ADMINISTRACIÓN E IMPLEMENTACIÓN DE CONTENEDORES EN UN PROVEEDOR DE CLOUD PÚBLICA

AUTOR

ANDRES FELIPE PARRADO CARMONA

Trabajo de grado presentado como requisito para optar al título de:

INGENIERO EN TELECOMUNICACIONES

Director:

LUIS FERNANDO GONZÁLEZ DELACALLE

UNIVERSIDAD MILITAR NUEVA GRANADA

FACULTAD DE INGENIERÍA

PROGRAMA DE INGENIERÍA EN TELECOMUNICACIONES

BOGOTÁ, 27 DE AGOSTO 2021

UNIVERSIDAD MILITAR
NUEVA GRANADA

Agradecimientos

Son pocas las ocasiones en la cuales tenemos la oportunidad de plasmar sentimientos de agradecimientos una vez que culminamos logros que equivalen a la inversión de una gran cantidad de esfuerzo y dedicación.

Agradezco y dedico este trabajo a mi familia, en especial a mi padre y mi madre los cual me otorgaron su pericia y tenacidad que me han definido hasta este momento, dándome el privilegio de estudiar además de ser el aliciente para finalizar mis estudios y este trabajo.

De igual manera mi más sincero agradecimiento al Ing. Luis González director del trabajo de grado y docente, quien ha sabido concederme su conocimiento en el área profesional, durante todo el desarrollo de mi pregrado y de este trabajo.

Finalmente agradezco a todos mis docentes y compañeros, que fueron parte de mi desarrollo profesional de los cuales logre aprender.

Resumen

En los últimos años la computación en la nube ha sido una de las principales tecnologías para el despliegue de infraestructura tecnología por parte de las empresas, las cuales ha solicitado servicios de cómputo remoto para prestar sus servicios por medio de proveedores de nube pública. Para lograr que los servicios se han más eficientes cuando son desplegados en la nube, los proveedores de nube pública optaron por integrar la nube nativa que ha permitido desplegar tecnologías de innovación, como los contenedores que optimizan y reducen el costo de la infraestructura desplegada en la nube pública.

Los proveedores de nube pública tiene los mismos modelos de servicios, pero con nombres diferentes, permitiendo desplegar el mismo tipo de aplicativo en cualquier proveedor. La gran mayoría de los proveedores otorga un crédito en dólares estadounidenses para utilizar los servicios de su nube totalmente gratis. Cada uno mantiene políticas de uso del crédito que indican un tiempo límite de uso o que servicios no se cubre por el crédito. La característica del crédito aplicable al modelo de servicios para desplegar contenedores logrará ser el factor de selección del proveedor indicador.

La red donde están desplegados los recursos de la nube del proveedor tiene una nomenclatura especifica para poder reconocer donde están ubicados los recursos, permitiendo seleccionar la región y zona donde serán reservados en el momento de solicitar los recursos. Comprender la red del proveedor permitirá desplegar óptimamente los recursos teniendo en cuenta en donde se despliega y desde donde se solicita el servicio.

Kubernetes tiene la gestión de infraestructura de IaaS e infraestructura dinámica de PaaS. Permitiendo poder gestionar el tipo de recurso y como está distribuido en la infraestructura. Los diferentes tipos de objetos permiten dar dinamismo a la parte lógica del aplicativo que se despliegue en Kubernetes.

La infraestructura gestionada de IaaS proviene de la configuración absoluta del recurso por parte del cliente. Se pueden seleccionar las características de la máquina virtual, como la CPU, el sistema operativo y la ubicación. El proveedor de nube se preocupará por mantener la disponibilidad de la máquina virtual.

La infraestructura dinámica de PaaS proviene de la gestión de un repositorio de lenguajes de programación de la aplicación del cliente. El cliente solo se preocupa por desplegar la parte lógica de la aplicación dejando de lado la infraestructura. El proveedor de nube otorgará los recursos y la infraestructura configurados para que el aplicativo funcione.

Kubernetes basa su infraestructura en él despliegue de objetos que administran a los contenedores, permitiendo ubicar el objeto en un recurso dedicado dentro del clúster. Kubernetes tiene una sintaxis de configuración igual para recursos, objetos o servicios, permitiendo gestionar toda la infraestructura de la misma forma sin importar el aplicativo o los requerimientos.

Kubernetes es un sistema que permite una administración completa de la infraestructura, debido a la declaración de estados por parte del administrador y de Kubernetes. Los estados declarados permitirán que el objeto Pod, pueda ejecutar y desplegar con éxito los contenedores. El plano de control otorgará soporte a la arquitectura del clúster realizando procesos internos para ejecutar y mantener activos los Pods, procesos que son liderados por un recurso denominado el nodo Maestro.

Kubernetes permite desplegar objetos dentro del clúster por medio de archivos programados con lenguaje Yaml. Un tipo de lenguaje que tiene la característica tener un sentido gramatical, posibilitando la lectura por parte de administrador y también del sistema Kubernetes. Los espacios de nombre son una etiqueta declarada dentro del archivo Yaml de un objeto para posicionarlo en una sección específica del clúster, generando diferentes espacios de nombre con la posibilidad de seccionar el aplicativo dentro del clúster.

Cuando los objetos y los recursos están desplegados en el clúster de Kubernetes, el mismo sistema presenta el comando Kubectl para monitoreo de toda la infraestructura. Kubectl tiene una sintaxis de 4 secciones que permiten comprender fácilmente la acción a realizar sobre los recursos u objetos. El monitoreo de Kubectl se realiza por simples consultas que presentan detalles sobre lo que está pasando con el recurso u objeto declarado en el comando Kubectl.

El despliegue es el objeto de Kubernetes que permite implementar varios Pods iguales al mismo tiempo. La gestión conjunta de todos los Pods de un Despliegue permite aplicar escalado o autoescalado a los Pods que se administran, además de permitir diferentes tipos de actualización del Despliegue según los requerimientos del aplicativo.

Los trabajos y crono-trabajos son objetos de Kubernetes que se caracterizan por realizar tareas temporales o periódicas. Existen subcategorías de estos objetos que permiten realizar varias tareas para reducir la utilización de recursos o dedicar varios Pods a la misma tarea para generar redundancia en el aplicativo.

La gestión del clúster permite la asignación automática o no, de recursos computacionales a los Pods según los requerimientos de la implementación. La sincronización de la gestión del clúster con la de los objetos, permitirá que los objetos puedan ser desplegados con eficiencia. Además de permitir que los objetos se puedan conectar a otros objetos o a diferentes tipos de redes.

