UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Despliegue de un cluster Kubernetes altamente disponible en Google Cloud Platform

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Gaspar Aparicio, Ernesto

Tutor/a: Acebrón Linuesa, Floreal

CURSO ACADÉMICO: 2021/2022

Resumen

El cloud computing, es un modelo tecnológico que se está implementando en la mayoría de empresas en los últimos años. Sus ventajas son obvias, ya sea porque se usa Software como servicio (SaaS), Plataforma como servicio (PaaS) o Infraestructura como servicio (IaaS). Ello supone para las empresas un ahorro de costes, espacios seguros, fácil accesibilidad y una personalización del servicio según los requisitos del cliente.

El objeto de este trabajo es estudiar qué facilidades, herramientas y comodidades ofrece Google Cloud Platform a la hora de desplegar un clúster de Kubernetes altamente disponible, con el objetivo de que pueda ser comparado con sus competidores y ayudar a las empresas a seleccionar un servicio u otro según sus requisitos.

Palabras clave: clúster altamente disponible, Google Cloud Platform, Knative, Kops, Kubeadm, Kubernetes.

Abstract

Cloud computing is a technological model that has been implemented in most companies in recent years. Its advantages are obvious, whether it is Software as a Service (SaaS), Platform as a Service (PaaS) or Infrastructure as a Service (IaaS). This offers companies cost savings, secure spaces, easy accessibility and customisation of the service according to the client's requirements.

The aim of this paper is to study what facilities, tools and conveniences Google Cloud Platform offers when deploying a highly available Kubernetes cluster, so that it can be compared with its competitors and help companies to select one service or another according to their requirements.

Keywords: highly available cluster, Google Cloud Platform, Knative, Kops, Kubeadm, Kubernetes.

Agradecimientos

Quiero agradecer a todos los profesores que me han impartido durante estos 4 años de carrera, por abrirme al maravilloso mundo de la informática.

A mi tutor, Floreal Acebrón Linuesa, por motivarme a profundizar en este tema, y por guiarme durante este periodo.

A mi familia y a mi pareja por el apoyo incondicional continuo.

Tabla de contenidos

Índice de figuras

Índice de anexos

1. Introducción

1.1 Marco del proyecto

Este Trabajo de Fin de Grado (TFG) consiste en la implementación de un clúster de Kubernetes altamente disponible en Google Cloud Platform (GCP), con la intención de que pueda, en un futuro, ser comparado con sus competidores a nivel de precios y facilidades para el administrador.

Se ha desplegado una simple aplicación web que podrá escalar automáticamente según el número de peticiones que reciba. También, se han utilizado herramientas de monitorización para verificar el uso de recursos del clúster y que pueda llegar a avisar al administrador según la métrica que escojamos (uso excesivo de CPU, almacenamiento limitado etc.).

1.2 Motivación

La virtualización de sistemas e infraestructuras en empresas es una práctica cada vez más común y muy interesante. Dado que mi motivación principal con este trabajo siempre ha sido de aprendizaje de competencias técnicas nuevas, decidí que esta temática era muy interesante y que apenas se había profundizado durante el Grado.

Las alternativas eran desplegar un clúster Kubernetes altamente disponible, pero en otro proveedor: Azure, Alibaba Cloud, Amazon Web Services (AWS). Mi elección fue la de GCP debido a la familiarización con el servicio que había adquirido durante mi estancia en el extranjero realizando el programa Erasmus+.

1.3 Objetivos

El objetivo de este trabajo es ilustrar paso a paso cómo se realiza un despliegue de un clúster Kubernetes altamente disponible en GCP. Además, se mostrará cómo se ha realizado el lanzamiento de una aplicación web escalable, para que se asemeje lo mejor posible a un entorno de producción real.

Los objetivos son:

1. Estudiar las alternativas para desplegar un clúster Kubernetes en GCP: Kubeadm, Kops y Knative.
2. Comparar las soluciones a nivel de facilidad, consumo de recursos y coste.
3. Implementar sobre un clúster una aplicación web escalable automáticamente.
4. Usar herramientas de monitorización disponibles y de alerta al administrador.

Para alcanzar estos objetivos, ha sido necesario entender con profundidad la arquitectura de Kubernetes mediante:

1. Despliegue de un clúster con Kubeadm en local.
2. Uso de herramientas de balanceo de carga (MetalLB).
3. Implementación de un sistema de archivos de red (NFS).

1.4 Estructura del documento

En el segundo capítulo se realiza un análisis del estado del arte, donde se verá el contexto histórico de los sistemas de orquestación de contenedores.

En el tercer capítulo se explica la metodología utilizada en el proyecto y una descripción de las soluciones implementadas.

En el cuarto capítulo se realiza el análisis de los patrones y la arquitectura utilizada en Kubernetes, que nos ayudará a entender mejor el sistema.

En el quinto capítulo se documenta cómo ha sido el desarrollo de cada una de las soluciones.

En el sexto capítulo se implanta la solución.

En el séptimo capítulo se describen las pruebas que se han realizado para comprobar la calidad y funcionamiento.

Por último, en el octavo capítulo, se concretan las conclusiones que se han obtenido durante la elaboración del proyecto.

2. Estado del arte

2.1 Contexto tecnológico

Años atrás, las empresas solían adquirir sistemas informáticos, los cuales se instalaban en las propias empresas. La adquisición y el mantenimiento de estos equipos conllevaban una gran inversión económica, a esto hay que añadirle el gasto eléctrico y el personal especializado, que se debía encargar del mantenimiento de estos equipos.

La compra de servidores y equipo informático físico no solo conllevaba una gran inversión económica, si no que a menudo las empresa o particulares que adquirían las máquinas realizaban un consumo inferior de la capacidad de cómputo total y se desperdiciaban recursos.

Estas situaciones impulsaron la necesidad de que para las compañías existieran servicios de infraestructuras de pago bajo demanda y, por lo tanto, el surgimiento de empresas que satisfagan esta demanda. Así aparece el concepto de computación en la nube o “Cloud Computing”.

La computación en la nube es la distribución de recursos informáticos bajo demanda a través de Internet mediante un esquema de pago por uso. En lugar de comprar, poseer y mantener servidores y centros de datos físicos, existe la posibilidad de acceder a servicios tecnológicos como: capacidad informática, almacenamiento y bases de datos; en función de sus necesidades a través de un proveedor de la nube.

Por otro lado, el diseño de software siempre se ha realizado con una arquitectura monolítica, en la que el software se estructura de forma que todos los aspectos funcionales quedan acoplados y sujetos en un mismo programa. En este tipo de sistema, toda la información está alojada en un servidor. Por ello, no existe separación entre módulos y las distintas partes de un programa se encuentran muy acopladas. Esto supone un problema a largo plazo, porque que se trata de un sistema no escalable de manera sencilla. Además, las empresas requieren realizar cambios en el software e implementarlos de forma fácil y rápida. Todo ello desencadena la aparición de la arquitectura de microservicios.

La arquitectura de microservicios conforma la idea de dividir los sistemas en partes individuales, permitiendo que se puedan tratar y abordar los problemas de manera independiente sin afectar al resto. El uso de las arquitecturas basadas en microservicios ha ido escalando en los últimos años.

El trabajo conjunto de la arquitectura de microservicios y la computación en la nube supone para las empresas la manera más eficiente y eficaz, en cuanto a coste/beneficio y tiempo, de trabajar en la actualidad. Se prevé que esta unión permanecerá durante varios años más.

Este tipo de arquitectura junto a la participación de proveedores en la nube ayudan a las empresas, que reciben millones de peticiones, a escalar sus aplicaciones rápidamente y ahorrar costes gracias al pago por demanda.

Hoy en día existen multitud de herramientas para organizar un sistema basado en microservicios. Estos servicios se albergan en contenedores, a su vez, estos necesitan ser organizados y automatizados para su despliegue. Esta función la realizan los orquestadores, y en el mercado se encuentran una amplia oferta: Kubernetes, Rancher, Docker Swarm, Docker-compose, OpenShift…

Kubernetes compite hoy en día en ser el orquestador de contenedores más completo. Lucha por el primer puesto en un mercado donde los entornos de "clustering" y contenedores son muy demandados. Es el orquestador por excelencia (1).

No solo son los distintos orquestadores quienes batallan, sino también los proveedores en la nube más grandes, siendo AWS (Amazon), GCP (Google) y Azure (Microsoft), competidores directos.

2.2 Crítica al estado del arte

En el trabajo de Osuna Fontan A. se realiza la implementación completa de un sistema de Cloud Computing con la tecnología OpenStack, estudiando su arquitectura y realizando una pequeña implantación de ella (2). OpenStack es un proyecto de código abierto de computación en la nube para proporcionar una infraestructura como servicio (3). En su trabajo el autor utiliza un IaaS al igual que en el presente trabajo con GCP.

En el trabajo de Cariñana Abasolo M. se instala, configura y evalúa un servidor basado en un clúster de computadores formado por máquinas virtuales Linux. En este estudio se realiza la misma tarea, pero mediante contenedores de Docker manejados por Kubernetes, una tecnología mucho más ligera y dinámica que las máquinas virtuales (4).

Este proyecto fusiona lo mejor de cada uno de estos estudios. Por un lado, hace uso de un potente IaaS como es GCP y por otro, virtualiza sistemas gracias al uso de contenedores en Kubernetes.

2.3 Propuesta

Este proyecto pretende ilustrar y documentar, cómo y con qué herramientas se puede desplegar un clúster de Kubernetes en Google Cloud Platform.

Las soluciones propuestas forman parte de proyectos de código abierto ya existentes, el objetivo es estudiar y comparar algunas de estas soluciones, centrándonos en: Kubeadm, Kops y Knative. A su vez se compararán las tres entre ellas a nivel de facilidad, consumo de recursos y coste.
También se mencionarán peculiaridades y características de cada una de ellas, con el fin de poder elegir la solución más favorable en cada uno de los casos.

3. Análisis del problema

3.1 Análisis DAFO

Una vez finalizado el estudio sobre el estado del arte sería adecuado realizar un buen análisis del problema o identificación de oportunidades.

Se analiza mediante un análisis DAFO las debilidades, amenazas, oportunidades y fortalezas del proyecto. Este análisis del entorno externo y de las características internas del proyecto, permiten obtener una representación gráfica de estas cualidades.

Se estudian tanto los factores internos como los externos. Los factores internos lo componen las debilidades, qué limitaciones tiene el proyecto, y las fortalezas, las cualidades por las que más destaca el proyecto. Los factores externos en cambio, los constituyen las oportunidades, es decir, las oportunidades que tiene nuestro proyecto frente a otros en el mercado, y las amenazas, qué problemas o situaciones externas pueden afectar a nuestro proyecto.

- Análisis interno:

  ✓ Fortalezas:
  - Se ofrecen distintas soluciones.
  - Se implementa un clúster altamente disponible para asemejar lo más posible un entorno de producción real.
  - No se requiere de un equipo de muchas prestaciones para implementar el proyecto.
  - Las soluciones implementadas son de código abierto.

  ✓ Debilidades:
  - Se requiere un mínimo de conocimiento sobre el tema.
  - Hay que disponer de un crédito o de dinero real para implementar las soluciones en la nube.
  - Alguna solución es tediosa y repetitiva en su instalación.

- Análisis externo:

  ✓ Oportunidades:
  - Se utiliza como IaaS a GCP que provee una infraestructura segura, competente y profesional.
  - Este proveedor en la nube tiene actualmente pocos competidores en el mercado.

  ✓ Amenazas:
  - Ante cualquier contratiempo no existe documentación suficiente para consultar por utilizar tecnologías novedosas.
  - Posible obsolescencia debido al crecimiento exponencial de las tecnologías de computación en la nube.

3.2 Identificación y análisis de soluciones posibles

Hoy por hoy, Kubernetes puede ser ejecutado en tu ordenador en local, en un grupo de clústeres de nuestra propia organización, en proveedores en la nube con máquinas virtuales o usando herramientas de administración de clústeres como lo es Google Kubernetes Engine (GKE) (5).

Como se ha comentado anteriormente, el levantamiento del clúster se realizará de tres maneras distintas. Primero lo haremos con Kubeadm, luego con Kops y más tarde con Knative. Cada una de estas formas ofrece ventajas y desventajas que a continuación se muestran.

Es importante recordar que el análisis de estas tres herramientas no es con el objetivo de identificar cual es mejor o peor, ya que esa decisión depende de los requisitos de la empresa u organización que desea implementarlos.

El objetivo del proyecto es exponer tres diferentes formas de desplegar un clúster Kubernetes altamente disponible en GCP y ver cómo cada una de ellas tiene un valor distinto que aportar.

3.2.1 Despliegue con Kubeadm

Kubeadm es un sistema de código abierto que nos permite despliegues automáticos, escalabilidad y gestión de contenedores de aplicaciones. Nos permite realizar el despliegue de un clúster de Kubernetes de manera sencilla.

Es la mejor herramienta para iniciarnos en el mundo de los clústeres de Kubernetes. Su ventaja es la simplicidad a la hora de crear un clúster mínimamente viable y que funcione de manera sencilla para el usuario. Esta utilidad se caracteriza por su funcionamiento eficiente, su compatibilidad con diversos sistemas operativos y su portabilidad, permitiendo al usuario el despliegue de un clúster en local o en la nube mediante el uso de máquinas virtuales.

Kubeadm automatiza pasos como la emisión y coordinación de certificados de seguridad de cada nodo, y también los permisos necesarios para el control de acceso basado en roles.

Se pretende que Kubeadm sea un componente compositivo de herramientas de nivel superior, es decir, que sirva de base para implementar herramientas superiores sobre ella.

Kubeadm es una excelente herramienta si se necesita una forma sencilla de probar Kubernetes, una manera para usuarios de automatizar la configuración de un clúster y testear su aplicación, o una base en otro ecosistema con un enfoque más amplio.

Se puede instalar y usar Kubeadm en varias máquinas: como un ordenador portátil, un conjunto de servidores en la nube, una Raspberry Pi, etc (6).

Para instalar Kubeadm se necesitará:

- Una o más máquinas corriendo Linux OS.
- 2 GiB o más de memoria RAM.
- Al menos 2 CPUs en la máquina que se use a modo de nodo master (se explicará más adelante).
- Conectividad de red entre todos los nodos del clúster.

3.2.2 Despliegue con Kops

Kops, abreviatura de Kubernetes Operations, es un proyecto oficial de Kubernetes de código abierto que nos permite crear, mantener, actualizar y borrar un clúster de Kubernetes a nivel de producción de alta disponibilidad (7). Actualmente, en su última versión 1.24.1, esta soportado por AWS y soporta DigitalOcean, GCP y OpenStack en fase beta.

Las ventajas que ofrece Kops, son que automatiza el provisionamiento de alta disponibilidad en un clúster, permite el uso de Terraform e incluso soporta add-ons.

Kops utiliza Kubeadm, por lo que no hay necesidad de elegir entre ambas herramientas, simplemente Kops es una implementación de Kubeadm con algunas mejoras.

3.2.3 Despliegue con Knative

Knative es un proyecto de la comunidad “open source” que incorpora ciertos elementos en Kubernetes, los cuales permiten implementar, ejecutar y gestionar aplicaciones “serverless” y en la nube. Es una extensión de Kubernetes que facilita y simplifica algunas acciones.

Mientras que Kubernetes requiere que los desarrolladores tengan que realizar tareas repetitivas, como hacer un “pull” del código de la aplicación y aprovisionarlo con un contenedor y una imagen para luego configurar las conexiones con el exterior, Knative facilita el trabajo. Gracias a la automatización de estas tareas, un desarrollador podrá definir el contenido y la configuración de un contenedor en un único fichero YAML y la herramienta hará el resto (8).

Knative está formado por dos componentes: “Serving” y “Eventing”. En este proyecto se instala el componente “Serving” que se encarga de los detalles de implementación de redes, autoescalado y uso y seguimiento del concepto de revisiones. De esta forma, los equipos pueden centrar su trabajo en la lógica de las aplicaciones utilizando cualquier lenguaje de programación deseado.

El autoescalado de Knative supone una novedad ya que implementa una solución que permite un escalado a 0 de nuestras aplicaciones, a diferencia de los autoescaladores

convencionales. Esto conlleva que se pague solo por lo que se usa y por ende se minimizan costes.

En Knative existe la posibilidad de realizar revisiones de las aplicaciones. Cada vez que se realice una actualización de una aplicación se creará una revisión, pero no borrará la versión antigua de la aplicación. Esto permite a los desarrolladores probar esta nueva versión de cara a los usuarios, dividiendo el tráfico de peticiones mediante “Traffic Spliting” (9).

