura 136).

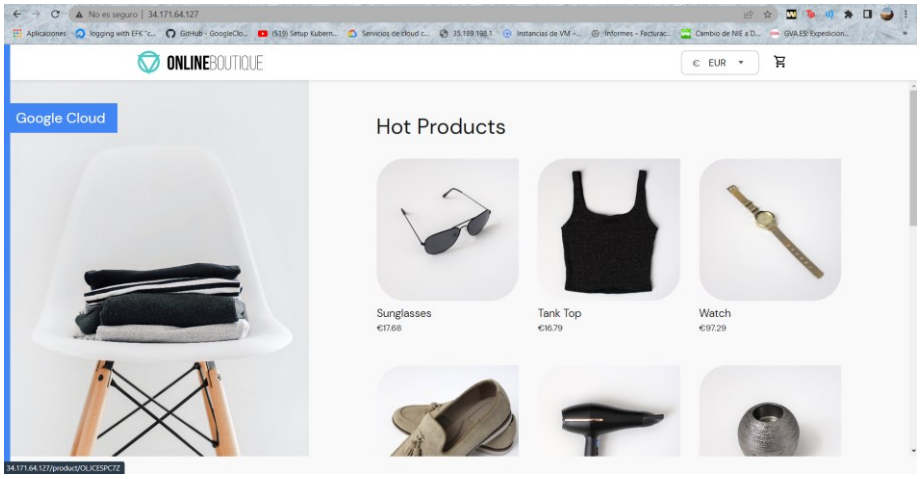

Figura 136: Comercio electrónico.

Los logs generados se almacenarán dentro de un disco persistente hasta que se borre expresamente dicho disco.

6. Conclusiones

El presente desarrollo cumple con los objetivos marcados al inicio de este. En primer lugar, tras la instalación de Kubernetes mediante Ansible, se ha estudiado los comandos que ejecutan contra la API y dashboard para dar solución al problema de la persistencia de logs en los entornos local y en las máquinas virtuales de Google Cloud.

Tras la caída de la aplicación, los logs no son accesibles a menos que se realice una tarea de exportación en tiempo real para cada una de las aplicaciones ejecutando dentro del clúster. Tras ver la limitación, se ha instalado Elasticsearch, Logstash y Kibana. Con esta pila, se ha solucionado el problema de persistencia de logs además de poder ver y filtrar los logs de forma gráfica.

En segundo lugar, se ha estudiado la solución de logs nativa en un clúster de Google Cloud. Google ofrece logging a nivel de pods para exportar y filtrar logs sin la instalación de ninguna herramienta por parte del administrador, siendo una solución rápida y completa para desarrolladores de aplicaciones.

En tercer lugar, se ha realizado pruebas sobre el clúster para medir el impacto de la solución implementada con las herramientas ELK y sobre la versión nativa en Cloud: StackDriver. Además, se ha desplegado una aplicación basada en microservicios sobre el clúster de Google Cloud y analizado sus logs.

Finalmente, se ha podido concluir y probar un sistema persistente de logs para dar solución al problema de los logs en local y en la nube.

Como conclusión personal, este desarrollo ha permitido obtener experiencia en la instalación y gestión del clúster tanto en local como en nube. También se extrae la importancia de realizar un estudio y planificación correcto para analizar problemas presentes en Kubernetes. Además, la realización de este proyecto ha servido para determinar los costes de uso de las máquinas virtuales y clústeres en Google Cloud.

En el entorno empresarial es común hallar un clúster instalado sobre los propios servidores de la empresa, por tanto la instalación realizada en este tesis puede servir de guía para implantar la solución en un entorno personal o profesional.

6.1 Trabajo futuro

La realización de este proyecto cumple con los requisitos formulados inicialmente, sin embargo, existen aspectos para mejorar para que el sistema sea más robusto y seguro.

Actualmente, los logs de las aplicaciones se almacenan en un volumen persistente dentro del clúster. Como extensión de este desarrollo, se puede externalizar el salvado de los logs en un nodo externo del clúster, de esta forma si el clúster falla, se puede acceder a los logs accediendo al nodo agregado. Sin embargo, la solución nativa de Google Cloud ofrece ya la independencia entre el volumen persistente y el clúster.