Kubernetes Engine es el servicio de la plataforma de la nube de Google sobre la cual se realizó la exploración del servicio por medio del despliegue de diferentes implementaciones, cada una de las implementaciones son la aplicación de la teoría de los capítulos anteriores.

La primera implementación permite desplegar una página web básica por medio de un contenedor que es gestionado por la infraestructura de Kubernetes. Mostrando los comandos necesarios para

modificar el clúster, como se habilitan los puertos de exposición del servicio web e interacción con los objetos de Kubernetes.

La segunda implementación permite desplegar una página web básica en la red interna que tiene el clúster una vez que es implementado. Mostrando los comandos de aplicación del archivo Yaml de un objeto Kubernetes, la interacción con la consola del objeto y la exposición local de la página web.

La tercera implementación permite desplegar un servicio web compuesto por varios objetos de Kubernetes, que son gestionados y desplegados al mismo tiempo. Mostrando los comandos para aumentar el número de objetos y la aplicación de un objeto dedicado a la exposición del servicio a internet.

La cuarta implementación permite desplegar y gestionar contenedores que realizan tareas temporales y periódicas. El primero gestiona un contenedor que realiza un cálculo matemático finito y el segundo imprime periódicamente una cadena de caracteres. Mostrando los comandos para observar las tareas realizadas por los objetos de Kubernetes.

La quinta implementación permite desplegar dos aplicaciones basadas en contenedores en un mismo clúster. La primera aplicación tiene contenedores de servicio web y la segunda tiene contenedores que generan peticiones a un URL específico. Mostrando los comandos para conectar los aplicativos y simular la actividad de una aplicación cuando se expone a internet.

Uno de los principales procesos que se ejecutan cuando se despliega una implementación exitosa en Kubernetes, es el poder realizar un monitoreo efectivo que permita comprender que está sucediendo con el sistema y los objetos. Para ello se creó un Dashboard seccionado en métricas para los Nodos y métricas para los Pods. También se crearon políticas de alarmas que envían el reporte a un canal de notificación diferente a GCP.

UNIVERSIDAD MILITAR NUEVA GRANADA

Tabla de Contenidos

UNIVERSIDAD MILITAR
NUEVA GRANADA

Lista de Figuras

Universidad Militar Nueva Granada

UNIVERSIDAD MILITAR NUEVA GRANADA

1. Introducción

1.1 Antecedentes

En los últimos años ha aumentado el número de empresas que utilizan la nube para desplegar e implementar su infraestructura tecnológica gracias al dinamismo que tiene la nube. Permitiendo dar mayor confianza a nuevas empresas o entidades que desean utilizar la nube. El dinamismo de la nube proviene del diseño y presentación de modelos de servicios por parte de los proveedores de nube que permiten desplegar diversos servicios específicos y dedicados en sus nubes [1]. Permitiendo desplegar una infraestructura o servicio sin tener que preocuparse por el tiempo despliegue o costos de mantenimiento de los equipos físicos.

[1] Gibson, J., Rondeau, R., Eveleigh, D., & Tan, Q. (2012, November). Benefits and challenges of three cloud computing service models. IEEE.

Debido a la popularidad y el aumento de utilización de la nube, algunas entidades gubernamentales de nivel nacional en Colombia como es el Ministerio de Tecnologías de la Información y las Comunicaciones (Mintic), expido una guía que permite identificar conceptos y entidades relacionadas en la computación en la nube:

Nube pública:

Es un modelo que permite el acceso omnipresente, conveniente, y por demanda a una red de un conjunto compartido de recursos computacionales configurables, que está aprovisionada para uso abierto por el público en general [2]. Los modelos de servicios son las formas como se presentan aplicaciones, servicios, recursos o infraestructura a un cliente de un proveedor de nube. Son varios los modelos, pero los más importantes son los siguientes:

Software como servicio (SaaS):

Modelo de servicios que da la capacidad al consumidor para proveerse de un aplicativo dedicado, listo y diseñado previamente por el proveedor, para ser inmediatamente utilizado [2].

Plataforma como servicio (PaaS):

Modelo de servicios que da la capacidad al consumidor para desplegar en la infraestructura de nube aplicaciones creadas o adquiridas por el consumidor [2].

Infraestructura como servicio (IaaS):

Modelo de servicios que da la capacidad al consumidor para proveer procesamiento, almacenamiento, redes y otros recursos de computación fundamentales sobre la infraestructura de nube [2].

[2] G.ST.02 Guía de Computación en la nube - Ministerio de Tecnologías de la Información y las Comunicaciones, Mintic.gov.co, 2018.

Nube tradicional

Existe una gran cantidad de empresas y aplicaciones que utilizan la computación en la nube dentro de sus infraestructuras. La denominación de nube tradicional proviene de utilizar los tres principales modelos individual o conjuntamente por el cliente, Donde dichos modelos son el despliegue tecnológico dentro de la nube de servicios tradicionales, que antes se desplegaban en recursos o infraestructuras propias que tenían las empresas o clientes [3].

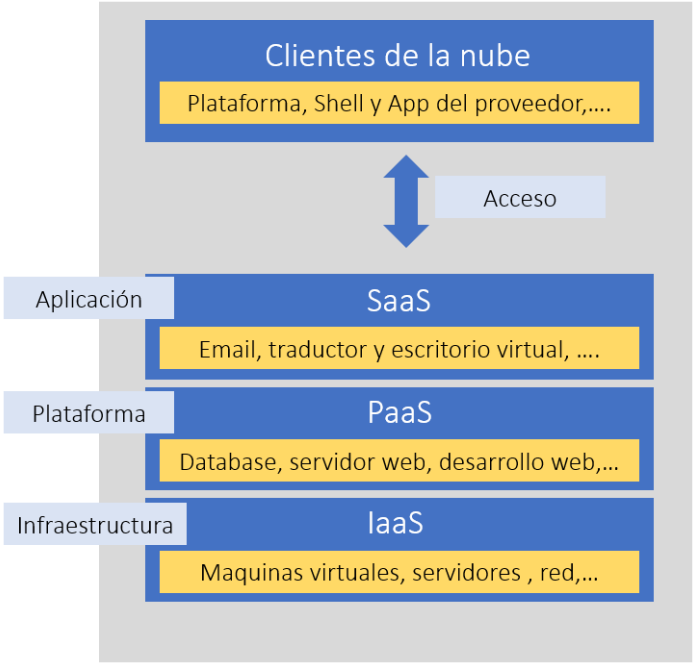

Figura 1.1: Estructura de modelos y servicios de la nube tradicional [3].

[3] Mohammadi, S., & Mohammadi, A. (2014). Effect of cloud computing in accounting and comparison with the traditional model.