3.3 Presupuesto

Para establecer un presupuesto en el caso más sencillo, se debe primero diferenciar entre coste de recursos humanos y coste material.

A) Recursos humanos

Los recursos constituyen las horas-hombre empleadas en el proyecto. En este trabajo las horas empleadas se han dividido de la siguiente forma:

- Aprendizaje tecnología
- Implementación tecnología
- Solución errores tecnología
- Documentación tecnología

B) Coste material

Para calcular el coste material total habría que sumar los costes de:

B.1) Implementación en local

Para la implementación de la solución local (Kubeadm en local) se ha hecho uso de un hipervisor (VirtualBox) y de la creación de 3 máquinas virtuales. Supone tener un ordenador de al menos 8GB de RAM y un procesador de 4 núcleos si queremos que la solución funcione de forma fluida. Por lo que se utilizó un ordenador con un procesador i5 de novena generación con 16 GB de RAM.

B.2) Implementación en la nube

Para la implementación de las soluciones en la nube (Kubeadm, Kops y Knative en GCP) GCP no limita a los usuarios a nivel de recursos, por lo que se puede hacer uso de máquinas realmente potentes. Como se paga por lo que se usa, esto supone un límite monetario que lo tendrá que establecer el usuario que alquila la máquina. GCP otorga a las cuentas recién creadas un crédito de regalo de 300.00€ para su libre uso dentro de la plataforma.

GCP ofrece una calculadora de precios donde se pueden consultar las tarifas actualizadas de los servicios y productos que ofrece (10).

4. Diseño de la solución

Con el fin de contextualizar un poco la tecnología, a continuación, se explicará qué arquitectura sigue esta.

No solo se hablará sobre el patrón maestro-esclavo que sigue el sistema, sino también de qué componentes la forman y de cómo interactúan unos con otros.

4.1 Patrón de diseño

Kubernetes está basado en el patrón de diseño maestro-esclavo. Se trata de un modelo de arquitectura distribuida centralizado, en el que contamos con una organización jerárquica entre las máquinas o nodos.

Se llama nodo maestro o “master” o Plano de Control (en Kubernetes) a la máquina o máquinas que coordinan el trabajo de otras máquinas. Se llama esclavo o “worker” a la máquina que realiza el trabajo que el nodo maestro le asigna. El nodo maestro sirve como centro de comunicación entre los nodos trabajadores y el sistema que solicita el resultado, ya sea un usuario o un proceso.

Este patrón se utiliza cuando existen dos o más procesos que deben ejecutarse de forma simultánea y continua. Comúnmente se utiliza cuando, por un lado, se responde a los controles de la interfaz de usuario (master), y por el otro, se ejecutan otros procesos simultáneamente (workers). Asimismo, resulta muy ventajoso cuando se utiliza en arquitecturas de microservicios (11).

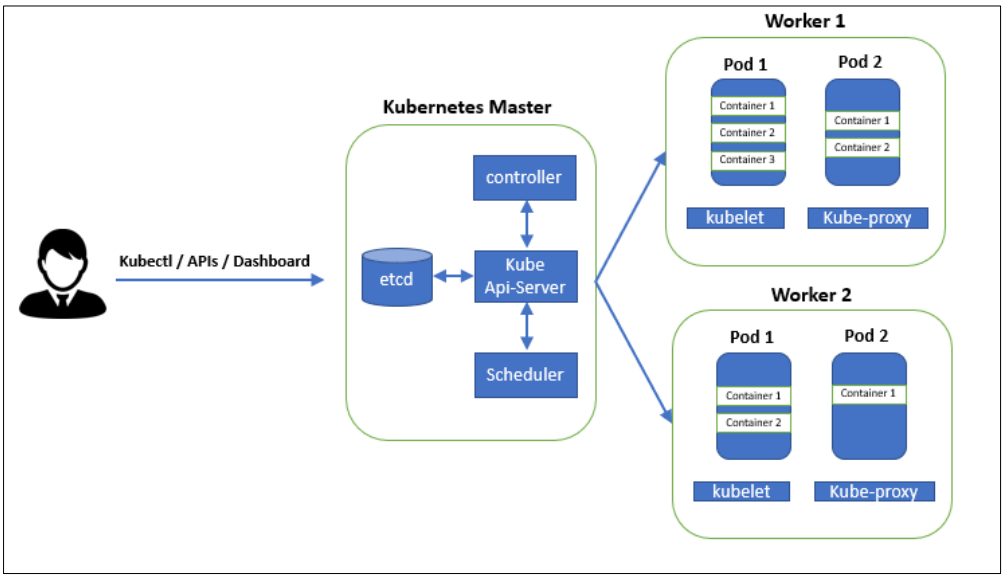

Figura 1. Patrón de diseño maestro-esclavo de Kubernetes. Tomada de Geekflare (12).

Algunas de las ventajas de esta arquitectura es la escalabilidad, ya que se puede aumentar el tamaño del sistema añadiendo más nodos trabajadores a la red. Y, por otro lado, la tolerancia a fallos, ya que si uno de los nodos trabajadores falla el sistema es

capaz de recuperarse repartiendo el trabajo del nodo afectado al resto de trabajadores mientras este trata de volver a estar disponible.

Este modelo también tiene algunas desventajas, ya que el nodo master es el único punto de contacto (SPOC – single point of contact) (13) y a la vez el único punto de fallo (SPOF – single point of failure) (14). Si el nodo maestro falla, perderemos todo el sistema. Por ello, es conveniente que se respalde este modelo con un mecanismo multi-maestro en el que existen otros nodos maestros para reemplazar al que esté operativo en caso de fallo, esto aporta disponibilidad al modelo.

4.2 Arquitectura multimaestro

Para una arquitectura multimaestro, es importante saber que los nodos maestros no se deben configurar todos en la misma zona o región, sino que lo que interesa es repartirlos en distintas ubicaciones. Ya que, dada la circunstancia de un evento que causara la caída del master (por ejemplo: un corte de luz, un mantenimiento de internet…) y que afectara en la localización del nodo con el que se está contactando, el resto de las zonas pudieran continuar intactas y ofrecer el servicio mientras este se recupera.

Los nodos maestros, dentro de una arquitectura multimaestro, se envían “heartbeats” o latidos de corazón que son unos paquetes de información que se trasmiten para manifestar su actividad entre ellos. En el momento en el que no se detecta uno de estos, se entiende que un nodo ha fallado, por lo que se somete el clúster a quorum. Un quorum es un sistema de votaciones de los nodos másteres de un clúster, en donde se decide si el clúster merece seguir en funcionamiento o si, por el contrario, se debería paralizar. Si se gana el quórum, el clúster sigue en funcionamiento, y si se pierde, se detiene hasta que el nodo que ha fallado se recupera. Es por esto por lo que es importante tener un número impar de nodos másteres, ya que se otorga quórum siempre a la mitad de los nodos más uno.

Cabe mencionar que este algoritmo es mucho más complejo en su profundidad, aunque en este trabajo, se menciona una pequeña parte de él.

4.3 Arquitectura de Kubernetes

En Kubernetes se utilizan los objetos de la Interfaz de Programación de Aplicaciones (API) de Kubernetes para describir el estado deseado del clúster: qué aplicaciones se quieren ejecutar, qué imágenes de contenedores usan, el número de réplicas, qué red, etc. Se especifica el estado deseado del clúster mediante la creación de objetos usando la API de Kubernetes, típicamente mediante la interfaz de línea de comandos `kubectl`.

La sintaxis para ejecutar comandos de `kubectl` es la siguiente (15):

```
kubectl [command] [TYPE] [NAME] [flags]
```

donde `[command]`, `[TYPE]`, `[NAME]` y `[flags]` son:

- `command`: Especifica la operación que se va a realizar en uno o más recursos, por ejemplo: `create`, `get`, `describe`, `delete`.
- `TYPE`: Especifica el tipo de recurso (Pod, Deployment, Service…).
- `NAME`: Especifica el nombre de un recurso.
- `flags`: Especifica configuraciones adicionales y opcionales.

Una vez que se especifica el estado deseado, el Plano de Control de Kubernetes realizará las acciones necesarias para que el estado actual del clúster coincida con el estado deseado. Para ello, Kubernetes realiza diferentes tareas de forma automática, como pueden ser: parar o arrancar contenedores, escalar el número de réplicas de una aplicación dada, etc.

El Plano de Control de Kubernetes mantiene un registro de todos los objetos de Kubernetes que existe en el sistema y ejecuta continuos bucles de control para gestionar el estado de estos. Los bucles responderán a los cambios que se realicen en el clúster y ejecutarán las acciones necesarias para hacer que el estado actual logre el estado deseado.

Un claro ejemplo es cuando se usa la API de Kubernetes para crear un Service (definido más adelante), se está proporcionando un nuevo estado deseado para el sistema. El Plano de Control de Kubernetes registrará la creación del objeto y llevará a cabo las instrucciones ejecutando las aplicaciones requeridas en los nodos del clúster, haciendo de esta manera que el estado actual coincida con el estado deseado.

Los objetos básicos de Kubernetes incluyen (5,16):

➢ Pod

Es un grupo de uno o más contenedores, con almacenamiento y red compartidos, y unas especificaciones de cómo ejecutar los contenedores. Representa el bloque de construcción básico de Kubernetes.

En vez de desplegar contenedores individualmente, se deberá desplegar y operar siempre sobre un Pod de contenedores.

Dado que un contenedor siempre debe ejecutar un único proceso, mediante los Pods se podrán incluir varios contenedores, ejecutando varios procesos diferentes en una única estructura y organizarlos más cómodamente.

➢ Deployment

Un Deployment es un objeto de Kubernetes que puede representar una aplicación en un clúster. Su principal ventaja es que permite realizar especificaciones en su fichero sobre la aplicación como son el número de réplicas o la versión de la imagen que se quiere correr.

➢ Service

Una vez desplegado el Deployment, si se quiere exponer nuestra aplicación al mundo y que cualquier persona pueda acceder a ella, se tendrá que crear un Service. Un Service o servicio puede ser de varios tipos:

- ClusterIP: Permite que el servicio solo pueda accederse desde dentro del clúster.
- NodePort: Permite que el servicio pueda accederse desde el exterior, pero fijando manualmente la IP y el puerto.
- LoadBalancer: Permite que el servicio pueda accederse desde el exterior usando el balanceador de carga del proveedor en la nube.
- External Service/External Name: Se usa particularmente para mapear el servicio con alguna base de datos que exista fuera del clúster.

➢ Namespace

El Namespace o espacio de nombres constituye un entorno dentro del espacio del clúster donde poder realizar labores sobre un campo de acción específico. Ofrece la capacidad de dividir y aislar un conjunto de recursos determinados del clúster.

4.3.1 Arquitectura del master

La función del master es la de intermediario entre usuario y clúster. Está formado por los siguientes componentes que almacenan y administran el estado del clúster.

Los componentes son (16):

➢ *kube-apiserver*

El servidor de la API es el componente del Plano de Control de Kubernetes que expone la API de Kubernetes. Se trata del “frontend” de Kubernetes, recibe las peticiones y actualiza acordemente el estado en el *etcd*.

Es el componente central usado por todos los otros componentes y clientes.

➢ *etcd*

Es un almacén de datos persistente, consistente y distribuido de clave-valor utilizado para almacenar toda la información del clúster de Kubernetes.

- *kube-scheduler*

El planificador *kube-scheduler* es el componente del Plano de Control que está pendiente de los Pods que no tienen ningún nodo asignado, y selecciona uno donde ejecutarlos.

Para decidir en qué nodo se ejecutará el Pod, se tienen en cuenta diversos factores: requisitos de recursos, restricciones de hardware/software/políticas, afinidad y antiafinidad, localización de datos dependientes, entre otros.

- *kube-controller-manager*

Es el componente del plano de control que ejecuta los controladores de Kubernetes.

Lógicamente, cada controlador es un proceso independiente, pero para reducir la complejidad, todos se compilan en un único binario y se ejecuta en un mismo proceso.

Estos controladores incluyen:

- Controlador de nodos: es el responsable de detectar y responder cuándo un nodo deja de funcionar.
- Controlador de replicación: es el responsable de mantener el número correcto de Pods para cada controlador de replicación del sistema.
- Controlador de endpoints: construye el objeto Endpoints, es decir, hace una unión entre los Services y los Pods.
- Controladores de tokens y cuentas de servicio: crean cuentas y tokens de acceso a la API por defecto para los nuevos Namespaces.

4.3.2 Arquitectura del worker

Los nodos workers se encargan de correr los contenedores. Están compuestos por los siguientes elementos (16):

- *kubelet*

Agente que se ejecuta en cada nodo de un clúster. Se asegura de que los contenedores estén corriendo en un Pod.

El agente *kubelet* toma un conjunto de especificaciones de Pod, llamados PodSpecs, que han sido creados por Kubernetes y garantiza que los contenedores descritos en ellos estén funcionando y en buen estado.

- *kube-proxy*

Kube-proxy permite abstraer un servicio en Kubernetes manteniendo las reglas de red en el anfitrión y haciendo reenvío de conexiones.

- *Runtime* de contenedores

El runtime de los contenedores es el software responsable de ejecutar los contenedores. Kubernetes soporta varios de ellos: Docker, containerd, cri-o, rktlet y cualquier implementación de la interfaz de runtime de contenedores de Kubernetes, o Kubernetes CRI.

Todos estos componentes corren como procesos individuales y sus dependencias vienen definidas en la figura 2.

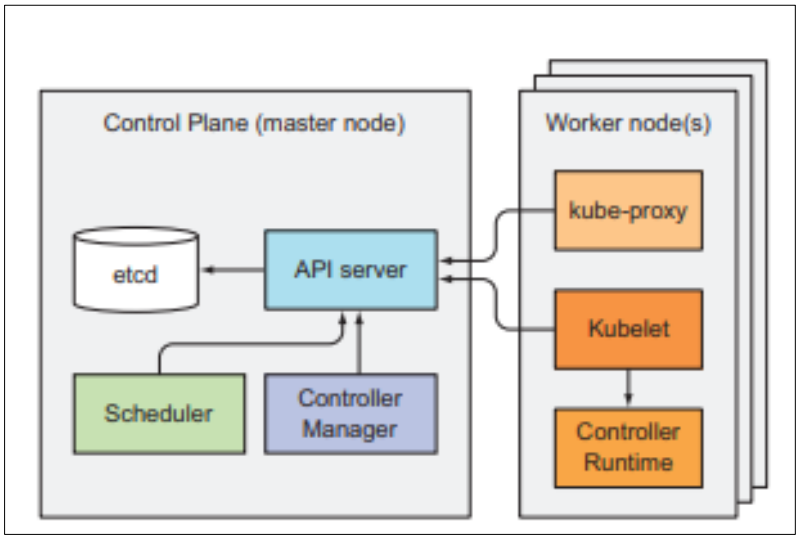

Figura 2. Componentes del Plano de Control (master) y nodos workers. Tomada de Kubernetes in Action (5).

5. Desarrollo de la solución

Una vez realizada la fase de análisis y de ver cómo está distribuida la arquitectura de la tecnología, es hora de la puesta en marcha del desarrollo de las soluciones.

En este apartado se documentará cómo ha sido el desarrollo de cada una de las soluciones. Previamente, ha sido necesario desarrollar una serie de actividades para poder comprender con profundidad la arquitectura de Kubernetes.

Se expondrán estas actividades porque se consideran de vital importancia para el desarrollo de las soluciones finales y su comprensión.

5.1 Despliegues preliminares

En esta sección se desarrollará cómo se ha realizado el despliegue de un clúster Kubeadm en local, la implementación del balanceador de carga MetalLB y la implementación de un NFS.

5.1.1 Despliegue de un clúster Kubernetes con Kubeadm en local.

Se pone en práctica el despliegue basándose en el trabajo de Matagne M.(17) .

El objetivo de este primer paso es el de desplegar un clúster Kubernetes con Kubeadm que sirva una simple página de HTML usando una imagen de NGINX.

El clúster que se va a desplegar consiste en 1 nodo master y 2 nodos workers, donde cada uno de los nodos será una máquina virtual distinta.

Hay que tener en cuenta que, este despliegue tiene el inconveniente de tener un SPOF, ya que solamente se va a desplegar un único nodo master y en caso de fallo, se podría perder cualquier comunicación con el clúster.

Especificaciones

Primero, se deberá elegir el hipervisor donde se crearán las 3 máquinas virtuales.

Un hipervisor es un software que crea y ejecuta máquinas virtuales cediendo recursos del host que lo ejecuta a cada una de ellas. Permite gestionar los recursos utilizados y además aísla su sistema operativo, que puede ser a elección del usuario.