Por otra parte, en el despliegue local se pueden estudiar formas de añadir seguridad de acceso a los logs para concluir el estado de aplicación descartando posibles manipulaciones en los logs. En el despliegue en Google Cloud el acceso a los apartados de logging están protegidos por cuentas de acceso.

En el desarrollo realizado, los servicios son de tipo NodePort, siendo peligroso en un entorno de producción ya que cualquier usuario no administrador puede comprometer la confidencialidad de los logs, en su lugar se debería utilizar un Ingress para configurar protocolos y permisos de acceso a los servicios.

Finalmente, como alternativa al despliegue de herramientas mediante ficheros yaml, se puede estudiar la implementación de la solución mediante Helm (gestor de paquetes de Kubernetes).

6.2 Relación con estudios cursados

Los años de grado y máster han ayudado para entender el funcionamiento de la tecnología utilizada en este desarrollo. En concreto, en este apartado, se va a relacionar el desarrollo con algunas partes de las asignaturas.

En primer lugar, la asignatura de Configuración y Optimización de Sistemas de cómputo ha servido de base para el entendimiento de Kubernetes. En esta asignatura se presupuestó el montaje de un clúster físico privado, lo cual ha servido para comparar con el coste de tener un clúster en la nube, siendo la opción en la nube notablemente inferior frente al coste que supone mantener el clúster privado. En segundo lugar, se estudió Kubernetes y se hizo la instalación en las máquinas virtuales mediante Kubeadm. A través de esta asignatura se obtuvo las herramientas y la capacidad de resolver problemas técnicos durante la instalación de Kubernetes en las máquinas virtuales en local. Al final de la asignatura, se comentó la idea de realizar el despliegue de Kubernetes en el entorno Cloud junto al problema de análisis de logs como desafío de TFM.

En segundo lugar, en la asignatura de Sistemas y Aplicaciones Distribuidas una de las cosas que se estudian es el patrón de arquitectura de microservicios y ha servido para entender la importancia de implementar módulos independientes para poder desplegar dentro de Kubernetes y escalar los módulos que reciben más carga en momentos puntuales. Además, en esta asignatura, se han estudiado los dos tipos de virtualizaciones: software y hardware que han servido de guía para la implementación práctica.

En tercer lugar, en las asignaturas de Redes y Seguridad y Computación de Altas Prestaciones se estudiaron distintos tipos de virtualizaciones y se realizó primer contacto con la computación en la nube: Ejecución de contenedores en Azure Management.

Todos estos conocimientos adquiridos a lo largo de los años han llevado a cumplir con los objetivos marcados para este desarrollo y fomentado el aprendizaje de nuevas tecnologías.

7. Bibliografía y referencias

Consulta de materiales de Poliformat de la asignatura de Configuración y Optimización de Sistemas de Cómputo, Redes y Seguridad, Sistemas y Aplicaciones Distribuidas y Computación de Altas Prestaciones.

BRENDAN BURNS, JOE BEDA, KELSEY HIGHTOWER (2019). Kubernetes Up & Running: Dive into the Future of Infrastructure.

PHILIPPE MARTIN (2021). Kubernetes: Preparing for the CKA and CKAD Certifications.

BRENDAN CREANE & ADMIT GUPTA (2021). Kubernetes Security and Observability: A Holistic Approach to Security Containers and Cloud Native Applications.

MARKO LUKSA (2018). Kubernetes In Action.

HIDETO SAITO, HUI-CHUAN CHLOE LEE, KE-JOU CAROL HSU (2018). Kubernetes Cookbook: Practical solutions to container orchestration.