Debido al avance tecnológico realizado por proveedores y clientes al desplegar o generar nuevos servicios en la nube. Se crea una nueva generación de la nube, mucho más óptima y eficientes con nuevos modelos de servicios y la permisividad de acoplar nuevas tecnologías en su infraestructura, denominada la nube nativa.

1.2 Objetivos

1.2.1 Objetivo general

- Identificar y caracterizar la administración e implementación de contenedores en un proveedor cloud pública, por medio de la compilación de información teórica y exploración de la plataforma del proveedor.

1.2.2 Objetivos específicos

Fueron cinco los objetivos específicos planteados para lograr identificar y caracterizar la administración e implementación de contenedores en un proveedor de nube pública:

1. Seleccionar un proveedor de cloud pública para la exploración de la administración e implementación de contenedores en cloud.
2. Comparar las arquitecturas existentes en el proveedor de cloud seleccionado.
3. Caracterizar la arquitectura implementada para la administración de contenedores del proveedor de cloud pública seleccionado.
4. Identificar y definir la teórica y estructura fundamental de la administración e implementación de contenedores en el proveedor de cloud.
5. Realizar implementaciones de contenedores en cloud, que permitan aplicar la caracterización previa de la teoría de administración e implementación de contenedores en cloud.
6. Realizar el monitoreo sobre una implementación, que permita generar un escenario de estrés para visualizar el comportamiento de las métricas del sistema y los objetos.

1.3 Metodología

En el capítulo 2 **Proveedores de nube pública**: se selecciona el proveedor de nube pública con mayor factibilidad y beneficios otorgados por el proveedor al momento de implementar contenedores en la nube. En el capítulo 3 **Modelos de servicios del proveedor** se comparan las arquitecturas existentes en el proveedor de nube seleccionado en el capítulo anterior. En el capítulo 4 **Caracterización de la arquitectura**: se caracteriza la arquitectura ofrecida por el proveedor de nube para la administración de contenedores. En el capítulo 5 **Gestión de objetos y clúster en Kubernetes**: se identifica, analiza y documenta las teorías relacionadas con la administración e implementación de contenedores en el proveedor de nube. En el capítulo 6 **Implementaciones en Kubernetes Engine**: se documentan diferentes implementaciones con el paso a paso para lograr administrar e implementar contenedores en el proveedor seleccionado. En el capítulo 7 **Monitoreo de una implementación**: se despliega una implementación más robusta que las anteriores, para lograr generar un escenario de estrés que monitoreara el sistema y los objetos, por medio de un servicio de GCP de Dashboard y políticas de alarmas.

Para lograr desarrollar cada uno de los objetivos se aplican metodologías de tipo exploratoria, aplicada y documental. Iniciando con una recopilación de información básica sobre diferentes proveedores, posteriormente se realiza una caracterización de la información que otorgar el proveedor sobre sus servicios en la nube como también la información del desarrollador de Kubernetes, realizando diagramas de flujo, cuadros comparativos además de la utilización de figuras para describir elementos del sistema de Kubernetes, paralelamente se explora la plataforma de la nube del proveedor y se despliegan aplicaciones para comprobar el funcionamiento de Kubernetes en la nube, finalmente se documentan instructivos de las implementaciones básicas que se realizan en la plataforma del proveedor.

1.4 Introducción en la nube nativa

Actualmente un cliente en la nube puede generar cientos de aplicativos y servicios en un instante, como también dichos servicios pueden tener miles o millones de solicitudes de acceso. Los aplicativos actuales desplegados en la nube deben soportan más clientes de los que se soportaban en anteriores años, además de gestionar más servicios. La nube nativa permite suplir las deficiencias que pueden tener las aplicaciones, servicios o la infraestructura a los requerimientos de la actualidad.

Nube nativa

La nube nativa consta de la utilización de 4 tecnologías: desarrollo y operaciones (DevOps), entrega continua, microservicios y contenedores. El utilizar cualquiera de esas tecnologías dentro de una infraestructura en la nube permite que sea más flexible, más rápida, de mejor calidad y por un costo menor en relación con la utilización de la nube tradicional, soportando muchos más clientes y permitiendo gestionar más servicios con menos recursos [4]. Pero para poder utilizar la nube nativa se requiere conocimientos en la tecnología y en los nuevos sistemas que la soportan.

[4] Gannon, D., Barga, R., & Sundaresan, N. (2017). Cloud-native applications. IEEE Cloud Computing.

Contenedor

Es un bloque de código lógico, que incluye el código de una aplicación junto con los archivos de configuración, las bibliotecas y dependencias necesarias para que la aplicación se ejecute [5]. Gracias a los contenedores se puede reducir la cantidad de recursos o requerimientos dentro de una infraestructura en la nube, volviéndola más eficiente y económica.

[5] B. Burns, "What is a container?", azure.microsoft.com, 2019.

Kubernetes

Es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios basados en contenedores. Kubernetes facilita la automatización y la configuración declarativa [6]. Una vez desplegado el sistema Kubernetes en la nube se puede orquestar eficientemente una infraestructura basada en múltiples contenedores e individualizar los servicios de una aplicación para manejarlos por medio de diferentes contenedores.

[6] Brewer, E. A. (2015, August). Kubernetes and the path to cloud native.

Actualmente algunas empresas desconocen como podría llegar a utilizarse el sistema de Kubernetes para implementar contenedores y optimizar sus infraestructuras en la nube, implicado que existan muy pocas implementaciones en la industria, baja producción científica y falta de caracterización del modelo de servicios de Kubernetes en la nube por parte de MinTic, en comparación con otras soluciones que podemos encontrar en la nube pública, así que este es el principal motivo para desarrollar este trabajo que permitirá entender la administración e implementación de contenedores en la nube pública.

2. Proveedores de nube pública

Los proveedores de nube pública tiene los mismos modelos de servicios, pero con nombres diferentes, permitiendo desplegar el mismo tipo de aplicativo en cualquier proveedor. La gran mayoría de los proveedores otorga un crédito en dólares estadounidenses para utilizar los servicios de su nube totalmente gratis. Cada uno mantiene políticas de uso del crédito que indican un tiempo límite de uso o que servicios no se cubre por el crédito. La característica del crédito aplicable al modelo de servicios para desplegar contenedores logrará ser el factor de selección del proveedor indicador.

La red donde están desplegados los recursos de la nube del proveedor tiene una nomenclatura especifica para poder reconocer donde están ubicados los recursos, permitiendo seleccionar la región y zona donde serán reservados en el momento de solicitar los recursos. Comprender la red del proveedor permitirá desplegar óptimamente los recursos teniendo en cuenta en donde se despliega y desde donde se solicita el servicio.