En este caso, se ha decantado por el uso de Oracle VirtualBox 6.1 debido a la familiarización con el sistema. El sistema es muy intuitivo y sencillo de utilizar.

Las especificaciones de las 3 máquinas virtuales son las siguientes:

Nodo Master:

- 2GB de RAM
- 2 procesadores virtuales
- CentOS7 Minimal como sistema operativo
- Disco duro de 10GB reservado dinámicamente
- Adaptador de red conectado a una red NAT, con IP 192.168.1.5

Nodos Workers:

- 1GB de RAM
- 1 procesador virtual
- CentOS7 Minimal como sistema operativo
- Disco duro de 10GB reservado dinámicamente
- Adaptador de red conectado a la misma red NAT, con IPs 192.168.1.6 y 192.168.1.7 respectivamente.

Configuración de las máquinas

Antes de instalar Kubeadm en las máquinas, y dado que se ha instalado CentOS7 Minimal como sistema operativo (una versión con las configuraciones mínimas de CentOS7), se debe configurar adecuadamente y asignarles sus correspondientes IPs.

En primer lugar, si se desea, es posible la asignación de *hostnames* adecuados para las máquinas. A continuación, se asignan las direcciones IPs a cada una de ellas, para ello se modifica el siguiente fichero:

```
# [k8smaster, k8snode1, k8snode2]

$ vi / etc / sysconfig / network - scripts / ifcfg - enp0s3
```

El comando “nmcli con” sirve para comprobar el nombre de la red nat que en este caso es enp0s3. Ahora, se edita el archivo de configuración y se le agrega: la dirección IP, la máscara de red y la dirección de DNS. Cada una de las máquinas debe tener su propia dirección IP dentro del rango de direcciones que se ha establecido en la red NAT de VirtualBox. Se han establecido las siguientes direcciones para cada máquina:

- 192.168.1. 5 para el nodo master
- 192.168.1. 6 para el nodo worker1
- 192.168.1. 7 para el nodo worker2

El resto del archivo se debe dejar con sus configuraciones predeterminadas.

A continuación, para que las máquinas puedan contactar entre ellas fácilmente, existe la posibilidad de editar el fichero de configuración de hosts y de añadir sus nombres junto a sus direcciones IPs correspondientes.

Para evitar copiar manualmente cada fichero existe la opción de utilizar el comando "scp" que permite al usuario copiar un fichero de una máquina a otra.

Para comprobar si se han seguido bien los pasos y que todo funciona correctamente se usa el comando "ping" para comunicar con cualquiera de las máquinas.

También se debe deshabilitar SELinux, que es una barrera de seguridad de Linux.

Instalación del clúster Kubernetes

Una vez configuradas las maquinas, se puede empezar con la instalación del clúster.

Se configura el fichero "/etc/yum.repos.d/kubernetes.repo", donde se debe especificar el repositorio para poder instalar Kubeadm. El contenido del fichero debe ser el siguiente:

```
# [k8smaster, k8snode1, k8snode2]

[kubernetes]

name=Kubernetes

baseurl=https://packages.cloud.google.com/yum/repos/kubernetes
-el7-x86_64

enabled=1

gpgcheck=1

repo_gpgcheck=1

gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg \

https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
```

Una vez especificado el repositorio se puede instalar y activar Kubeadm y Docker en las máquinas. Seguidamente, se tiene que desactivar el swap, que se encuentra dentro del archivo "/etc/fstab"

Inicialización del clúster

En esta sección se trabaja únicamente con el nodo master. Como ya se ha instalado Kubeadm y Docker en todas las máquinas (master y workers), es desde el master donde se debe inicializar el clúster.

Para ello se utiliza este comando:

```
$ kubeadm init
```

La ejecución del comando deja una larga traza impresa en la salida estándar de la consola. Al final de esta, se pide que se guarde el comando que comienza por “kubeadm join …”. Este comando se tiene que guardar en un fichero y enviarlo a los workers, ya que hará falta más adelante.

Por otro lado, la salida estándar también exige que se ejecuten una serie de comandos para ofrecer permisos de administrador al usuario que está inicializando el clúster:

```
$ mkdir -p $HOME/.kube

$ cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ chown $(id -u):$(id -g) $HOME/.kube/config
```

Después de unos minutos, se podrá ejecutar el comando “kubectl get nodes” y, si todo ha salido bien, deberá mostrar el listado de los nodos dentro del clúster que por ahora debe ser de 1.

Creación de la red de Pods

A continuación, se tiene que crear una red de Pods para que los workers puedan conectarse entre sí. Para ello, se hace uso de Calico que, para su instalación, basta con descargar y aplicar el manifiesto YAML de la documentación oficial.

Se comprueba que todos los Pods del clúster y los de Calico estén en funcionamiento:

```
$ kubectl get pods -n kube - system
```

Una vez que todos los Pods están en estado “Ready”, es momento de agregar los workers al clúster. Para ello, se accede a los nodos workers y se ejecuta el comando que comienza con “kubeadm join” en cada uno de ellos, que previamente se ha guardado en un fichero aparte como se ha indicado anteriormente. Es importante escribir la dirección IP respectiva para cada worker en el comando.

Ahora en el master se ejecuta de nuevo “kubectl get nodes” donde se debe mostrar los nodos workers y se debe esperar a que su estado pase a “Ready”.

Despliegue de servidor NGINX

Una vez desplegado el clúster se puede pasar a crear un primer servidor NGINX.

Existen 2 formas de desplegar servicios en los clústeres de Kubernetes: de forma declarativa o de forma imperativa. Es recomendable que se use siempre la primera forma ya que permite visualizar el manifiesto o fichero YAML que contiene las configuraciones del servicio. Tener la capacidad de visualizar el fichero YAML es importante antes de aplicarlo al clúster para evitar posibles actos fraudulentos.

En este caso concreto se usa la forma imperativa para comprobar su funcionalidad, más adelante, el resto del desarrollo se realizará de forma declarativa, ya que constituye buenas prácticas.

En esta instalación, se opta por desplegar un Deployment de una imagen NGINX de forma imperativa con el siguiente comando:

```
$ kubectl create deployment nginx-app --image=nginx \

--replicas=2
```

Este comando crea un Deployment llamado “nginx-app” con la imagen de “nginx” sobre 2 Pods.

Una vez creado el Deployment se crea un Service para exponer el servidor con el fin de que se pueda acceder a él:

```
$ kubectl expose deployment nginx-app --port=80 \

--type=LoadBalancer
```

Con “kubectl get services” se pueden ver los servicios desplegados hasta ahora en el clúster, aquí se muestra el servicio que se acaba de crear con su correspondiente IP y puerto.

Se puede acceder a la página desde la consola utilizando el comando “curl” con la IP y puerto correspondiente al servicio y, si todo ha ido bien, deberá devolver la página HTML de NGINX.

5.1.2 Implementación del balanceador de carga MetalLB.

A continuación, se muestra la instalación de un balanceador de carga en el clúster desplegado anteriormente en local (18).

Un balanceador de carga o *LoadBalancer* es un tipo de servicio (Service) de Kubernetes que sirve de intermediario entre el tráfico recibido y los workers. El balanceador de carga, distribuye el tráfico entre los nodos libres y aparte ofrece una IP pública para poder acceder a los servicios.

Los proveedores en la nube tienen un balanceador de carga implementado por defecto, por lo que, si se crea un servicio de tipo LoadBalancer, el proveedor se encargará de otorgar una IP pública y accesible desde el exterior del clúster.

En este caso, como no se está trabajando con ningún proveedor en la nube todavía, si no que se está ejecutando un clúster en local, se precisa de implementar un balanceador de carga por cuenta propia. Para ello, se usa MetalLB que es un balanceador de carga que permite exponer aplicaciones del clúster que han sido desplegadas en máquinas virtuales.

Instalación del balanceador de carga

Primero, es necesario crear un Namespace donde se deben establecer todos los Pods y dependencias para MetalLB. Todos los objetos y configuraciones que se creen más adelante se instalarán dentro del espacio de nombres.

Una vez creado el Namespace, se descargará y se aplicará el manifiesto directamente desde el repositorio oficial de MetalLB:

```
$ kubectl apply -f
https://raw.githubusercontent.com/google/metallb/v0.12.0/manif
ests/metallb.yaml
```

Configuración del balanceador de carga

Una vez instalado, se debe configurar el rango de IPs que podrá utilizar el balanceador. Para ello, se crea y se aplica un archivo YAML de configuración para MetalLB con el contenido descrito en el Anexo 1.

Para verificar que la instalación ha sido satisfactoria se puede ejecutar el siguiente comando, y comprobar que todos los Pods se encuentran en estado “Running”:

```
$ kubectl get pods -n metallb-system
```

El comando “kubectl get pods” genera un listado de Pods del espacio de nombres “default”. Si se requiere especificar un espacio de nombres concreto, es necesario el uso de “-n” como se muestra anteriormente.

Uso del balanceador de carga en la configuración de NGINX

Para poder poner en práctica el trabajo de MetalLB hace falta crear un Deployment de NGINX sencillo y exponerlo mediante un Service de tipo LoadBalancer. Como se observa en el Anexo 2, se despliega tanto el Deployment como el Servicio en el mismo archivo y de forma declarativa.

Para verificar el correcto funcionamiento de la herramienta se tienen que listar los servicios disponibles y comprobar que el servicio recién creado de NGINX contiene una dirección IP externa.

5.1.3 Implementación de un NFS.

En esta parte se explica cómo se implementa un servidor NAS al que se accede mediante NFS (18). Se utiliza este servidor para almacenar una página web NGINX que posteriormente será accedida por las réplicas que ejecuten el Service. De esta forma, al ser un mismo fichero compartido por los Pods, no se tendrá que alojar en cada Pod la página web. Además, su centralización permitirá realizar cambios de forma rápida y eficaz al solo tener que acceder a una única máquina.

Para implementar este servidor, se utiliza una máquina virtual que sirve como servidor externo al clúster. Este nodo tiene alojada la web y la servirá mediante el protocolo NFS.

Por otro lado, se hace uso del fichero YAML con el Deployment y Service de NGINX de la sección anterior “Implementación de un balanceador de carga con MetalLB” (Anexo 2). A este fichero se le añaden varios objetos y se le modifican algunas opciones para completar la actividad.

Preparación del entorno

Primero se crea una máquina virtual nueva, con IP estática 192.168.1.200, con las mismas especificaciones que un nodo worker:

- 1GB de RAM
- 1 procesador virtual
- CentOS7 Minimal como sistema operativo
- Disco duro de 10GB reservado dinámicamente
- Adaptador de red conectado a la misma red NAT

Instalación y configuración del sistema NFS

Se accede a la máquina nueva y se instala el paquete “nfs-utils”. Luego, se crea el directorio que se exporta y se cambia el usuario al que pertenece:

```
$ mkdir -p /var/web

$ chown nfsnobody:nobody /var/web
```

A continuación, se deben configurar unas flags de SELinux y también, el cortafuegos para que permita el acceso de las peticiones relacionadas con los servicios que utiliza el protocolo NFS, como se ve en el Anexo 3.

Posteriormente, se habilitan los servicios relacionados con NFS los cuales son “rpcbind” y “nfs-server”.

Se modifica el archivo “/etc/exports” añadiendo la siguiente línea:

```
/var/web 192.168.2.200/24(rw,sync,no_root_squash)
```

Y una vez hecho esto y reiniciando el servicio “nfs-server”, se logra exportar la carpeta compartida.

Uso de NAS en Kubernetes mediante volúmenes persistentes

Es importante que todos los nodos cuenten con la utilidad de NFS para que puedan montar el directorio del servidor NAS que se acaba de exportar. Para ello, se debe instalar el paquete “nfs-utils” en estos nodos.

Ahora se comprueba que se puede acceder al directorio compartido mediante el siguiente comando en el nodo master:

```
$ showmount -e 192.168.2.200
```

Para asignar espacio de disco a los Pods, se hará uso de 2 herramientas de Kubernetes. La primera es el volumen persistente o PersistentVolume. Con este objeto se podrá definir diferentes volúmenes que pueden ser posteriormente reclamados por los Pods. Un volumen persistente es similar a un disco físico con sus características (capacidad, permisos, numero de accesos simultáneos…) que espera a ser utilizado.

La otra herramienta es la de asignación/reclamación de volúmenes persistentes o PersistentVolumeClaim. Con ella los Pods tienen acceso a los volúmenes persistentes creados.

Se define el PersistentVolumeClaim a continuación del PersistentVolume en un fichero YAML y se aplican conjuntamente (Anexo 4). De esta forma el Pod que esté corriendo el servidor de NGINX puede acceder al nodo NAS.

Cambio de imagen de nginx para habilitar PHP

El Deployment se hace con una imagen NGINX-PHP. Esta imagen contiene un plugin de PHP que permite ver al usuario la página web con un contenido dinámico. Por lo que se tiene que modificar la línea que define la imagen por esta otra:

```
image: trafex/php-nginx:latest
```

Esta imagen utiliza un directorio diferente para cargar los archivos PHP, por lo que se debe modificar cualquier referencia de “/usr/share/nginx/html/” a “/var/www/html”

A continuación, en el servidor NAS se crea un archivo index.php en la carpeta compartida “/var/web” con el siguiente contenido:

```
<h1>HTML5 Test Page</h1>

<h2>Served by NGINX running on Kubernetes for COS through NFS</h2>

<?php

echo "Served from server ".$_SERVER['SERVER_ADDR']."<br>";

?>
```

Uso de volúmenes persistentes en NGINX

Se modifica el fichero YAML de NGINX para poder asignarle el objeto de tipo PersistentVolumeClaim. Esto importa la carpeta compartida del servidor NAS en el Pod que ejecuta el Service, por lo que NGINX puede mostrar la página creada anteriormente.

Por último, se aplica el fichero YAML del servicio y se accede al Service como de costumbre.

5.2 Despliegue de las soluciones

Lo primero que se ha tenido que realizar para poder desplegar los clústeres en Google Cloud, ha sido una cuenta de Google. Con esta cuenta, no solo se tiene acceso al servicio, si no que al ser nuevos en la plataforma se nos ofrecerá 300.00€ de crédito para gastar.

Parte de los despliegues se han realizado desde el shell del cloud, aunque se puede descargar para ejecutarlo desde local y conectarse iniciando sesión con el cloud, y otra parte desde la interfaz gráfica que ofrece GCP.

5.2.1 Despliegue con Kubeadm en GCP

El procedimiento para este despliegue es muy parecido al de Kubeadm en local. La idea es alquilar 3 máquinas virtuales en GCP y comenzar con el procedimiento usual de Kubeadm (19).

Al igual que con el despliegue en local, hay que tener en cuenta que este despliegue tiene el inconveniente de tener un SPOF, ya que solamente se va a desplegar un único nodo master y la caída del master implicaría la paralización del clúster.

Creación de las máquinas virtuales

Primero, se crea un nuevo proyecto en GCP al que asignarle los recursos, a la vez que se configura la región y zona. Además, se debe crear una Nube Virtual Privada o Virtual Private Cloud (VPC) y una subnet, y configurar las reglas del cortafuegos para las comunicaciones internas y externas (Anexo 5).

Seguidamente, se reserva una IP pública para el nodo master:

```
$ gcloud compute addresses create kubernetes-controller --
region     $(gcloud config get-value compute/region)

$ PUBLIC_IP=$(gcloud compute addresses describe

  kubernetes-controller \

  --region $(gcloud config get-value compute/region) \

  --format 'value(address)')
```

Tras haber realizado estas configuraciones, se pueden crear las máquinas virtuales. En el Anexo 6 se muestra la creación de la máquina virtual que conforma el nodo master, y en el Anexo 7 la de los workers.

Instalación de Docker en los nodos

Una vez creadas y configuradas las máquinas virtuales, se va a instalar Docker en los nodos. Todos estos pasos se tienen que repetir en cada nodo. Se usará apt como gestor de paquetes para esta instalación.

Primero, se instalan los paquetes para permitir que apt pueda usar el repositorio a través de HTTPS. Se agrega la clave oficial GPG de Docker y el repositorio de apt de Docker.

A continuación, se listan todas las versiones disponibles de Docker y se elige la versión que se desee instalar, en este caso se instala la versión 5:19.03.12~3-0~ubuntu-bionic:

```
$ sudo apt-get update && sudo apt-get install -y \
  docker-ce=5:19.03.12~3-0~ubuntu-bionic \
  docker-ce-cli=5:19.03.12~3-0~ubuntu-bionic
$ sudo apt-mark hold containerd.io docker-ce docker-ce-cli
```

Se configura el Daemon con:

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

Por último, se reinicia y habilita Docker.