Información acerca de monitorización, logs y debugging consultado en siguiente repositorio: https://kubernetes.io/es/docs/tasks/debug-application-cluster/

Investigación acerca de la arquitectura de logging: https://kubernetes.io/docs/concepts/cluster-administration/logging/

Referencia acerca de Elastic y Kibana en el apartado 3: Escribiendo Logs con Elasticsearch y Kibana del siguiente artículo: https://kubernetes.io/es/docs/tasks/debug-application-cluster/_print/

Documentación acerca de logging en la siguiente bibliografía: https://cloud.google.com/logging/

Información acerca de las herramientas ELK: https://aws.amazon.com/es/opensearch-service/the-elk-stack/

Instalación de herramientas de Kubernetes y documentación: https://kubernetes.io/es/docs/tasks/tools/

Definición y arquitectura de Kubernetes: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Comandos más utilizados a modo de chuleta para ejecutar contra la API de kubectl: https://kubernetes.io/docs/reference/kubectl/cheatsheet/

Repositorio de herramientas ELK: https://www.docker.elastic.co/

Definición y arquitectura de Kubernetes: https://www.redhat.com/es/topics/containers/what-is-kubernetes

Documentación acerca de Kubernetes en Wikipedia: https://es.wikipedia.org/wiki/Kubernetes

Guía de implementación de la pila ELK en local https://coralogix.com/blog/kubernetes-logging-with-elasticsearch-fluentd-and-kibana/ y https://www.weave.works/blog/kubernetes-observability-log-aggregation-using-elk-stack

Anexo: Objetivos de Desarrollo Sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	**Alto**	**Medio**	**Bajo**	**No Procede**
ODS 1. **Fin de la pobreza.**				✓
ODS 2. **Hambre cero.**				✓
ODS 3. **Salud y bienestar.**				✓
ODS 4. **Educación de calidad.**		✓		
ODS 5. **Igualdad de género.**		✓		
ODS 6. **Agua limpia y saneamiento.**				✓
ODS 7. **Energía asequible y no contaminante.**		✓		
ODS 8. **Trabajo decente y crecimiento económico.**	✓			
ODS 9. **Industria, innovación e infraestructuras.**	✓			
ODS 10. **Reducción de las desigualdades**.				✓
ODS 11. **Ciudades y comunidades sostenibles**.		✓		
ODS 12. **Producción y consumo responsables.**	✓			
ODS 13. **Acción por el clima.**	✓			
ODS 14. **Vida submarina.**				✓
ODS 15. **Vida de ecosistemas terrestres.**				✓
ODS 16. **Paz, justicia e instituciones sólidas.**				✓
ODS 17. **Alianzas para lograr objetivos.**				✓

Tabla 4: ODS.

A continuación se van a relacionar los objetivos marcados en la tabla 4 con el presente TFM.

En primer lugar, los ODS 1,2,3 no proceden ya que no se está llevando a cabo ninguna acción para remediar la pobreza ni se hace actos de ayuda relacionados con el hambre cero ni con el salud bienestar.

En cuando al ODS 4, Kubernetes está en constante evolución y para ello, es fundamental entrenar las capacidades de autoaprendizaje. Las tecnologías utilizadas en este desarrollo fomentan la calidad en la educación ya que Kubernetes es el orquestador por excelencia para realizar despliegues en entornos local y nube.

Referente al ODS 5 y 10 el sector de informática no discrimina a ninguna persona por su género. Se trata de un sector donde se valora más la capacidad y los conocimientos, donde la aparición de nuevas herramientas es común y es totalmente accesible en igualdad de condiciones por ambos géneros. Además, este sector brinda la posibilidad de trabajar donde y como quieren. En el presente desarrollo se soluciona una parte de logging relacionada con la labor de un administrador de sistemas siendo apta para ambos géneros.

Referente al ODS 6, 7, 11, 12, 13 hacen referencia a acciones sostenibles para el medio ambiente. Para el desarrollo en el entorno local se ha utilizado un portátil y la energía suministrada por la red eléctrica. Esta energía procede de varias fuentes sostenibles como eólica y solar.

Para el desarrollo en la nube se ha elegido la plataforma de Google Cloud. Google permite el despliegue de las máquinas virtuales en regiones de todo el mundo. Para el desarrollo de este proyecto, se ha utilizado la región de Bélgica para el despliegue tanto del clúster como máquinas virtuales ya que dicha zona se muestra como energía proveniente de fuentes libre de carbono (ver figura 137).