2.1 Selección de proveedor de nube pública

Para la selección del proveedor de nube pública se calificarán las siguientes características:

- Es fundamental que el proveedor tenga incorporado el sistema de Kubernetes (Condición obligatoria).
- Crédito gratuito para utilizar en las implementaciones en el sistema de Kubernetes.
- Servicio de repositorio para contenedores.
- Capacitación gratuita sobre el sistema de Kubernetes.

Proveedores de nube preseleccionados

Debido a que la primera condición es obligatoria para lograr ser seleccionado, se tuvieron en cuenta los siguientes 3 proveedores de nube pública:

- **Google Cloud**
- **Microsoft Azure**
- **Amazon Web Services**

Estos tres anteriores proveedores mantienen incorporados el sistema de Kubernetes dentro de sus plataformas de nube, como se puede observar en la figura 2.1. Se puede dirigir a sus correspondientes plataforma dando clic en el nombre de cada uno de los proveedores.

Figura 2.1: Proveedores y Sistemas Kubernetes. Logos extraídos de: Google Cloud, Amazon Web Services, Microsoft Azure.

Nube de Google

El modelo de servicios de Google es **Motor de Kubernetes de Google**. Referente al crédito gratuito, Google nos da **300 USD** utilizables en el sistema de Kubernetes, de igual manera existe un servicio de repositorio de contenedores y como un plus Google ofrece un curso gratuito sobre el manejo básico de cada uno de sus servicios más importantes.

Microsoft Azure

El modelo de servicios de Microsoft Azure es **Servicio de Kubernetes elásticos**. Referente al crédito gratuito, Microsoft Azure nos da **200 USD** utilizables en el sistema de Kubernetes, de igual manera existe un servicio de repositorio de contenedores.

Servicios web de Amazon

El modelo de servicios de Amazon es **Servicio de Kubernetes**. Referente al crédito gratuito, Microsoft Azure mantiene una capa gratuita de algunos servicios pero no para su servicio de Kubernetes. De igual manera que los dos anteriores proveedores existe un servicio de repositorio de contenedores.

Comparativa de proveedores de nube pública

Como se observa en la figura 2.2 se dieron valores porcentuales diferentes a cada una de las características comparadas; estos valores determinan el grado de importancia o influencia, para la selección del proveedor de nube pública para realizar la exploración documental y aplicativa de la administración e implementación de los contenedores.

2.1.1 Resultados de selección del proveedor de nube pública

Una vez que se le dan valores numéricos a cada una de las características de los proveedores se obtiene los resultados de la figura 2.3.

Características del proveedor de nube	Valor porcentual de influencia de la característica	Nube de Google	Microsoft Azure	Servicios Web de Amazon
Modelo de servicios	40%	Motor de Kubernetes	Servicio de Kubernetes Elásticos (EKS)	Servicio de Kubernetes (AKS)
Crédito gratuito en servicios	30%	300 USD	200 USD	No disponible para Kubernetes
Reposotorio de Contenedores	20%	Registro de Contenedores	Registro de Contenedores Elásticos	Registro de Contenedores
Capacitación gratuita en Kubernetes	10%	Fundamentos de la plataforma en la nube de Google: Infraestructura básica	No tiene	No tiene
	100%			

Figura 2.2: Evaluación de características para la selección de proveedor de nube pública.

Características del proveedor de nube	Valor máximo de calificación	Nube de Google	Microsoft Azure	Servicios Web de Amazon
Modelo de servicios	40%	50	50	50
Crédito gratuito en servicios	30%	50	40	0
Reposotorio de Contenedores	20%	50	50	50
Capacitación gratuita en Kubernetes	10%	50	0	0
Puntaje Total	100%	50	42	30

Figura 2.3: Calificación de características para la selección de proveedor de nube pública.

Proveedor de nube pública seleccionado

Debido a los valores obtenidos en la figura 2.3, se seleccionó la nube de Google para hacer la debida exploración documental y aplicada de la administración e implementación de contenedores en la nube pública. Se puede afirmar que los factores diferenciadores con los otros proveedores, es un mayor crédito para gastos en las implementaciones y la capacitación gratuita sobres sus servicios.

2.2 Nube pública de Google

La nube de Google es la más apropiada respecto a las características de selección previamente presentadas y de ahora en adelante se examinará la documentación que nos otorga Google, sobre sus servicios relacionados con el sistema de Kubernetes.

2.2.1 Características de la nube de Google

Puede existir un concepto diferente de nube para cada proveedor debido a las características que este tenga en sus servicios o infraestructura. Google basa su concepto de la nube en 5 características principales [7]:

Bajo demanda y autoservicio

Como clientes de la nube; interactuamos con una interfaz automatizada para lograr obtener capacidad de procesamiento, almacenamiento y red que necesitamos, sin intervención humana a

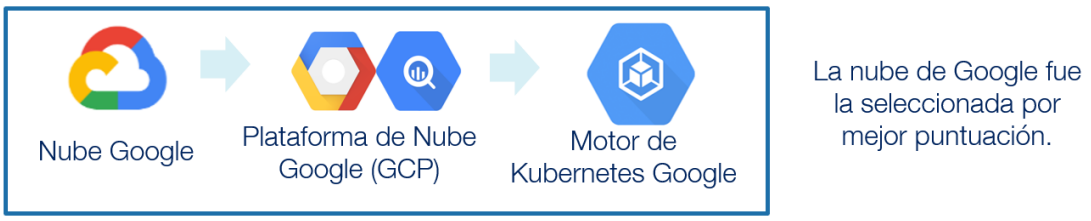

Figura 2.4: Proveedor seleccionado; plataforma de la nube; sistema Kubernetes. Logos extraídos de: Google Cloud.

la hora de la petición de los servicios.

Amplio acceso a la red

Podemos acceder a los recursos a través de una red desde cualquier lugar y en cualquier momento.

Recursos compartidos

Los recursos son asignados por el proveedor a los clientes desde de un inmenso fondo común, aplicando los beneficios de la economía de escala. De igual esto evita que los clientes tengan que preocuparse por la ubicación o procedencia de los recursos.

Elasticidad rápida

Los recursos son elásticos. Si solicito más recursos estos se me asignan rápidamente y si requiero menos estos se reducen.

Servicio medido

Solo se paga por los recursos que se usan o reservan. Si dejan de utilizarse, se deja de pagar por dichos recursos.

La nube de Google basa sus características en: [7] Peter M. Mell, Timothy Grance "The NIST Definition of Cloud Computing", Special Publication (NIST SP), 2011.