Instalación de Kubeadm, kubelet y kubectl

Una vez instalado Docker en todas las máquinas, se instala Kubeadm, kubelet y kubectl en los nodos. Todos los siguientes pasos se tienen que ejecutar en todos los nodos, tanto master como workers.

Primero, se agrega la clave de GPG:

```
$ curl -s https://packages.cloud.google.com/apt/doc/apt-
key.gpg | sudo apt-key add –
```

Se añade el repositorio de apt de Kubernetes y se actualiza apt para conseguir los datos del repositorio de Kubernetes.

Igual que con Docker, se listan las versiones disponibles de Kubeadm y se instala la versión compatible deseada, en este caso se instala la versión 1.18.6-00:

```
$ sudo apt-get install -y kubelet=1.18.6-00 kubeadm=1.18.6-00
kubectl=1.18.6-00

$ sudo apt-mark hold kubelet kubeadm kubectl
```

Inicialización del clúster

Una vez terminado de instalar Kubernetes en todos los nodos, se puede inicializar el clúster, este paso puede llevar algunos minutos. Esta vez, solo se ejecutan estos comandos en el nodo master.

Se inicializa el clúster de la siguiente forma:

```
$ KUBERNETES_PUBLIC_ADDRESS=$(gcloud compute instances
describe controller \

--zone $(gcloud config get-value compute/zone) \

--format='get(networkInterfaces[0].accessConfigs[0].natIP)')

$ sudo kubeadm init \

--pod-network-cidr=10.244.0.0/16 \

--ignore-preflight-errors=NumCPU \

--apiserver-cert-extra-sans=$KUBERNETES_PUBLIC_ADDRESS
```

Igual que en la instalación con Kubeadm en local, se recibe un mensaje en la salida estándar de la consola que contiene un comando para posteriormente agregar los workers al clúster. Se debe copiar este comando que empieza por “kubeadm join” para su posterior uso. También, hay que guardar el fichero “kubeconfig” generado por la instalación en el directorio “home” para obtener permisos de administrador.

A continuación, se descarga y se aplica el manifiesto para la red de Pods de Calico desde el repositorio oficial.

Cuando estén todos los Pods en estado “Running”, se comprueba el estado del nodo master (`kubectl get nodes`), que tendrá que haber pasado a estado “Ready”.

Se añade cada worker al clúster inicializado. Todos estos pasos se deben ejecutar únicamente en los nodos workers. Para ello, se hace uso del comando que se ha copiado anteriormente que comenzaba con “kubeadm join”.

5.2.2 Despliegue con Kops en GCP

Kops es una herramienta que permite crear y configurar clústeres de Kubernetes de forma fácil y sencilla. Pese a que Kops está en fase beta para proveedores como GCP y Azure, se puede desplegar un clúster completamente funcional. Es cuestión de tiempo que actualicen los recursos y lancen la fase final del proyecto para Google Cloud (7).

Por otro lado, se implementa la alta disponibilidad en este clúster y más adelante (apartado 7. Pruebas) se prueba su funcionalidad.

Instalación del cliente de Kops y kubectl

Para realizar desde el despliegue de Kops, se tiene que acceder a la consola de Google, e instalar Kops (20).

Para ello, se descarga el cliente desde el repositorio oficial de Kops, mediante los siguientes comandos:

1) Descargar el paquete:

```
$ curl -Lo kops
https://github.com/kubernetes/kops/releases/download/$(c
url -s
https://api.github.com/repos/kubernetes/kops/releases/la
test | grep tag_name | cut -d '"' -f 4)/kops-linux-amd64
```

2) Otorgar permisos de ejecución:

```
$ chmod +x ./kops
```

3) Moverlo para crear una variable de entorno:

```
$ sudo mv ./kops /usr/local/bin/
```

4) Comprobar el funcionamiento con:

```
$ kops version
```

A continuación, se necesita instalar `kubectl`. En este caso ya estaba instalado en la consola de Google, pero si no lo estuviera, se deberá instalar `kubectl` de la misma manera.

Creación de un bucket de almacenamiento

Kops precisa de un almacenamiento externo, para guardar el estado y las configuraciones de un clúster. Para ello, se crea una Google Cloud Storage Bucket en la misma cuenta. Para crear un "bucket" vacío, simplemente se tiene que escribir el siguiente comando, se puede usar cualquier nombre siempre que esté disponible:

```
$ gsutil mb gs://kubernetes-clusters-ernesto/
```

Una vez creado el bucket, se exporta la variable para no tener que escribirla cada vez:

```
$ export KOPS_STATE_STORE=gs://kubernetes-clusters-ernesto/
```

Despliegue del clúster

Finalizados estos pasos de configuración, se puede crear un primer clúster con Kops. Para crear el clúster, se precisa del comando "kops create cluster". Este comando crea un objeto Cluster y un objeto InstanceGroup. Un InstanceGroup o grupo de instancias es un objeto de Kops que administra un conjunto de instancias en la nube, pero que son registradas dentro del clúster como nodos. Estos dos objetos son con los que se trabaja cuando usamos Kops.

Primero, se exporta una variable de entorno PROJECT, que contiene el nombre del proyecto para no tener que escribirlo cada vez:

```
$ PROJECT=`gcloud config get-value project`
```

Y a continuación, se crea el clúster:

```
$ kops create cluster simple.k8s.local

--zones us-central1-a --state ${KOPS_STATE_STORE} /

--project=${PROJECT}
```

La zona se puede elegir entre cualquiera de la lista de zonas que ofrece GCP y si todo ha salido bien, se podrá listar el objeto Cluster con el comando siguiente:

```
$ kops get cluster --state ${KOPS_STATE_STORE}
```

Si listamos los grupos de instancias se puede observar que se han creado dos grupos de instancias, uno con un master configurado y otro con un worker (figura 3):

```
tfgernestogaspar3@cloudshell:~ (proyecto-tfg-361617)$ kops get instancegroup --s
NAME                    ROLE    MACHINETYPE     MIN     MAX     ZONES
master-us-central1-a    Master  n2-standard-2   1       1       us-central1-a
nodes-us-central1-a     Node    n1-standard-2   1       1       us-central1-a
```

Figura 3. Listado de grupos de instancias en Kops.

El comando “kops create cluster” crea el objeto Cluster y los InstanceGroups en el “state store”, pero realmente no ha creado ninguna instancia u objeto en la nube de GCP todavía. Para que se hagan efectivos estos cambios se hace uso del siguiente comando:

```
$ kops update cluster simple.k8s.local --yes
```

Al ejecutar el comando, Kops crea los grupos de instancia en GCP que correrán estas instancias. Esto sirve por si en algún momento alguna instancia del grupo se destruye, GCP la relanzará.

Después de unos minutos, se debe ser capaz de ejecutar “kubectl get nodes”. Llegado este momento el clúster ya está listo.

Alta disponibilidad en Kops

Anteriormente, en el despliegue con Kubeadm se ha comentado que el clúster desplegado tenía el inconveniente de la existencia de un SPOF. En esta ocasión se muestra un despliegue multi-master que evita este problema.

Se migra de un clúster de un solo nodo master a un clúster con una arquitectura multi-master para así conseguir la alta disponibilidad.

En el ejemplo, se incorporan 2 nodos maestros más, para hacer un total de 3 nodos maestros ya que deben ser impares.

Se van a seguir los siguientes pasos:

A) Crear subredes

   Se especifica la subred donde se incorporan los nuevos nodos maestros, en este caso, se definen todos los nodos bajo la misma región (us-central1) y cada maestro en una zona distinta (us-central1-a, us-central1-b, us-central1-c). Para esta ocasión, la subred (us-central1) se especifica en la creación del clúster y dado que se despliegan los nodos en la misma subred, no hace falta modificar el fichero.

B) Crear grupos de instancias maestros

   Se crean los nuevos nodos que se incorporarán como nodos maestros al clúster. Para ello se crea un objeto InstanceGroup en Kops por cada nodo, y se especifica la subnet, la zona donde se quiere la máquina virtual y su tipo, como se ve en la figura 4:

```
apiVersion: kops.k8s.io/v1alpha2
kind: InstanceGroup
metadata:
  creationTimestamp: "2022-09-06T09:04:01Z"
  generation: 1
  labels:
    kops.k8s.io/cluster: simple.k8s.local
  name: master-us-central1-b
spec:
  image: ubuntu-os-cloud/ubuntu-2004-focal-v20220118
  machineType: n2-standard-2
  manager: CloudGroup
  maxSize: 1
  minSize: 1
  nodeLabels:
    cloud.google.com/metadata-proxy-ready: "true"
    kops.k8s.io/instancegroup: master-us-central1-b
  role: Master
  subnets:
  - us-central1
  zones:
  - us-central1-b
```

Figura 4. Descripción de un InstanceGroup maestro.

C) Agregar maestros a la configuración del clúster

Una vez creados los objetos InstanceGroup, se indican estos en el archivo de configuración del clúster. Dentro de este archivo, se encuentra un apartado para especificar qué nodos maestros conforman el clúster.

D) Actualizar el clúster para sincronizar con GCP

Se actualiza el clúster para que Kops pueda desplegar las nuevas instancias que se han creado y además, se realiza un “kops rolling-update cluster” para crear estas instancias también en GCP.

Una vez creadas las instancias y añadidas al clúster, se habrá conseguido un clúster altamente disponible.

5.2.3 Despliegue con Knative en GCP

Para la instalación de Knative, primero se crea un clúster simple de Kubernetes, para luego instalar el cliente de Knative (kn) (21).

Instalación del clúster

Primero, se crea el clúster de Kubernetes, para ello hay que habilitar la API de Kubernetes Engine en nuestra cuenta (22). Si la API no ha sido habilitada aún, aparecerá la siguiente imagen (figura 5):

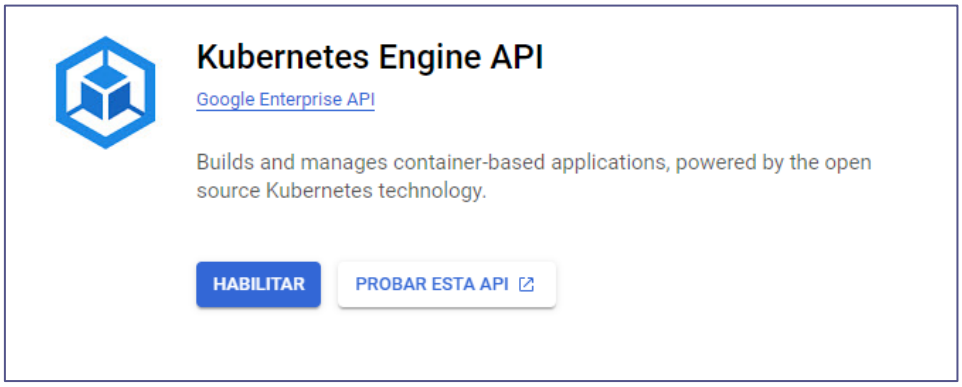

Figura 5. Modal API de Kubernetes Engine. Tomada de GCP Kubernetes Engine (22).

Una vez habilitada la API, se accede a Kubernetes Engine, al apartado de Clústeres y se crea un clúster básico. Después de varias pruebas, esta es la configuración que resulta óptima para esta tecnología:

El clúster estará formado por 2 nodos iguales, uno master y otro worker de:

- 4 vCPUs.
- 4GB de RAM.
- Ubuntu como sistema operativo.
- Disco duro de 10 GB.

Para ello, se hace uso de la creación “Estándar” que ofrece la herramienta, y así poder editar la cantidad de nodos y su configuración al gusto (figura 6):

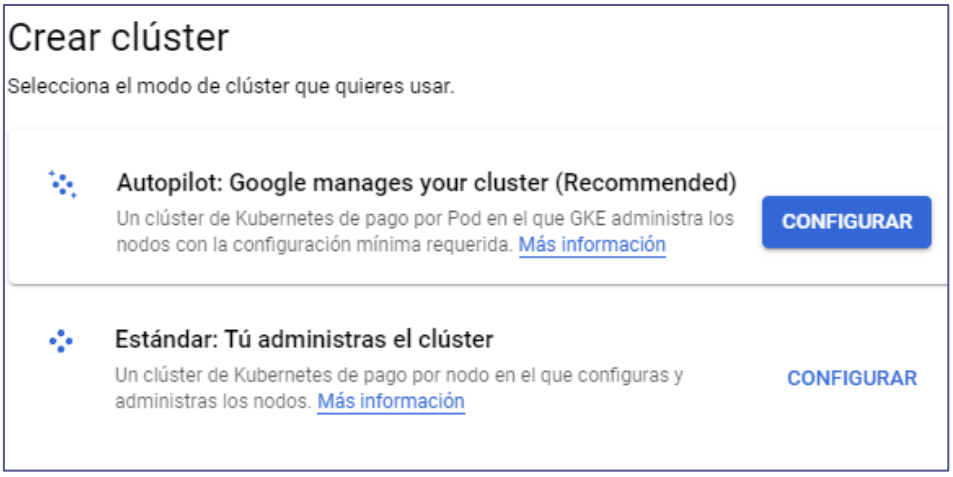

Figura 6. Modal modo de clúster. Tomada de GCP Kubernetes Engine (22)

Se escribe un nombre adecuado para el clúster y dentro de la sección “default-pool”, se tiene que editar el número de nodos a 2, como se muestra en la siguiente figura:

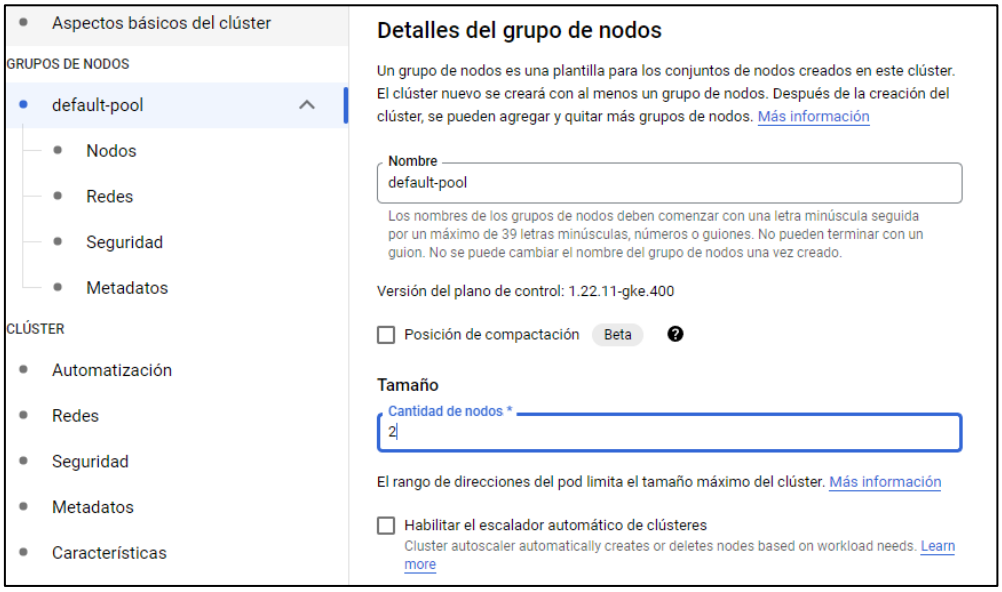

Figura 7 a. Detalle del número de nodos. Tomada de GCP Kubernetes Engine (22).