Seleccionar una ubicación ▾ | Selecciona un tipo de máquina ▾ | Selecciona otros recursos ▾ | Borrar todo

Zonas	Ubicación	Tipos de máquina	CPU	Recursos	Emisiones de CO_2 ▲
europe-west1-c	Saint-Ghislain, Bélgica, Europa	E2, N2, N2D, T2D, N1, M2, C2	Ivy Bridge, Sandy Bridge, Haswell, Broadwell, Skylake, Cascade Lake, Ice Lake, AMD EPYC Rome, AMD EPYC Milan	GPU	CO_2 bajo
europe-west1-d	Saint-Ghislain, Bélgica, Europa	E2, N2, N2D, T2D, N1, M1, M2, C2, C2D	Ivy Bridge, Sandy Bridge, Haswell, Broadwell, Skylake, Cascade Lake, Ice Lake, AMD EPYC Rome, AMD EPYC Milan	GPU	CO_2 bajo
europe-west1-b	Saint-Ghislain, Bélgica, Europa	E2, N2, N2D, T2D, N1, M1, M2, C2	Ivy Bridge, Sandy Bridge, Haswell, Broadwell, Skylake, Cascade Lake, Ice Lake, AMD EPYC Rome, AMD EPYC Milan	GPU	CO_2 bajo

Figura 137: Fuente extraída de: https://cloud.google.com/compute/docs/regions-zones

Además, Google está concienciado para maximizar la energía libre de carbono que consume su infraestructura o aplicaciones: “Google estableció recientemente el objetivo de que para el año 2030 usaremos electricidad sin emisiones de carbono a fin de que tus aplicaciones puedan funcionar las 24 horas del día, en cualquier lugar de todas las regiones de Google Cloud.” [18]

Para medir el origen de la energía, se utiliza le métrica CFE (que hace referencia al porcentaje de energía libre de carbono). A mayor CFE, más porcentaje de la energía usada en esa región proviene de una fuente ECO libre de carbono. Con esto política se consigue reducir el uso de carbono para generar energía.

Google ofrece la posibilidad de elegir la región, en este desarrollo se han utilizado las máquinas N2 en la zona europe-west1-c ubicadas en Bélgica que tienen una tasa del CFE del alrededor del 82%, es decir que el 82% de energía que se utiliza en una hora está libre de huella de carbono (ver figura 138).

[18] https://cloud.google.com/sustainability

Carbon data across GCP regions

Google Cloud Region	Location	Google CFE%	Grid carbon intensity (gCO_2eq/kWh)	Google Cloud net operational GHG emissions	
europe-north1	Finland	91%	127	0	Low CO_2
europe-southwest1	Madrid	*	121	0	Low CO_2
europe-west1	Belgium	82%	110	0	Low CO_2

Figura 138: Fuente extraída de https://cloud.google.com/sustainability/region-carbon

Referente a los ODS 8 y 9: Si las empresas o instituciones optan por escoger un medio reutilizable para desplegar su infraestructura, afecta directamente a la política y la economía de la empresa: Mejora su imagen frente al exterior, se reducen costes de infraestructuras y se optimizan recursos presentes ya que por ejemplo desaparecerían salas con nodos físicos que generan altas cargas de trabajo y consumo de energía. Al usar una nube se utilizarían los recursos necesarios evitando desperdiciar, se eliminan altos costes iniciales de la compra de un clúster y los costes de su mantenimiento, siendo la opción en la nube más barata pagando únicamente por lo que se consume (modelo pago por uso). Otra de las razones es que la empresa ya no es la única entidad que tiene la única copia de seguridad de los datos, haciendo que ante un riesgo de caída o pérdida de datos en local o nube (ambos a la vez son posible pero hay menor probabilidad), la probabilidad de recuperar información sea alta.

Referente a los ODS 14,15,16 y 17 no se aplican ya que no se lucha contra la paz, justicia in se tiene en cuenta acciones para la vida submarina, ni la vida de ecosistemas terrestres más allá de un consumo de recursos justos y formas de energía ECO.