2.2.2 Red, regiones y zonas de la nube de Google

Google administra todos sus recursos que están detrás de sus servicios dentro de sus centros de datos globales. Estos centros de datos están ubicados en 24 regiones, 73 zonas en aproximadamente 144 puntos de presencia en 200 países, en la figura 2.5 se puede observar la distribución actual de la red de centro de datos de Google.

Es esencial entender como se constituye las regiones y zonas como se observa en la figura 2.6, esto debido a la distribución de recursos que se pueda dar según los modelos de servicios que nos ofrezca Google con relación a Kubernetes.

Multi-regiones

Google divide el mundo en tres áreas multi-regionales:

- América
- Europa
- Asia-Pacífico

Y a su vez, las multi-regiones se dividen en regiones.

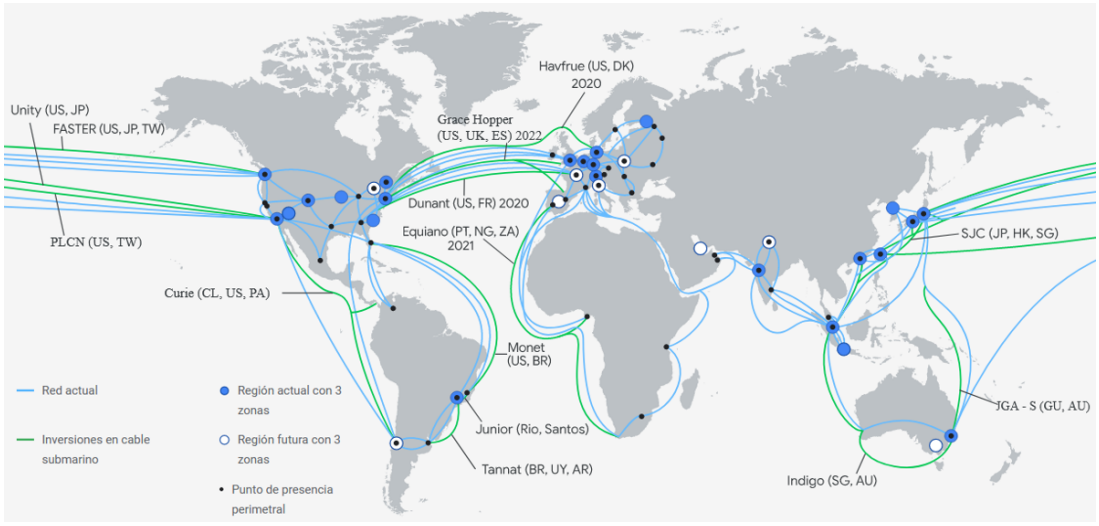

Figura 2.5: Mapa de la infraestructura de la nube de Google : imagen extraída del URL

Región

Una región puede cubrir una gran cantidad de países de un continente, donde existe una rápida conectividad de red: con latencias de red de menos de 1 milisegundo. Y a su vez las regiones se dividen en zonas.

Zona

Una zona es un área geográfica específica para el despliegue de recursos de la nube de Google. Se podría pensar que una zona es como un centro de datos dentro de una región, pero una zona no es únicamente un solo centro de datos [8].

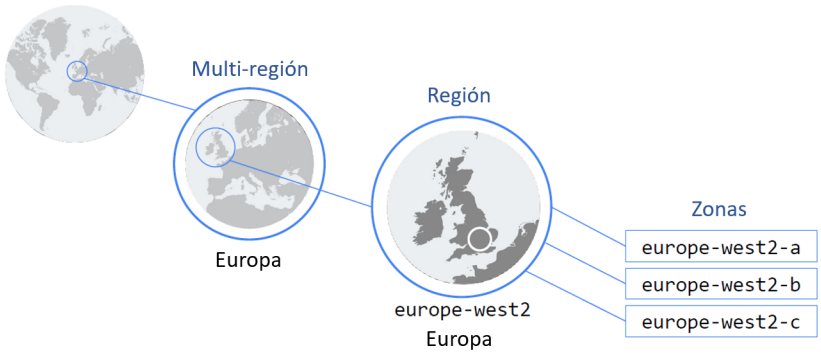

Figura 2.6: Constitución de las regiones y zonas de la nube de Google.

[8] Google Cloud "Geografía y regiones", Información general de Google Cloud, 2020.

En el siguiente capítulo se examinará los modelos de servicios de la plataforma de Google Cloud (Plataforma de la nube de Google), comparando los modelos estándar de IaaS y PaaS para lograr comprender el modelo de servicios que presenta la plataforma de Google Cloud con relación a Kubernetes.

SCIENTIÆ · PATRIÆ · FAMILIÆ
UNIVERSIDAD MILITAR NUEVA GRANADA

3. Modelos de servicios del proveedor

Kubernetes tiene la gestión de infraestructura de IaaS e infraestructura dinámica de PaaS. Permitiendo poder gestionar el tipo de recurso y como está distribuido en la infraestructura. Los diferentes tipos de objetos permiten dar dinamismo a la parte lógica del aplicativo que se despliegue en Kubernetes.

La infraestructura gestionada de IaaS proviene de la configuración absoluta del recurso por parte del cliente. Se pueden seleccionar las características de la máquina virtual, como la CPU, el sistema operativo y la ubicación. El proveedor de nube se preocupará por mantener la disponibilidad de la máquina virtual.

La infraestructura dinámica de PaaS proviene de la gestión de un repositorio de lenguajes de programación de la aplicación del cliente. El cliente solo se preocupa por desplegar la parte lógica de la aplicación dejando de lado la infraestructura. El proveedor de nube otorgará los recursos y la infraestructura configurados para que el aplicativo funcione.

Kubernetes basa su infraestructura en él despliegue de objetos que administran a los contenedores, permitiendo ubicar el objeto en un recurso dedicado dentro del clúster. Kubernetes tiene una sintaxis de configuración igual para recursos, objetos o servicios, permitiendo gestionar toda la infraestructura de la misma forma sin importar el aplicativo o los requerimientos.

3.1 Soluciones de plataforma de la nube de Google

Los modelos de servicios están relacionados con el tipo de solución que ofrece la nube de Google, definido con el tipo de infraestructura que requerimos implementar dentro de la nube de Google, es decir; si requerimos una infraestructura totalmente gestionable la mejor solución es Compute Engine, debido a que se puede configurar por completo las características de una máquina virtual, siendo la unidad base del modelo de IaaS. En el extremo contrario se puede conseguir una infraestructura dinámica, esto debido a que se pueden obtener servicios totalmente específicos y desarrollados que podremos agregar a una infraestructura ya establecida de forma muy rápida y automática.