Seguidamente, en la sección de “Nodos” se edita la cantidad de recursos que se desea para cada nodo, en este caso se elige la configuración “e2-highcpu-4” que aprovisiona a las máquinas de 4 CPUs virtuales y 4GB de memoria cada una, como se ha mencionado con anterioridad:

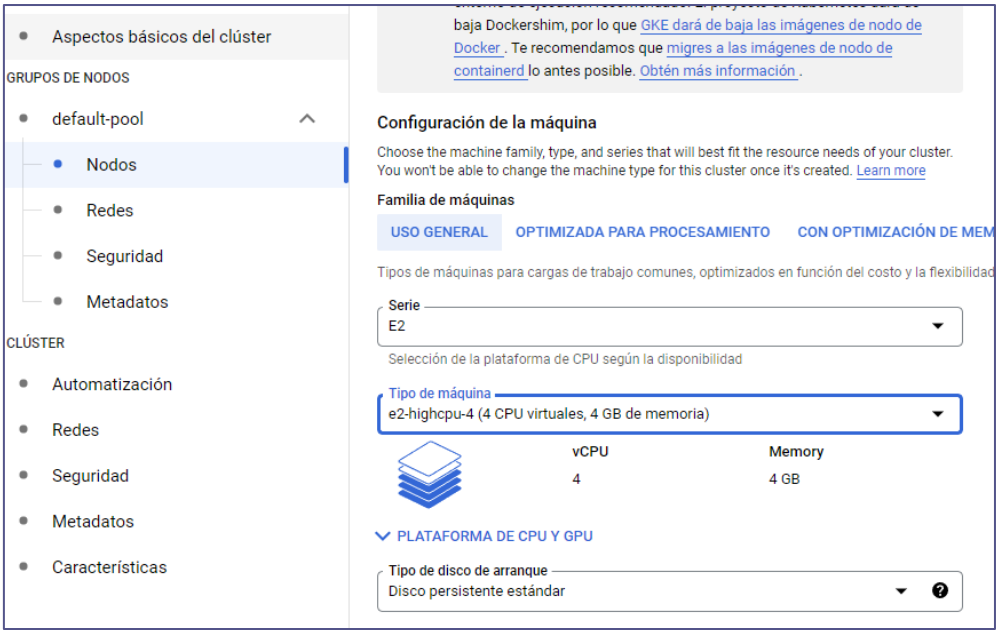

Figura 7 b. Detalle del tipo de máquina. Tomada de GCP Kubernetes Engine (22).

Una vez seleccionados los recursos se pulsa en “Crear” y se espera unos minutos.

Configuración del clúster

Se tiene que acceder al clúster, e instalar el cliente de Knative (kn). Se descarga la versión deseada del repositorio oficial y se actúa como de costumbre. Una vez instalado, se configura Knative Serving, que como se ha explicado anteriormente permite el despliegue de servicios, entre otras ventajas (23). Para ello, simplemente se tiene que aplicar el código del repositorio oficial de Knative, siguiendo las instrucciones que ofrece la página oficial. Adicionalmente, se instala la red de Pods que, en este caso, se utiliza Kourier, ya que es la recomendada para esta instalación.

Después de configurar Knative Serving con Kourier, se pasa a la instalación del DNS, el cual se instala aplicando un simple fichero YAML de la página.

Ya estaría finalizada la instalación de Knative en el clúster de Kubernetes y se puede empezar a desplegar servicios.

5.3 Comparación de las soluciones.

5.3.1 Facilidad.

Kubeadm ofrece una forma de implementación un tanto tediosa, ya que se deben crear manualmente las máquinas virtuales tanto master como worker. Además, en cada máquina se tiene que instalar Docker, Kubeadm, kubelet y kubectl.

Por otro lado, tanto Kops como Knative son responsables del ciclo de vida del clúster, desde el aprovisionamiento de la infraestructura hasta la actualización y la eliminación. Kops resulta algo más problemático puesto que está en fase beta para GCP.

Por lo personal, Knative ha resultado ser una herramienta más fácil de manejar.

5.3.2 Consumo de recursos.

Kubeadm es la solución que consume menos recursos dado que configura un clúster mínimamente viable para su uso, solo con las funcionalidades más básicas que ofrece Kubernetes.

Knative en cambio es la herramienta que más recursos consume de entre las tres, ya que si se quiere implementar algún tipo de servicio se tendrá que instalar Knative Serving. Esto supone la instalación de la totalidad de la herramienta: servicios, rutas, configuraciones y revisiones; de manera que, aunque solo se utilice uno de los servicios te obliga a la instalación de los cuatro, con el consumo de recursos que ello conlleva.

5.3.3 Coste.

El coste de Kubeadm será menor ya que como se ha mencionado anteriormente es la solución que menos recursos consume, frente a Knative y Kops.

El despliegue de Kubeadm se ha realizado con 3 nodos con máquinas de tipo n1-estándar-1 que constan de 1 CPU y 3.75 GB de RAM, cada una.

El despliegue de Kops se ha realizado, según lo recomendado en la documentación oficial, con 3 nodos: una máquina de tipo n1-estándar-1 para el master, que consta de 1 CPU y 3.75 GB de RAM; y dos máquinas de tipo n1-estándar-2 para los dos workers, que constan de 2 CPUs y 7.5 GB de RAM.

Por último, el despliegue de Knative se ha realizado con 2 nodos con máquinas de tipo e2-highcpu-4 que constan de 4 CPUs y 4GB de RAM, cada una.

Kubeadm	Kops	Knative
3 nodos, cada uno de: - 1 CPU - 4 GB de RAM	1 nodo de: - 1 CPU - 3.75 GB de RAM 2 nodos de: - 2 CPUs - 7.5 GB de RAM	2 nodos, cada uno de: - 4 CPUs - 4 GB de RAM
0,0475 $/h * 3	0,0475 $/h + 0,0950 $/h * 2	0,09894 $/h * 2
0,1425 $/h total	0,2375 $/h total	0,19788 $/h total

Tabla de comparativa de precios.

Aplicando las hojas de precios de Compute Engine en la página oficial de Google Cloud, se obtiene el coste estimado (24). Como se observa en la tabla anterior, el despliegue de un clúster con Kubeadm resulta una solución más económica.

6. Implantación de la solución.

En este apartado se presenta la etapa de implantación de la solución. Para ello, se llevarán los despliegues de los clústeres a explotación en cada solución propuesta. A continuación, se muestra cómo se despliega un servicio con una imagen NGINX en Kubeadm y Kops, y en Knative.

Como se conoce, Kops es una herramienta para ayudar a la creación y administración de clústeres Kubernetes. Pero a la hora de crear los servicios e implementar aplicaciones, Kops sigue utilizando `kubectl`, por lo tanto, el despliegue de las aplicaciones demo se hacen de la misma forma tanto en Kubeadm como en Kops.

Por otro lado, Knative tiene su propio componente aparte llamado Knative Serving (explicado más adelante), que necesita ser instalado independientemente para poder desplegar servicios en la herramienta.

6.1 Implantación de una imagen NGINX con Kubeadm y Kops.

Para desplegar un servicio en Kubeadm y Kops, se puede hacer de la misma manera que como se ha desarrollado en la solución de Kubeadm en local. Por lo tanto, es tan sencillo como crear y aplicar un fichero similar al del Anexo 2.

Este fichero contiene un Deployment que se tiene que exponer mediante un Service. Para la demostración se usa el tipo de Service: LoadBalancer, ya que el clúster se encuentra desplegado en un proveedor en la nube que puede otorgar una dirección IP.

Para comprobar que ha funcionado el despliegue del servicio, se ejecuta:

```
$ kubectl get services
```

Este comando muestra la lista de servicios dentro del espacio de nombres “default”:

```
tfgernestogaspar@cloudshell:~/servicios (brave-watch-353922)$ kubectl get services
NAME         TYPE           CLUSTER-IP    EXTERNAL-IP      PORT(S)        AGE
kubernetes   ClusterIP      10.64.0.1     <none>           443/TCP        2d22h
nginx        LoadBalancer   10.64.7.164   35.238.228.251   80:31894/TCP   2m42s
```

Figura 8. Listado de servicios del clúster.

Se puede notar que el proveedor en la nube ha otorgado la IP 35.238.228.251 para el servicio recién desplegado.

Por lo tanto, se puede acceder al buscador de Google y escribir la IP junto al puerto 80 para acceder al servicio:

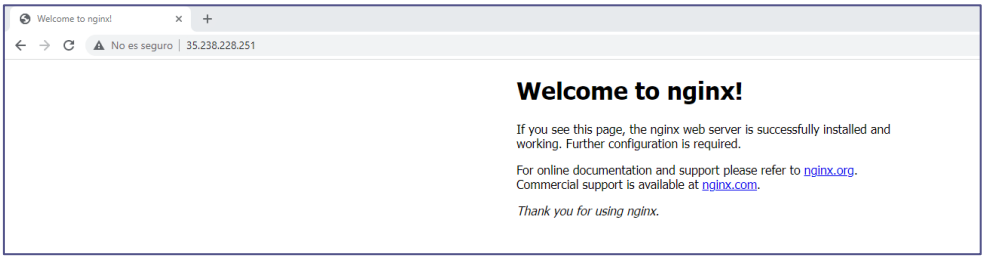

Figura 9. Acceso al servicio de NGINX.

6.2 Implantación de una imagen NGINX con Knative.

Como se explicó anteriormente, Knative es un "wrapper" de Kubernetes por lo que se puede desplegar un servicio de la misma manera: se crea un Deployment y luego se expone mediante un Service. Si se actuara de esta manera entonces no tendría sentido el uso de Knative ya que esta herramienta está pensada para simplificar los despliegues.

Los archivos YAML de servicios en Knative son muy sencillos de desarrollar debido a su sintaxis. A continuación, se quiere desplegar un servicio NGINX mediante un Service. Como se puede observar en el Anexo 8, el fichero YAML no es nada alejado de lo que es un Deployment y un Service de Kubernetes, de hecho, son los dos ficheros fusionados (25).

Gracias a la DNS instalada previamente, Knative es capaz de brindar de una URL en vez de una dirección IP como es habitual. Seguidamente se aplica el fichero como de costumbre y se ejecuta el siguiente comando para poder copiar la URL al servicio:

```
$ kubectl get ksvc
```

```
tfgernestogaspar@cloudshell:~/knative (brave-watch-353922)$ kubectl get ksvc
NAME       URL                                                LATESTCREATED     LATESTREADY
my-nginx   http://my-nginx.default.34.135.215.129.sslip.io   my-nginx-00001    my-nginx-0000
```

Figura 10. Listado de servicios de Knative.

Accediendo al link se tendrá acceso a la página desplegada.

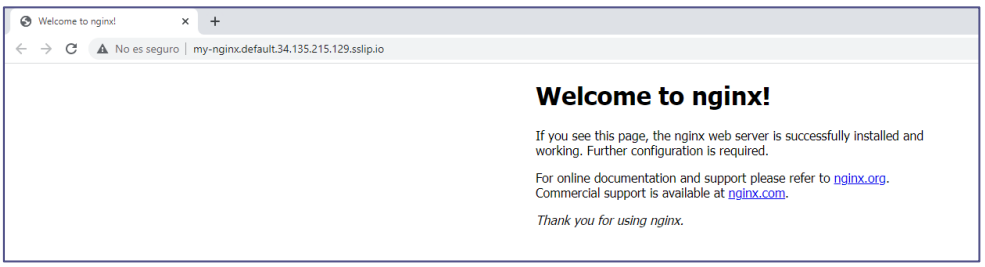

Figura 11. Acceso al servicio de NGINX de Knative

7. Pruebas

En este apartado se presentan las pruebas que se han realizado para verificar que la solución cumple con la propuesta del trabajo. Se muestra cómo este sistema de clúster en la nube es altamente disponible, y se comprueba la eficiencia de los autoescaladores mediante pruebas de carga.

Por otro lado, es interesante conocer qué es y para que se usa la observabilidad. La observabilidad u “obervability” es la habilidad de medir el estado interno de un sistema analizando sus respuestas. Los tres pilares de la observabilidad lo constituyen: “logging”, “monitoring” y “tracing”.

“Logging” se basa en medir los eventos que suceden en un clúster (“logs”). “Tracing” mide el camino (”traces”) que sigue un paquete de información para detectar posibles cuellos de botella. “Monitoring” se basa en analizar las métricas (“metrics”) de un clúster.

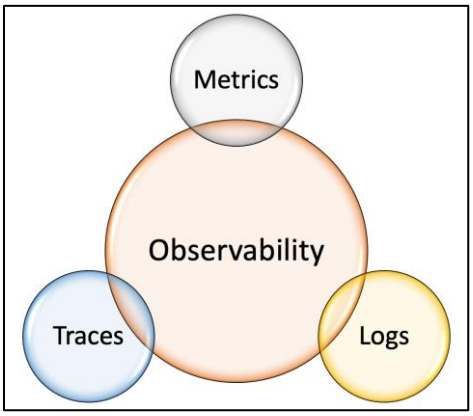

Figura 12. Gráfico de “observability”.

En esta sección también se presenta, una herramienta para monitorizar las métricas de un clúster denominada Prometheus y, se explica su implementación y funcionamiento.

7.1 Prueba de alta disponibilidad

Para probar la disponibilidad de este clúster de Kops se recrea un escenario real donde se interrumpe uno de los nodos master y se comprueba que no se impide la conexión con el clúster.

A continuación, se ilustran los pasos realizados:

Como se ha detallado en el apartado 5. se despliega un clúster con 3 nodos master y un worker:

```
tfgernestogaspar3@cloudshell:~/2servicios (proyecto-tfg-361617)$ kubectl get nodes
NAME                          STATUS   ROLES           AGE    VERSION
master-us-central1-a-fc5q     Ready    control-plane   100m   v1.24.4
master-us-central1-b-2f4h     Ready    control-plane   91m    v1.24.4
master-us-central1-c-sdn3     Ready    control-plane   34m    v1.24.4
nodes-us-central1-a-k3jh      Ready    node            136m   v1.24.4
```

Figura 13. Listado de nodos de clúster en Kops.

Se procede a apagar la instancia de un nodo master llamada “master-us-central1-a-fc5q”. Para hacer esto, se conecta mediante SSH al nodo y se apaga desde dentro mediante el comando “poweroff”:

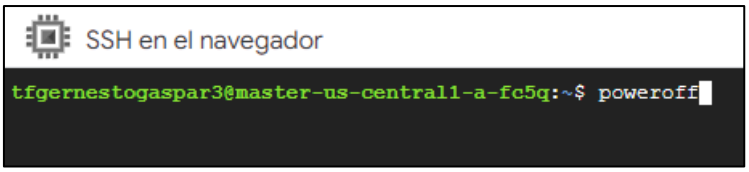

Figura 14. Desconexión de un nodo master.

Después de unos minutos, si se intentan listar el número de nodos se obtiene lo siguiente:

```
tfgernestogaspar3@cloudshell:~/2servicios (proyecto-tfg-361617)$ kubectl get nodes
NAME                          STATUS   ROLES           AGE    VERSION
master-us-central1-b-2f4h     Ready    control-plane   92m    v1.24.4
master-us-central1-c-sdn3     Ready    control-plane   35m    v1.24.4
nodes-us-central1-a-k3jh      Ready    node            138m   v1.24.4
```

Figura 15. Listado de nodos de clúster en Kops tras desconexión de un master.

Esto indica, no sólo que uno de los nodos master ha desaparecido del clúster, si no que, el mero hecho de recibir una respuesta del comando indica que se puede contactar correctamente con el API de Kubernetes y que, por lo tanto, uno de los otros nodos está respondiendo a las peticiones.

Si se desconecta una segunda máquina, se puede comprobar que el clúster pierde el quorum y se paraliza. Por lo que, si se intenta contactar con el clúster no se recibe respuesta alguna:

```
tfgernestogaspar3@cloudshell:~ (proyecto-tfg-361617)$ kubectl get nodes
The connection to the server 104.197.89.141 was refused - did you specify the right host or port?
```

Figura 16. Conexión fallida después de quorum.

Por último, si se inician las máquinas apagadas, estas volverán a unirse al clúster automáticamente y se recuperaría el estado visto en la figura 13.

Evidentemente, este ejemplo esta realizado con un número de nodos menor, pero si se escala a un entorno de producción real (con cientos de nodos), el funcionamiento sería el mismo.

Con esta sencilla prueba, se afirma que efectivamente el despliegue de este clúster es altamente disponible.

7.2 Pruebas de carga

Para que un clúster de Kubernetes pueda responder con eficiencia a toda la demanda recibida, es necesario el uso de autoescaladores. Un autoescalador es una herramienta que, como dice el nombre, permite escalar automáticamente aplicaciones para satisfacer toda la carga que pueda llegar a recibir el sistema.

En el entorno de Kubernetes y sistemas en la nube, existen 3 tipos de autoescaladores: horizontal (HPA), vertical (VPA) y de clúster. En este trabajo se han puesto en práctica el autoescalado horizontal y vertical.

7.2.1 Autoescalado de Pods Horizontal

El autoescalador horizontal de Pods o HorizontalPodAutoscaler (HPA), es una herramienta de Kubernetes que permite revisar la demanda de recursos y escalar automáticamente el número de replicas. De forma predeterminada, el escalador automático de Pods comprueba la API de métricas cada 60 segundos para ver si se requieren cambios en el número de réplicas. Si se define un HPA para una implementación determinada, por ejemplo, un Deployment, se podrá definir el número mínimo y máximo de réplicas que se ejecutarán en base a una métrica personalizada como puede ser el uso de CPU (26).

A continuación, se muestra la instalación de un HPA sobre un clúster de Kubernetes desplegado con Kubeadm (27).