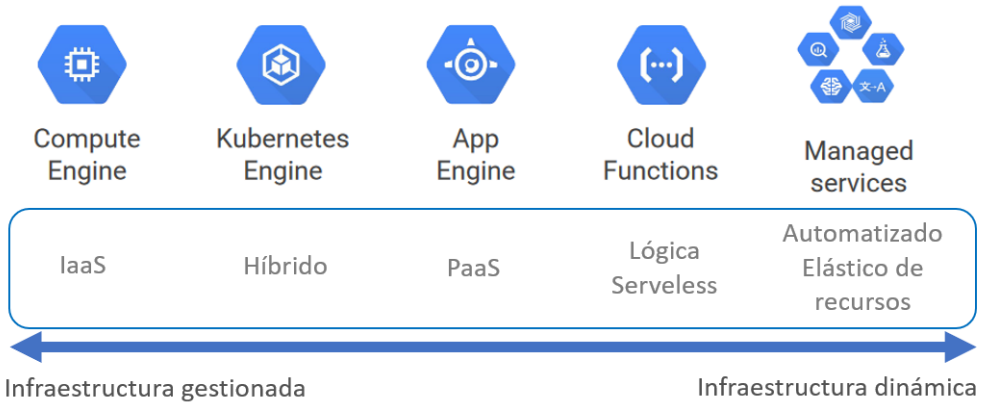

Figura 3.1: Ofertas según el modelo de servicios en la nube de Google. Logos extraídos de: Google Cloud

3.1.1 Soluciones para la implementación de contenedores en la nube de Google

En relación con las ofertas que nos permiten la implementación de contenedores dentro de nuestras infraestructuras, la nube de Google ofrece tres soluciones de cómputo como se puede observar en la figura 3.2. En el desarrollo de este capítulo entenderemos cuáles son las diferencias que existen entre cada una de las ofertas, reconociendo la unidad básica de la oferta, la acción que podemos realizar sobre la unidad básica y recrear la implementación de un mismo servicio en las tres diferentes ofertas.

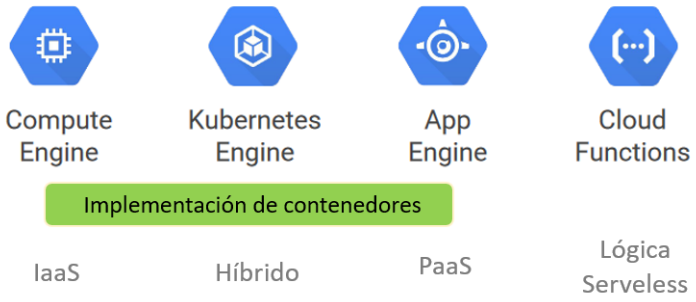

Figura 3.2: Soluciones de cómputo en la nube de Google. Logos extraídos de: Google Cloud

3.2 Compute Engine (IaaS)

Dentro de las ofertas de IaaS se puede obtener computación en bruto, almacenamiento y red, por medio de Compute Engine. Dichos componentes están organizados en centros de datos físicos y virtualizados por lo solo pagará por lo que asignamos [9]. Compute Engine nos otorga las siguientes 4 características:

- Máquinas virtuales totalmente personalizadas
- Discos persistentes y SSD locales opcionales
- Equilibrio de carga global y autoescalado
- Facturación por segundo

Universidad Militar Nueva Granada

[9] Google Cloud "Productos", Compute Engine, 2020.

3.2.1 Recursos de Compute Engine

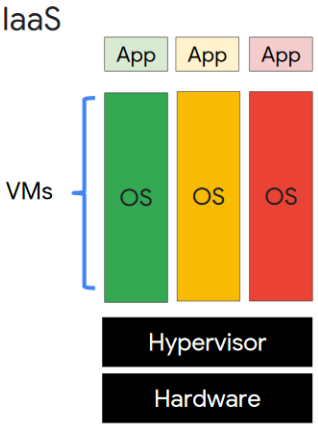

Figura 3.3: Virtualización de sistemas operativos en IaaS.

La unidad o componente más pequeño dentro de Compute Engine es una VM (Máquina virtual). El sistema operativo de dichos VMs puede llegar a tener un tamaño de incluso terabytes, causando un arranque de la VMs de minutos. Pero a su favor se tiene un sistema totalmente configurable a elección. Gracias a esta infraestructura se puede implementar servidores web, bases de datos o middlewares. Como se observa en la figura 3.3, cualquier implementación (App) siempre tendrá asignada una única VM. La implementación siempre dependerá del sistema operativo que es virtualizado por Hypervisor, afectando el despliegue de la aplicación en cuanto a los recursos que son asignados por Hypervisor, igual que el sistema operativo no asigna todos sus recursos a la aplicación debido a que realiza otros procesos paralelos para ejecutar la aplicación, así que la aplicación debe esperar a que el sistema operativo se inicie por completo para ser ejecutada eficientemente.

3.2.2 Implementación y casos de uso de Compute Engine

Arquitectura básica de una página web implementada en GCP (Plaforma de la nube de Google) con Compute Engine (figura 3.4)

1. El desarrollo del código de la página web puede ser realizado externo o internamente de la plataforma de Google Cloud.
2. Compute Engine nos permitirá implementar el código como si lo hiciera desde un equipo físico.
3. En este caso la VM-1 ocupa el código original de la implementación. Si por alguna razón, el tráfico y la cantidad de usuarios aumenta en la página web podremos aplicar un balanceador de carga, la oferta de Compute Engine nos permite internamente crear cuotas de recursos. Que una vez sobrepase el consumo de recursos de la oferta, podemos programar la activación de copias de la VM para mantener un servicio óptimo de nuestra aplicación así que la VM-2 es una copia de exacta de su versión original VM-1.
4. Si por alguna razón no se logra establecer o configurar algún tipo de servicio dentro de nuestra infraestructura en Compute Engine, GCP cuenta con un gran catálogo de servicios que podremos asociar. Pero con la desventaja de que dependemos enteramente la

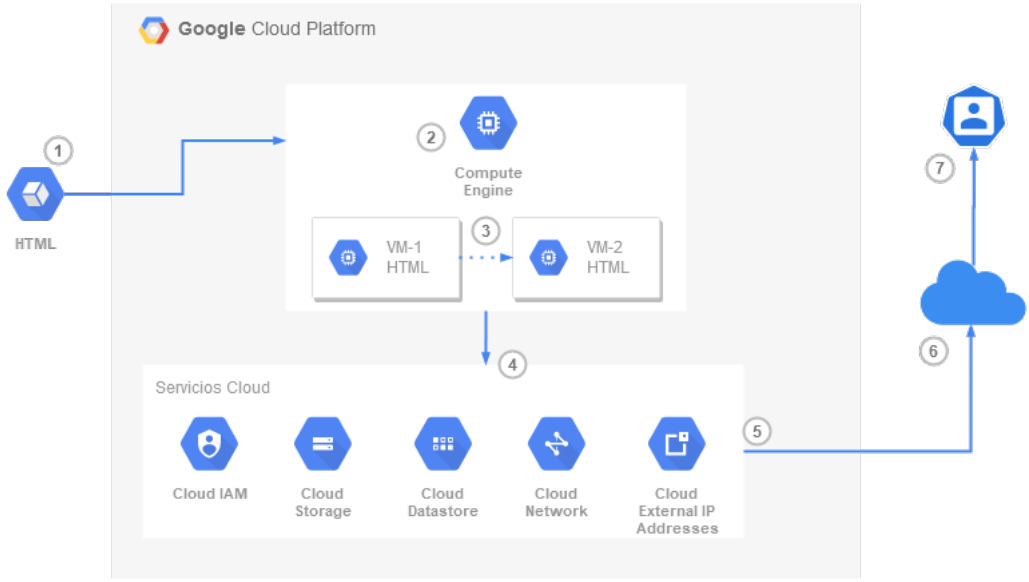

Figura 3.4: Arquitectura simple de una implementación de HTML con Compute Engine.

configuración interna, permisividad de gestión o cobro de dichos servicios. La eficiencia y la reducción de costos dentro de la infraestructura está en solicitar la menor cantidad servicios fuera de Compute Engine y configurar en su totalidad el servicio de la página web dentro de Compute Engine.

5. La plataforma de Google Cloud otorga servicios de asignación de IPs públicas para la exposición de nuestro servicio en internet.
6. Una vez expuesto el servicio este puede ser asociado a red privada, pública u otros proveedores de nube.
7. Finalmente el servicio es ofrecido como una página web y según la configuración de cuotas de recursos o la cantidad tráfico que reciba la misma, la infraestructura puede cambiar como se ejemplifica en el paso 3.

Casos de uso de Compute Engine

La plataforma de Google Cloud establece 3 casos por defecto en el cuales es más eficiente utilizar la oferta de Compute Engine:

- Cuando se requiere control completo sobre el sistema operativo y el hardware virtual
- Para realizar migraciones de ascenso y descenso a la nube
- Cuando necesitamos más flexibilidad y permisividad administrativa en nuestra infraestructura

3.3 App Engine (PaaS)

En relación con PaaS, el cliente puede vincular el código de sus bibliotecas para que posteriormente App Engine le proporcione acceso a la infraestructura que la aplicación necesita, lo que le permite centrarse en la lógica de la aplicación para que el cliente solo pague por lo que utiliza [10], App Engine proporciona las siguientes características:

- Una plataforma completamente administrada de código
- Agiliza el despliegue y la escalabilidad de las aplicaciones
- Proporciona soporte para los principales lenguajes de programación
- Apoya el monitoreo integrado, el registro y el diagnóstico

- Simplifica el control de versiones y pruebas de tráfico

[10] Google Cloud "Serverless computing", Documentación de Google App Engine, 2020.

3.3.1 Recursos de App Engine

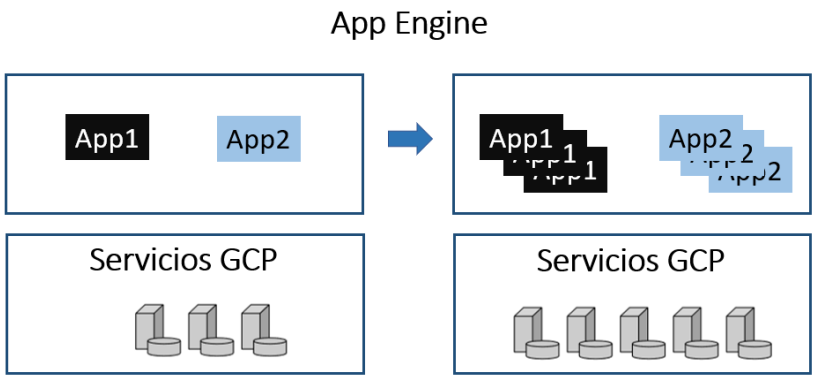

Figura 3.5: Infraestructura de App Engine.

Observando la figura 3.5 cada parte es un código configurado como cargas de trabajo autónomas que están usan servicios. A medida que se aumenta la demanda de su aplicación, la plataforma la escala independientemente de la carga de trabajo y la infraestructura, pero esto conlleva a que siempre dependas de la forma en que está compuesta la arquitectura subyacente de los servicios.

3.3.2 Implementación y casos de uso de App Engine

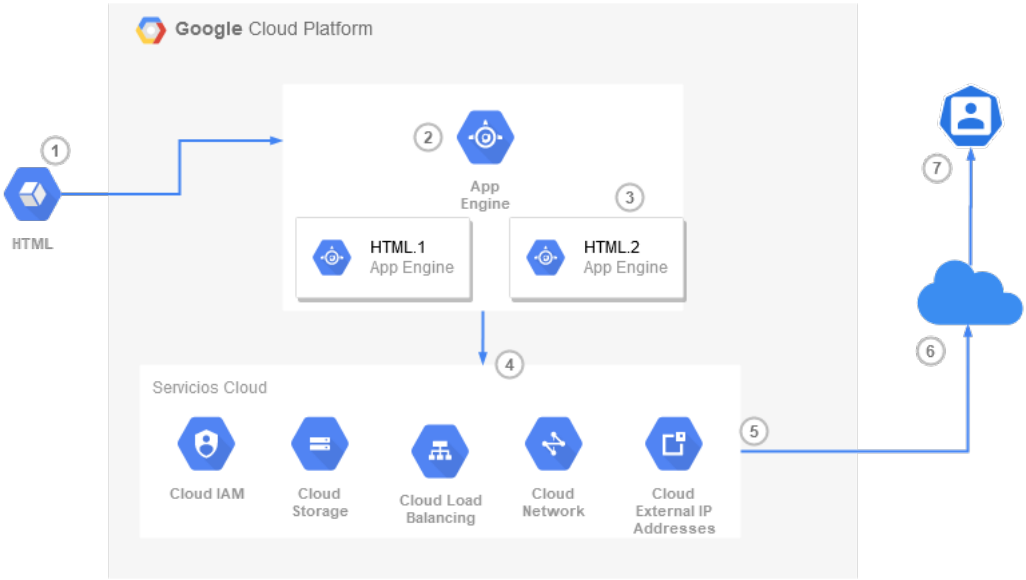

Figura 3.6: Arquitectura simple de una implementación de HTML con App Engine.