Para instalar el HPA primero se tendrá que instalar un servidor de métricas. Este se encarga de recopilar métricas de recursos, como la utilización de memoria RAM, de cada nodo worker y las expone al servidor de API de Kubernetes.

Para instalarlo basta con descargar el manifiesto del repositorio oficial y aplicarlo a nuestro clúster.

Después de unos minutos, se puede comprobar su correcta instalación mediante:

```
$ kubectl top nodes
```

Este comando devuelve las métricas de los nodos actuales.

A continuación, se tiene que instalar el recurso HPA. Este recurso va sujeto a un Deployment y hay que especificarle:

- La métrica que se desea medir.
- A partir de qué valor de la métrica escalar el servicio.
- Hasta qué número máximo y mínimo de réplicas se pretende escalar.

Este es el manifiesto del recurso que se instalará:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
 name: nginx
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

En el presente trabajo, se le describirá a la herramienta que se le quiere adjuntar a un servicio de NGINX, poniendo de objetivo la utilización de CPU al 50% con un número máximo de réplicas de 10 y un número mínimo de 1. Por lo tanto, cuando se cargue el servicio de NGINX con numerosas peticiones, la herramienta medirá si sobrepasa el 50% de utilización de CPU, si lo hace, creará una réplica y se dividirán la carga. La herramienta creará hasta un máximo de 10 réplicas y cuando la carga descienda a 0, las réplicas se reducirán a un mínimo de 1.

Se ilustrará mejor con un ejemplo.

```
tfgernestogaspar@cloudshell:~/HPA (brave-watch-353922)$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
nginx-6999c774cd-zrwj7   1/1     Running   0          9m17s
tfgernestogaspar@cloudshell:~/HPA (brave-watch-353922)$ kubectl top pods
NAME                     CPU(cores)   MEMORY(bytes)
nginx-6999c774cd-zrwj7   0m           2Mi
tfgernestogaspar@cloudshell:~/HPA (brave-watch-353922)$ kubectl get hpa
NAME    REFERENCE          TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
nginx   Deployment/nginx   0%/50%    1         10        1          6m27s
tfgernestogaspar@cloudshell:~/HPA (brave-watch-353922)$
```

Figura 17. Listado de Pods. Consumo de Pods. Listado de HPAs.

En la captura anterior, se puede observar que el Deployment ha generado 1 replica (como viene especificado en el manifiesto del Deployment) y que actualmente su utilización de CPU está al 0% ya que no está recibiendo ninguna petición.

Ahora se va a generar demanda en el servicio de NGINX y se podrá ver cómo responde el autoescalador a la carga recibida. Para generar numerosas peticiones a la página se ha optado por utilizar el siguiente comando:

```
$ kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c "while sleep 0.0001; do wget -q -O- http://nginx; done"
```

Este comando hará consultas ilimitadas al servicio.

Después de unos minutos el resultado es el siguiente:

```
tfgernestogaspar@cloudshell:~/HPA (brave-watch-353922)$ kubectl get hpa -w
NAME    REFERENCE          TARGETS    MINPODS   MAXPODS   REPLICAS   AGE
nginx   Deployment/nginx   0%/50%     1         10        1          8m58s
nginx   Deployment/nginx   2%/50%     1         10        1          9m16s
nginx   Deployment/nginx   114%/50%   1         10        1          9m46s
nginx   Deployment/nginx   114%/50%   1         10        3          10m
nginx   Deployment/nginx   120%/50%   1         10        3          10m
nginx   Deployment/nginx   21%/50%    1         10        3          10m
nginx   Deployment/nginx   16%/50%    1         10        3          11m
nginx   Deployment/nginx   24%/50%    1         10        3          11m
nginx   Deployment/nginx   19%/50%    1         10        3          12m
nginx   Deployment/nginx   7%/50%     1         10        3          12m
nginx   Deployment/nginx   9%/50%     1         10        3          13m
nginx   Deployment/nginx   18%/50%    1         10        3          13m
nginx   Deployment/nginx   17%/50%    1         10        3          14m
nginx   Deployment/nginx   18%/50%    1         10        3          14m
nginx   Deployment/nginx   4%/50%     1         10        3          15m
nginx   Deployment/nginx   4%/50%     1         10        3          15m
nginx   Deployment/nginx   0%/50%     1         10        2          15m
nginx   Deployment/nginx   0%/50%     1         10        2          19m
nginx   Deployment/nginx   0%/50%     1         10        1          20m
nginx   Deployment/nginx   0%/50%     1         10        1          20m
```

Figura 18. Observación en bucle del listado de HPAs.

Como se puede observar a medida que va aumentando la carga, el número de réplicas van aumentando, en este caso se pasa de 1 réplica a un total de 3. Después, se pausan las consultas por lo que se ve que la carga va disminuyendo y el número de réplicas también.

En la siguiente figura se puede observar cómo a medida que se va generando la carga, se van creando nuevas réplicas del servicio. Se puede apreciar el ciclo de vida de estas 2 nuevas réplicas, desde su creación hasta su eliminación.

```
tfgernestogaspar@cloudshell:~ (brave-watch-353922)$ kubectl get pods -w
NAME                      READY   STATUS              RESTARTS   AGE
nginx-576c8bfb74-dpxnq    1/1     Running             0          11m
loadgenerator             0/1     Pending             0          0s
loadgenerator             0/1     Pending             0          0s
loadgenerator             0/1     ContainerCreating   0          0s
loadgenerator             1/1     Running             0          2s
nginx-576c8bfb74-rnn5v    0/1     Pending             0          0s
nginx-576c8bfb74-rnn5v    0/1     Pending             0          0s
nginx-576c8bfb74-rnn5v    0/1     ContainerCreating   0          0s
nginx-576c8bfb74-xb825    0/1     Pending             0          0s
nginx-576c8bfb74-xb825    0/1     Pending             0          0s
nginx-576c8bfb74-xb825    0/1     ContainerCreating   0          0s
nginx-576c8bfb74-rnn5v    1/1     Running             0          4s
nginx-576c8bfb74-xb825    1/1     Running             0          5s
loadgenerator             0/1     Error               0          5m42s
loadgenerator             0/1     Error               0          5m44s
nginx-576c8bfb74-rnn5v    1/1     Terminating         0          5m38s
nginx-576c8bfb74-rnn5v    0/1     Terminating         0          5m39s
nginx-576c8bfb74-rnn5v    0/1     Terminating         0          5m39s
nginx-576c8bfb74-rnn5v    0/1     Terminating         0          5m39s
nginx-576c8bfb74-xb825    1/1     Terminating         0          10m
nginx-576c8bfb74-xb825    0/1     Terminating         0          10m
nginx-576c8bfb74-xb825    0/1     Terminating         0          10m
nginx-576c8bfb74-xb825    0/1     Terminating         0          10m
```

Figura 19. Observación en bucle del listado de Pods.

7.2.2 Autoescalado de Pods Vertical

Como se ha observado, el autoescalador horizontal de pods funciona correctamente y es muy fácil de instalar, pero no todas las aplicaciones pueden ser escaladas horizontalmente. Para esas aplicaciones, existe otra opción que es el escalado vertical.

El autoescalador vertical de pods o VerticalPodAutoescaler (VPA), es también una herramienta que permite satisfacer la demanda recibida a un servicio de Kubernetes. A diferencia del HPA, VPA no creará nuevas réplicas o Pods idénticos del servicio si no que editará el Pod actual que ejecuta el servicio y lo satisfará añadiéndole más recursos como CPU o memoria RAM.

Dado que un nodo siempre va a poseer más recursos que un Pod, siempre se podrá autoescalar un Pod con un servicio de forma vertical. Kubernetes en cambio, no permite la edición de los recursos de un Pod mientras este está en ejecución por lo que el VPA creará una nueva réplica del Pod con los recursos actualizados y borrará la réplica antigua de menor potencia (5).

A continuación, se mostrará la instalación de un VPA sobre un clúster de Kubernetes desplegado con Kubeadm (28).

Para realizar la instalación, al igual que con HPA, se precisará del servidor de métricas, por lo que se hará uso del que se ha instalado con anterioridad.

Después se clonará el repositorio oficial de VPA y se tendrá que ejecutar el script llamado “vpa-up.sh” ubicado dentro de la carpeta “hack” dentro de la carpeta “vertical-autoscaler”. Este script creará todos los objetos necesarios para el correcto funcionamiento de la herramienta.

Se asegurará que los 3 Pods que constituyen el escalador están en estado “Running”:

```
vpa-admission-controller-67fdd5bbc4-747gv   1/1   Running   0   12h
vpa-recommender-5fff6c4c7f-s8w88            1/1   Running   0   12h
vpa-updater-84896d745b-c6scr                1/1   Running   0   12h
```

Figura 20. Listado de Pods VPA.

Una vez instalados los recursos se va a mostrar un ejemplo de su uso. Para ello, se instalará el Deployment que se puede encontrar en el Anexo 9.

Este Deployment está formado por 2 réplicas de una imagen Ubuntu Slim (una imagen de una versión mínima de Ubuntu). Además, se indica que los recursos usarán 100m de CPU y 50Mi de RAM y que al arrancar ejecutará un comando en bucle para generar carga en el servicio.

Ahora, se instalará el autoescalador vertical de Pods. Este recurso, al igual que el HPA, va sujeto a un Deployment. El manifiesto que se instalará es el siguiente:

```
kind: VerticalPodAutoscaler

metadata:

 name: my-vpa

spec:

  targetRef:

    apiVersion: "extensions/v1beta1"

    kind: Deployment

    name: my-auto-deployment

  updatePolicy:

    updateMode: "Auto"
```

Una vez desplegado el Deployment y el VPA, la herramienta de autoescalado empezará a medir las métricas del Deployment y recomendará unas métricas nuevas.

Si se ejecuta `$ kubectl describe vpa <nombre-vpa>` se podrá ver una descripción de cómo actúa el VPA. La siguiente figura es un extracto del comando:

```
        f:recommendation:
          .:
          f:containerRecommendations:
    Manager:         recommender
    Operation:       Update
    Time:            2022-08-29T11:59:47Z
  Resource Version:  361427
  UID:               06b4a551-7743-417f-b8d0-e6c5c78d6f14
Spec:
  Target Ref:
    API Version:  extensions/v1beta1
    Kind:         Deployment
    Name:         my-auto-deployment
  Update Policy:
    Update Mode:  Auto
Status:
  Conditions:
    Last Transition Time:  2022-08-29T11:59:47Z
    Status:                True
    Type:                  RecommendationProvided
  Recommendation:
    Container Recommendations:
      Container Name:  my-container
      Lower Bound:
        Cpu:     205m
        Memory:  262144k
      Target:
        Cpu:     627m
        Memory:  262144k
      Uncapped Target:
        Cpu:     627m
        Memory:  262144k
      Upper Bound:
        Cpu:     434009m
        Memory:  7960300k
Events:          <none>
```

Figura 21. Recomendación VPA.

El funcionamiento del VPA es de recomendar unas métricas en base a los recursos que va a utilizar el Servicio. Por ejemplo, en la figura se puede observar cómo recomienda para nuestro Servicio un límite mínimo de 205m de CPU y 262144k de RAM (“Lower Bound”) y un límite máximo de 43400m de CPU y 7960300k de RAM (“Upper Bound”). Pero realmente el escalador ha detectado que el Pod funcionará de forma efectiva usando 627m de CPU y 262144k de RAM (“Target”).

No solo recomienda las métricas si no que actualizará el Pod que ejecuta el Servicio de forma automática, eliminando el antiguo (100m CPU y 50mi RAM) y lo actualizará con un Pod nuevo y mejorado (627m CPU y 262144k de RAM) para que así el Pod pueda satisfacer la demanda recibida.

7.3 Herramientas de monitoreo

El monitoreo de Kubernetes se realiza para mantener el estado y la disponibilidad de las aplicaciones o servicios creados en Kubernetes. Es importante utilizar herramientas de monitoreo para realizar seguimientos y analizar el rendimiento o los cuellos de botella de un clúster.

En esta sección se presenta una de las herramientas más utilizadas en el mercado: Prometheus.

El desarrollo de este apartado va a estar formado por las siguientes partes:

- Configuración de Prometheus
- Configuración de AlertManager
- Prueba de regla

7.3.1 Prometheus

Prometheus es una herramienta de monitoreo y alerta para clústeres de Kubernetes Con Prometheus se podrán visualizar gráficas, tablas y estadísticas de los recursos de Kubernetes (29).

Prometheus recoge y almacena sus métricas como datos de series temporales, es decir, la información de las métricas se almacena con una variable temporal que indica en qué momento se registró, junto con pares clave-valor.

Aparte de monitorizar, Prometheus cuenta también con una funcionalidad de alertas que junto a AlertManager (explicado en 7.2.2) conforman la herramienta perfecta de alerta.

En Prometheus las alertas se encuentran supeditas al cumplimiento de una regla. Esta regla se puede definir en base a las métricas que el servidor recoge.

A continuación, se va a describir brevemente la instalación de Prometheus, después se definirá una regla y posteriormente se mostrará el funcionamiento del sistema de “alerting”.

Para hacer funcionar este ejemplo primero se ha tenido que configurar un “Node Exporter”, que no es más que un servicio que expone las métricas de los nodos. Este servicio se despliega mediante el uso de un objeto de Kubernetes llamado DaemonSet.

A continuación, se detallará la instalación de la herramienta para un clúster de Kubernetes en GCP:

Creación de Namespace y ClusterRole

Primero se crea un Namespace donde instalarán todos los componentes de monitoreo y a continuación, se crea un ClusterRole. Mediante este objeto seremos capaces de otorgarle permisos get, list y watch a nuestros nodos, servicios, Pods etc. (Anexo 10).

Creación de Config Map

Seguidamente, se crea un ConfigMap para almacenar las configuraciones de Prometheus (Anexo 11).

Prometheus almacena las reglas en un fichero llamado “prometheus.rules”. En el ejemplo del Anexo 11, se puede observar cómo dentro del fichero se ha establecido una regla que mide la utilización del CPU de los nodos que, si sobrepasa el 80% se disparará. Además, el administrador tiene la posibilidad de añadir una descripción para aportar algo de información adicional.

Creación del Deployment

Se instala el Deployment de Prometheus que define las especificaciones del servicio.
El fichero YAML se puede encontrar detallado en el Anexo 12.

Se expondrá el Deployment mediante un Service de tipo LoadBalancer en el puerto 8080.

Conectar con el Dashboard de Prometheus

Una vez expuesto el servicio de Prometheus, se podrá acceder a su “dashboard” a través de la IP del servicio.

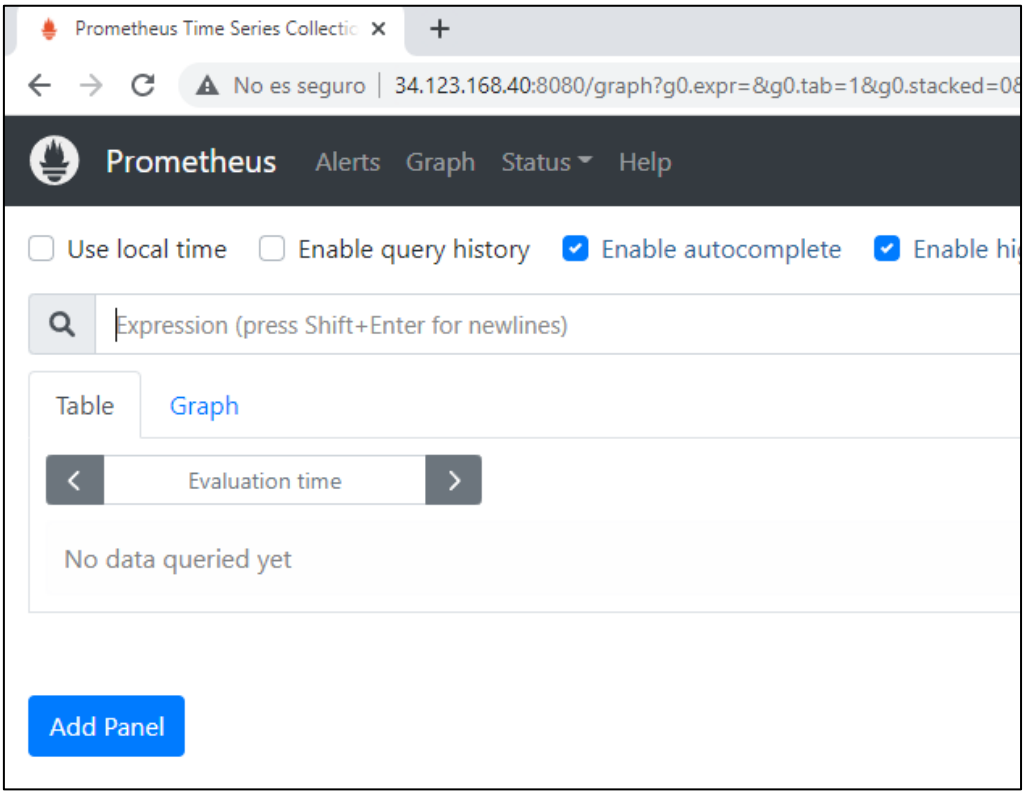

Figura 22. Dashboard de Prometheus.

7.3.2 AlertManager

AlertManager es una herramienta que permite gestionar las alertas de Prometheus. Permite enrutar alertas a receptores vía Email, Slack etc. También es capaz de silenciar e inhibir alertas (30).

Se ejecuta como un servicio aparte de Prometheus, y para enlazarlos, se especifica dentro el archivo ConfigMap de Prometheus, en el apartado de alerting, el “endpoint” del servicio de AlertManager y una vez enlazados, se tendrán que definir reglas en Prometheus para poder empezar a usar la herramienta.

AlertManager también cuenta con su propio “dashboard”, así que, una vez desplegado el servicio, se puede acceder a él para observar las reglas que han sido disparadas.

AlertManager puede avisar si se ha disparado una regla a través de varios métodos y plataformas. En este trabajo se ha configurado para que avise al administrador mediante Gmail. Para que la herramienta pueda enviar correos, necesita ser participe del usuario y contraseña del remitente del correo electrónico. En este caso particular al usar Gmail existe la posibilidad de usar un token generado como contraseña que se ha ocultado por privacidad.

Especificar endpoint en Prometheus

Dentro del fichero Config de Prometheus se tendrá que especificar el endpoint del sistema de alerting de la siguiente forma:

```
  prometheus.yml: |
    global:
      scrape_interval:     15s
      evaluation_interval: 15s
      # scrape_timeout is set to the global default (10s).
    # Alertmanager configuration
    alerting:
      alertmanagers:
      - scheme: http
        static_configs:
        - targets:
           - "alertmanager.monitoring.svc:9093"
```

Hay que reiniciar el servicio de Prometheus cada vez que se haga algún cambio en su archivo ConfigMap, por lo que se tendrá que reiniciar tanto el ConfigMap como el Deployment.

Instalación de ConfigMap de AlertManager

El archivo de ConfigMap de AlertManager tiene el siguiente contenido:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: alertmanager-config
  namespace: monitoring
data:
  config.yml: |-
    global:
    templates:
    - '/etc/alertmanager/*.tmpl'
    route:
      receiver: alert-emailer
      group_by: ['alertname', 'priority']
      group_wait: 10s
      repeat_interval: 30m
      group_interval: 5m

    receivers:
    - name: alert-emailer
      email_configs:
      - to: tfgernestogaspar2@gmail.com
        from: tfgernestogaspar2@gmail.com
        smarthost: smtp.gmail.com:587
        auth_username: tfgernestogaspar2@gmail.com
        auth_identity: tfgernestogaspar2@gmail.com
        auth_password: ************
```

Como se observa en el archivo, existe un apartado para especificar las credenciales del sistema de correo que se quiere utilizar. En este caso se ha configurado el sistema de alerta para que envíe un correo de Gmail con emisor y receptor a la misma persona.

Instalación de ConfigMap de AlertTemplate

Por otro lado, se especifica que estructura y estilo contendrá el correo electrónico enviado mediante un archivo ConfigMap. Este archivo está constituido por una plantilla. AlertManager sustituirá dinámicamente los valores de las plantillas en función de la alerta emitida.

Instalación del servicio

Por último, se despliega el servicio de AlertManager mediante un Deployment. Y como ya se conoce, se expone el Deployment en el puerto 9093 mediante un Service de tipo LoadBalancer.

7.3.3 Prueba de carga con Prometheus

En este apartado se pondrá en explotación las herramientas instaladas de alerting. Para ello, como se ha explicado, se ha definido una regla en Prometheus que detectará si un nodo sobrepasa el uso de CPU un 80%.

Para mostrar el funcionamiento, se violará esta regla generando carga mediante un comando de peticiones infinitas a un servicio PHP desplegado en el clúster.

Generación de carga

Se genera la carga mediante el comando:

```
$ kubectl run -i --tty load-generator --rm --image=busybox --
restart=Never -- /bin/sh -c "while sleep 0.0001; do wget -q -
O- http://php-apache; done"
```

En la siguiente figura se muestra el resultado tras 5 minutos:

Figura 23. Ejecución de la prueba de carga.

Este comando, realiza consultas en bucle al servicio y genera la carga suficiente para que el nodo al que se le ha asignado el servicio sobrepase el límite de 80% de utilización de CPU.

Disparo de alerta en Prometheus

Una vez el nodo sobrepasa el límite, se dispara la regla en Prometheus. La regla pasa de “Inactive” (inactiva), a “Pending” (pendiente) y de ahí a “Firing” (Disparando).

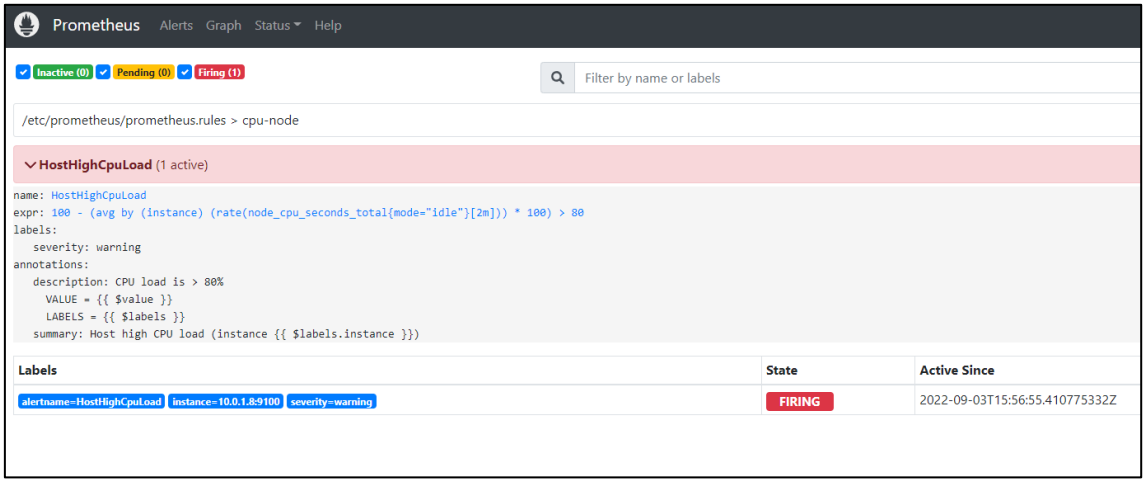

Figura 24. Disparo de alerta en dashboard de Prometheus.

Captación del disparo en AlertManager

Una vez disparada la alerta en Prometheus, AlertManager la captará y lanzará la orden de enviar el correo electrónico al destinatario configurado.

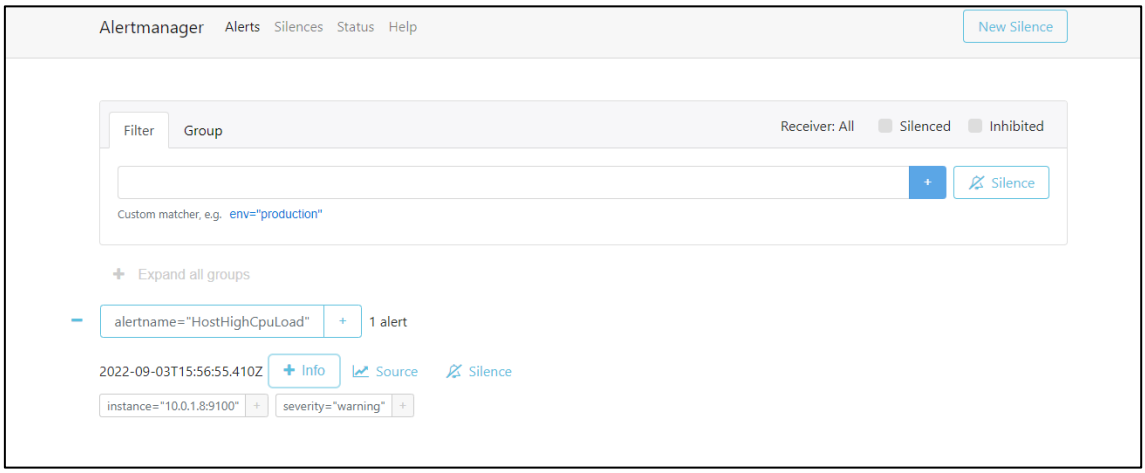

Figura 25. Captación de alerta disparada en AlertManager.

Correo

Por último, el destinatario recibirá un correo de este tipo:

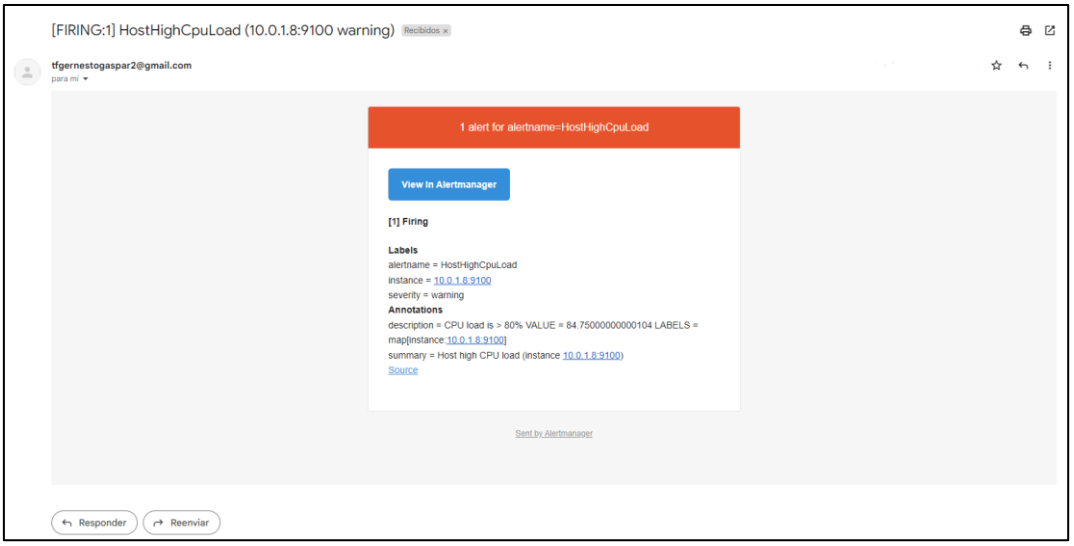

Figura 26 a. Vista del correo electrónico.

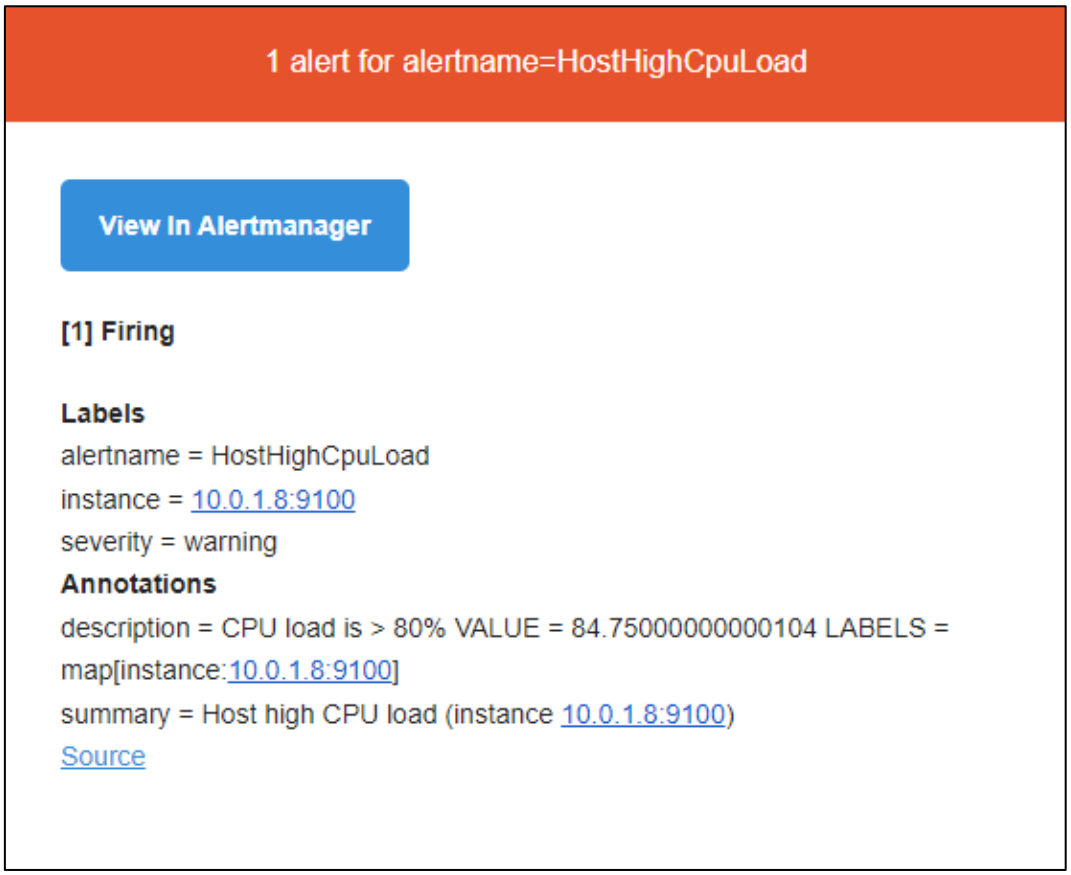

Figura 26 b. Detalle del correo electrónico.

Donde se especifica: qué regla se ha disparado, con qué valor, que instancia ha disparado la regla y una breve descripción sobre ella.

8. Conclusiones y trabajo futuro

8.1 Conclusiones

1. Las alternativas estudiadas en este trabajo para desplegar un clúster Kubernetes en GCP son: Kubeadm, Kops y Knative. En este trabajo se despliega un clúster altamente disponible en GCP con Kops.

2. Knative es el que presenta un nivel de implementación más fácil y sencillo. Kubeadm es el que consume menos recursos sin ningún tipo de servicio instalado y por lo tanto es el más económico.

3. Se ha implementado sobre un clúster una aplicación web escalable automáticamente.

4. Se ha implementado Prometheus como herramienta de monitorización y AlertManager como utilidad de alerta al administrador.

8.2 Trabajo futuro

Como trabajo futuro se puede proponer:

- Desplegar un clúster multi-master en Kubeadm.
- Estudiar el funcionamiento interno de la arquitectura de Knative.
- Desarrollar el autoescalado del clúster.
- Analizar las herramientas de “observability” que ofrece GCP.
- Implementar Grafana como herramienta de monitoreo secundaria a Prometheus.

9. Bibliografía

1. Swarm mode overview | Docker Documentation [Internet]. [citado 2022 ago 3]. Disponible en: https://docs.docker.com/engine/swarm/
2. Fin De Grado T, Carlos A, Fontan O. Creación de sistema Cloud con OpenStack.
3. Open Source Cloud Computing Infrastructure - OpenStack [Internet]. [citado 2022 ago 6]. Disponible en: https://www.openstack.org/
4. López PJ, María R, Cardona EB. Instalación, configuración y evaluación de un servidor de Internet de alta disponibilidad con equilibrado de carga basado en clúster. 2016;
5. Luksa M. Kubernetes in action. Kubernetes in action. Shelter Island: Manning Publications Co.; 2018.
6. Creating a cluster with kubeadm | Kubernetes [Internet]. [citado 2022 ago 6]. Disponible en: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/
7. kOps - Kubernetes Operations [Internet]. [citado 2022 ago 10]. Disponible en: https://kops.sigs.k8s.io/
8. What is Knative? | IBM [Internet]. [citado 2022 ago 11]. Disponible en: https://www.ibm.com/cloud/learn/knative
9. Home - Knative [Internet]. [citado 2022 ago 11]. Disponible en: https://knative.dev/docs/
10. Calculadora de precios de Google Cloud | Google Cloud [Internet]. [citado 2022 ago 12]. Disponible en: https://cloud.google.com/products/calculator
11. Application Design Patterns: Master/Slave - NI [Internet]. [citado 2022 ago 12]. Disponible en: https://knowledge.ni.com/KnowledgeArticleDetails?id=kA03q000000x1r9CAA&l=en-US
12. Understanding Kubernetes Architecture [Internet]. [citado 2022 ago 12]. Disponible en: https://geekflare.com/kubernetes-architecture/
13. Single Point of Contact - Wikipedia, la enciclopedia libre [Internet]. [citado 2022 ago 15]. Disponible en: https://es.wikipedia.org/wiki/Single_Point_of_Contact
14. Single point of failure - Wikipedia [Internet]. [citado 2022 ago 15]. Disponible en: https://en.wikipedia.org/wiki/Single_point_of_failure

15. Command line tool (kubectl) | Kubernetes [Internet]. [citado 2022 ago 15]. Disponible en: https://kubernetes.io/docs/reference/kubectl/

16. Conceptos | Kubernetes [Internet]. [citado 2022 ago 16]. Disponible en: https://kubernetes.io/es/docs/concepts/

17. Matagne M. Configuración y optimización de sistemas de cómputo. Actividad 3: Creación de un clúster Kubernetes (K8S). 2021.

18. Moreira Flors P. Configuración y optimización de sistemas de cómputo. Ampliación actividad 3: Creación de un clúster Kubernetes (K8S). 2021.

19. Martin P. Kubernetes Preparing for the CKA and CKAD Certifications. Gif-sur-Yvette: Apress; 2021.

20. Deploying to GCE - Alpha - kOps - Kubernetes Operations [Internet]. [citado 2022 ago 19]. Disponible en: https://kops.sigs.k8s.io/getting_started/gce/

21. About installing Knative - Knative [Internet]. [citado 2022 ago 19]. Disponible en: https://knative.dev/docs/install/

22. Getting started – Google Cloud console [Internet]. [citado 2022 ago 22]. Disponible en: https://console.cloud.google.com/getting-started

23. Install Serving with YAML - Knative [Internet]. [citado 2022 ago 22]. Disponible en: https://knative.dev/docs/install/yaml-install/serving/install-serving-with-yaml/

24. Todos los precios | Documentación de Compute Engine | Google Cloud [Internet]. [citado 2022 ago 22]. Disponible en: https://cloud.google.com/compute/all-pricing?hl=es#e2_machine-types

25. Converting a Kubernetes Deployment to a Knative Service - Knative [Internet]. [citado 2022 ago 24]. Disponible en: https://knative.dev/docs/serving/convert-deployment-to-knative-service/#kubernetes-nginx-deployment-and-service

26. Conceptos: Escalado de aplicaciones en Azure Kubernetes Service (AKS) - Azure Kubernetes Service | Microsoft Docs [Internet]. [citado 2022 ago 24]. Disponible en: https://docs.microsoft.com/es-es/azure/aks/concepts-scale

27. Farooq T. Configuración y optimización de sistemas de cómputo. Ampliación Kubernetes. 2020.

28. Vertical Pod Autoscaler - Amazon EKS [Internet]. [citado 2022 ago 26]. Disponible en: https://docs.aws.amazon.com/eks/latest/userguide/vertical-pod-autoscaler.html

29. Prometheus - Monitoring system & time series database [Internet]. [citado 2022 ago 26]. Disponible en: https://prometheus.io/

30. Alerting overview | Prometheus [Internet]. [citado 2022 ago 26]. Disponible en: https://prometheus.io/docs/alerting/latest/overview/

10. Anexo

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: metallb-system
  name: config
data:
  config: |
    address-pools:
    - name: my-ip-space
      protocol: layer2
      addresses:
      - 192.168.2.240-192.168.2.250
```

Anexo 1. Contenido del archivo de configuración de MetalLB.

```
apiVersion: apps/v1
kind: Deployment
metadata:
      name: nginx
      namespace: default
spec:
      selector:
          matchLabels:
                app: nginx
      replicas: 2
      template:
          metadata:
                labels:
                      app: nginx
          spec:
                containers:
                - name: nginx
                image: nginx
                ports:
                - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
      name: nginx-service
       namespace: default
spec:
      selector:
          app: nginx
      type: LoadBalancer
      ports:
          - name: http
          port: 8080
          protocol: TCP
          targetPort: 8080
```

Anexo 2. Contenido del archivo de un servicio NGINX de forma declarativa.

```
$ setsebool -P nfs_export_all_rw 1
$ setsebool -P nfs_export_all_ro 1
$ yum -y install policycoreutils-python
$ semanage fcontext -a -t public_content_rw_t "/var/web(/.*)?"
$ restorecon -Rv /var/web
$ firewall-cmd --permanent --add-service=nfs
$ firewall-cmd --permanent --add-service=mountd
$ firewall-cmd --permanent --add-service=rpc-bind
$ firewall-cmd –reload
```

Anexo 3. Configuración de flags de SELinux y cortafuegos.

```
apiVersion: v1
kind: PersistentVolume
metadata:
      name: nfs-www
spec:
      storageClassName: ""
      capacity:
            storage: 1Gi
      accessModes:
            - ReadOnlyMany
      persistentVolumeReclaimPolicy:
      mountOptions:
            - hard
            - nfsvers=4.1
      nfs:
            path: /var/web
            server: 192.168.2.200
            readOnly: false
```

```
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
     name: nfs-www-pvc
spec:
     storageClassName: ""
     volumeName: nfs-www
     accessModes:
          - ReadOnlyMany
     volumeMode: Filesystem
     resources:
          requests:
          storage:1G
```

Anexo 4. Contenido archivo de volúmenes persistentes del NFS.

```
$ gcloud compute firewall-rules create \
kubernetes-cluster-allow-internal \
--allow tcp,udp,icmp \
--network kubernetes-cluster \
--source-ranges 10.240.0.0/24,10.244.0.0/16
$ gcloud compute firewall-rules create \
kubernetes-cluster-allow-external \
 --allow tcp:22,tcp:6443,icmp \
--network kubernetes-cluster \
--source-ranges 0.0.0.0/0
```

Anexo 5. Configuración de cortafuegos para un clúster en GCP.

```
$ gcloud compute instances create controller \
--async \
--boot-disk-size 200GB \
--can-ip-forward \
--image-family ubuntu-1804-lts \
--image-project ubuntu-os-cloud \
--machine-type n1-standard-1 \
--private-network-ip 10.240.0.10 \
--scopes compute-rw,storage-ro,service-management, \
  service-control, logging-write, monitoring \
--subnet kubernetes \
--address $PUBLIC_IP
```

Anexo 6. Creación máquina virtual del nodo master.

```
$ for i in 0 1; do \
  gcloud compute instances create worker-${i} \
  --async \
  --boot-disk-size 200GB \
  --can-ip-forward \
  --image-family ubuntu-1804-lts \
  --image-project ubuntu-os-cloud \
  --machine-type n1-standard-1 \
  --private-network-ip 10.240.0.2${i} \
  --scopes compute-rw, storage-ro, service-management,
   servicecontrol, logging-write, monitoring \
  --subnet kubernetes; \
 done
```

Anexo 7. Creación máquina virtual de los nodos workers.

```
apiVersion: serving.knative.dev/v1
kind: Service
metadata:
  name: my-nginx
spec:
  template:
    spec:
       containers:
        -image: nginx
          ports:
          -containerPort: 80
```

Anexo 8. Contenido fichero Service en Knative.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-auto-deployment
spec:
  selector:
    matchLabels:
      app: my-auto-deployment
  replicas: 2
  template:
    metadata:
      labels:
        app: my-auto-deployment
    spec:
      containers:
      - name: my-container
```

```
        image: cruse/ubuntu-slim:0.1
        resources:
          requests:
            cpu: 100m
            memory: 50Mi
        command: ["/bin/sh"]
        args: ["-c", "while true; do timeout 0.5s yes
>/dev/null; sleep 0.5s; done"]
```

Anexo 9. Ejemplo de contenido de un Deployment para probar el VPA.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources:
  - nodes
  - nodes/proxy
  - services
  - endpoints
  - pods
  verbs: ["get", "list", "watch"]
- apiGroups:
  - extensions
  resources:
  - ingresses
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
```

```
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: default
  namespace: monitoring
```

Anexo 10. Contenido archivo ClusterRole de Prometheus.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoring
data:
  prometheus.rules: |-
    groups:
      - name: cpu-node
        rules:
          - alert: HostHighCpuLoad
            expr: 100 - (avg by(instance)
(rate(node_cpu_seconds_total{mode="idle"}[2m])) * 100) > 80
            for: 0m
            labels:
              severity: warning
            annotations:
              summary: Host high CPU load (instance {{
$labels.instance }})
              description: "CPU load is > 80%\n  VALUE = {{
$value }}\n  LABELS = {{ $labels }}"


  prometheus.yml: |
    global:
      scrape_interval:     15s
      evaluation_interval: 15s
      # scrape_timeout is set to the global default (10s).
    # Alertmanager configuration
    alerting:
      alertmanagers:
```

```
    - scheme: http
      static_configs:
      - targets:
         - "alertmanager.monitoring.svc:9093"
  rule_files:
    - /etc/prometheus/prometheus.rules
  scrape_configs:

    - job_name: 'prometheus'
      static_configs:
      - targets: ['localhost:9090']

    - job_name: 'node-exporter'
      kubernetes_sd_configs:
        - role: endpoints
      relabel_configs:
      - source_labels: [__meta_kubernetes_endpoints_name]
        regex: 'node-exporter'
        action: keep
```

Anexo 11. Contenido archivo ConfigMap de Prometheus.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoring
  labels:
    app: prometheus
spec:
  replicas: 1
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  selector:
    matchLabels:
      app: prometheus
  template:
    metadata:
      labels:
        app: prometheus
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9090"
    spec:
      containers:
      - name: prometheus
        image: prom/prometheus
        args:
```

```
        - '--storage.tsdb.retention=12h'
        - '--storage.tsdb.path=/prometheus/'
        - '--config.file=/etc/prometheus/prometheus.yml'
      ports:
        - containerPort: 9090
      resources:
        requests:
            cpu: 200m
            memory: 200M
        limits:
            cpu: 300m
            memory: 300M
      volumeMounts:
      - name: prometheus-config-volume
        mountPath: /etc/prometheus
      - name: prometheus-storage-volume
        mountPath: /prometheus

    volumes:
    - name: prometheus-config-volume
      configMap:
          defaultMode: 420
          name: prometheus-config

    - name: prometheus-storage-volume
      emptyDir: {}
```

Anexo 12. Contenido archivo Deployment de Prometheus.

Acrónimos

API: Application Programming Interface (Interfaz de Programación de Aplicaciones).

AWS: Amazon Web Services

GCP: Google Cloud Platform

GKE: Google Kubernetes Engine

HPA: Horizontal Pod Autoescaling (Autoescalado Horizontal de Pods).

IaaS: Infraestructure as a Service (Infraestructura como Servicio).

NFS: Network File System (Sistema de Archivos de Red).

PaaS: Platform as a Service (Plataforma como Servicio).

SaaS: Software as a Service (Software como Servicio).

SPOC: Single Point of Contact (Punto Único de Contacto).

SPOF: Single Point of Failure (Punto Único de Fallo).

VPA: Vertical Pod Autoescaling (Autoescalado Vertical de Pods).

Glosario de términos

Apt: es una herramienta de gestión de paquetes para sistema Linux.

Autoescalado: capacidad de escalar un servicio de manera automática.

Balanceador de carga: se refiere a la técnica usada para compartir el trabajo a realizar entre varios ordenadores, procesos, discos u otros recursos.

Clúster: Un sistema de procesamiento paralelo que consta de un conjunto de computadoras independientes interconectadas entre sí y que actúan como un solo recurso computacional.

Contenedor: Es una virtualización del sistema operativo de un servidor que es capaz de ejecutar desde un microservicio a una aplicación de mayor tamaño.

Deployment: es un objeto de Kubernetes que puede representar una aplicación en un clúster.

Docker: Es un sistema operativo para contenedores.

Endpoint: es un dispositivo informático remoto que se comunica a través de una red a la que está conectado.

Frontend: es la parte de un programa o dispositivo a la que un usuario puede acceder directamente.

Hipervisor: es un proceso que crea y ejecuta máquinas virtuales.

Infraestructura: conjunto de software y hardware sobre el que se soportan los servicios de una organización.

Kubectl: es la interfaz de línea de comandos de Kubernetes.

Kubernetes o K8S: es un conocido orquestador de contendores.

Namespace: Espacio de nombres. Se usa para declarar un ámbito que contiene un conjunto de objetos relacionados.

Nodos: En redes de computadoras cada una de las máquinas es un nodo.

Orquestador de contenedores: es un sistema que organiza y automatiza el despliegue de contenedores.

Open source: que es de código abierto.

Plano de Control: se refiere al nodo maestro. Sistema encargado de la interacción de los usuarios con el clúster de Kubernetes.

Pod: es un grupo de uno o más contenedores. Representa el bloque de construcción básico de Kubernetes.

Quorum: sistema de votación de nodos maestros para decidir la continuidad de un clúster.

Revisión: es un concepto de Knative que se crea cada vez que se actualiza un servicio.

Serverless: sin servidor. Es una solución que permite crear y ejecutar aplicaciones con rapidez y menor costo total de propiedad, ya que no es necesario aprovisionar y administrar infraestructura.

Service: es un objeto de Kubernetes que permite exponer un Deployment.

Shell: se refiere al intérprete de comandos de los sistemas operativos basados en Unix.

Traffic Splitting: es un concepto de Knative que permite la división del tráfico entre revisiones.

YAML: es un formato de serialización.

YUM: es una herramienta de gestión de paquetes para sistema Linux.

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

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Hace unos años, las empresa y particulares solían hacer uso de sistemas informáticos instalados en sus propias oficinas. Esto conllevaba una gran inversión económica, tanto en la adquisición como en el mantenimiento de los equipos, en el gasto eléctrico, en el personal especializado, etc. Por otro lado, el desarrollo de software siempre se ha realizado con una estructura sólida y muy acoplada, que impedía la escalabilidad de manera sencilla de un programa. En este Trabajo de Fin de Grado, se expone al público una forma innovadora de relacionar las infraestructuras cloud con las arquitecturas de microservicios. Este sistema conjunto de modelos constituye un nuevo avance tecnológico para los usuarios, que quieren desplegar servicios más ligeros y flexibles sin necesidad de ser propietarios de grandes y costosas infraestructuras físicas. Los sistemas de cómputo en la nube, además, brindan un espacio seguro, de fácil accesibilidad y con una personalización de servicios a gusto del consumidor. Por ello, se ha relacionado este TFG principalmente, con un nivel Alto, con el ODS nº 9. Industria, innovación e infraestructuras, ya que supone una innovación en las infraestructuras de una sociedad facilitando el comercio internacional y permitiendo el uso eficiente de los recursos.

Por otro lado, el estudio recopila información y datos verídicos sobre cómo montar un sistema de clúster altamente disponible en la nube mediante el uso de autoescaladores. Como ya se conoce, los sistemas cloud obtienen beneficio mediante el pago por uso de sus servicios e infraestructuras en la nube. Gracias al manejo de autoescaladores, existe la posibilidad de sólo pagar por lo que se usa, con el consiguiente ahorro y crecimiento económico que causa en las empresas que lo implementan. Es por esta razón, que se ha relacionado el trabajo en segundo lugar, con un nivel Alto el ODS nº 8. Trabajo decente y desarrollo económico.

Al mismo tiempo, se relaciona con un nivel Alto el ODS nº 4. Educación de calidad, ya que este TFG recopila información de fuentes de datos fiables, como las documentaciones oficiales de las tecnologías de las que se hace uso. También analiza otros trabajos realizados con anterioridad, como Trabajos de Fin de Grado de otros alumnos al igual que, actividades complementarias de la asignatura del Master en Ingeniería Informática de la UPV: Configuración y optimización de sistemas de cómputo. También se ha puesto en práctica la consulta de libros de renombre utilizados al nivel de un TFG. Para fomentar el desarrollo de un país, es de vital importancia una educación de calidad, clave para la movilidad socioeconómica ascendente y abandonar así la pobreza.

Por último, como se ha visto a lo largo del proyecto, se intentan asignar recursos a un servicio acorde al consumo real que se va a realizar de este. Se evita así un consumo menor de los recursos por dos vías: una vía de ahorro de infraestructuras físicas que pueden ser muy costosas y una segunda vía gracias al uso del autoescalador vertical, que permite analizar si un servicio solicita más recursos de los que utilizará y limitárselos a su consumo necesario. Es por esto por lo que este Trabajo de Fin de Grado va relacionado con un nivel Alto, con el ODS nº 12 Producción y consumo responsable. Ya que el consumo y la producción sostenible consiste en hacer más y mejor con menos.