Arquitectura básica de una página web implementada en GCP (Plataforma de la nube de Google) con App Engine (figura 3.6)

1. El desarrollo del código de la página web puede ser realizado externo o internamente de la plataforma de Google Cloud.
2. App Engine permitirá adjuntar el código HTML y mantener un registro de las versiones como también soporte sobre el código.
3. En este caso el balanceo de carga deberá realizarse asociado el servicio externamente a App Engine, debido a que este se encarga únicamente del código de nuestro servicio web.
4. App Engine funciona como un sistema de APIs (Interfaz de programación de aplicaciones) para que el acople con servicios externos sea fácil y rápido.
5. Para que nuestro servicio funcione de la misma forma que se hace en Compute Engine, deben ser asociados varios servicios externos con la ventaja de evitar su compleja configuración pero con la desventaja de un mayor costo de implementación.
6. Una vez expuesto el servicio este puede ser asociado a red privada, pública u otros proveedores de nube.
7. Finalmente el servicio es ofrecido como una página web, la eficiencia de la prestación del servicio dependerá de la calidad de control sobre el código con App Engine y la asociación eficiente de los correctos servicios externos.

Casos de uso de App Engine

La plataforma de Google Cloud establece 3 casos por defecto en el cuales es más eficiente utilizar la oferta de App Engine:

- Sitios web
- Aplicación móvil y backends de juegos
- RESTful APIs

3.4 Kubernetes Engine (Solución Híbrida)

Ya teniendo claras las ofertas de IaaS y PaaS de la plataforma de Google Cloud. Con Kubernetes Engine se busca obtener cualidades de ambas infraestructuras siendo una oferta híbrida, teniendo una administración conjunta y total de cargas de trabajo individuales, más una configuración absoluta de la infraestructura [11]. Kubernetes Engine se caracteriza por las siguientes cualidades:

- Plataforma de Kubernetes totalmente gestionada
- Soporta la escalabilidad de clúster, discos persistentes, actualizaciones automáticas y reparaciones de nodos (VM)
- Integración incorporada con los servicios de la plataforma de Google Cloud
- Portabilidad a través de múltiples entornos
  - Computación híbrida
  - Computación multi-nube

[11] Google Cloud "Kubernetes Engine", Descripción general de GKE, 2020.

3.4.1 Recursos de Kubernetes Engine

La función más simple del sistema de Kubernetes Engine es permitir la escalabilidad independiente de cada carga de trabajo como en PaaS y una capa de configuración del sistema operativo y el hardware como en IaaS. Lo que precisa esto es la creación de una caja invisible que contiene un código y sus dependencias, con acceso limitado a su propia partición del sistema de archivos y el hardware. Esto es representado en la primera parte de las cuatro en la figura 3.7.

En la segunda parte de la figura 3.7 solo se requiere que el host sea una partición del SO (sistema operativo) y Hardware que soporta a los contenedores. La idea es que se virtualiza el SO para escalar al modo que lo hace PaaS, pero permitiendo la flexibilidad que nos da IaaS.

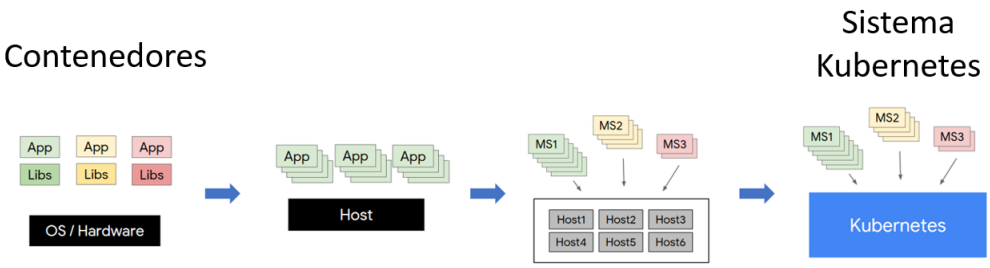

Figura 3.7: Sistema de Kubernetes para contenedores.

En la parte 3 de la figura 3.7 el código obtiene un grado de portabilidad y versatilidad permitiéndole un escalado y activación optimizados. Lo más probable es que queramos construir nuestras aplicaciones utilizando más de un contenedor, como lo hace una arquitectura de microservicios. Al construir de esta forma se puede conectar como una red, creando despliegues y escalonamiento eficiente dentro de un grupo de host, de ese modo Kubernetes Engine actúa como una herramienta que ayuda a facilitar la orquestación de muchos contenedores en muchos hosts.

3.4.2 Implementación y casos de uso de Kubernetes Engine

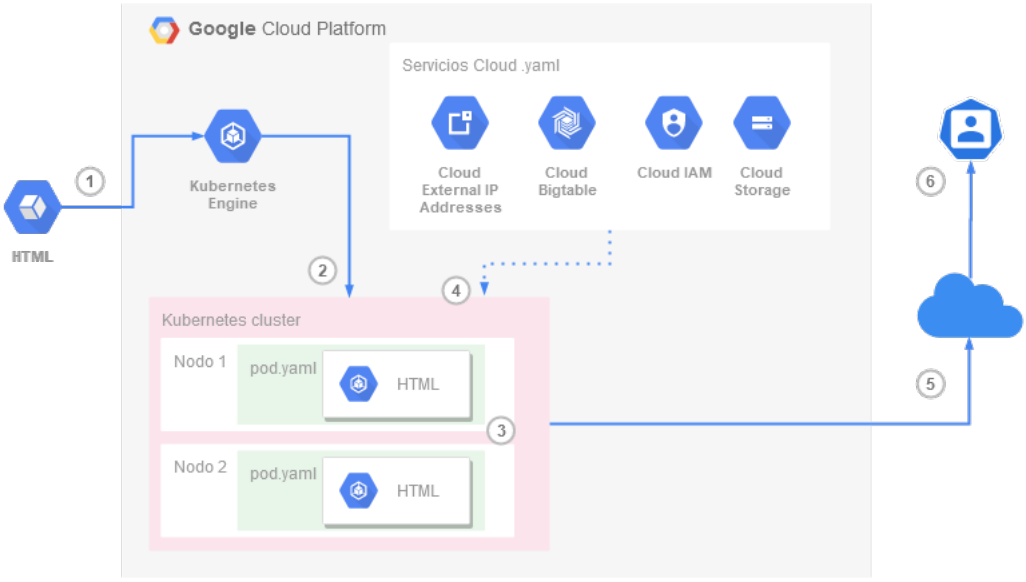

Figura 3.8: Arquitectura simple de una implementación de HTML con contenedores en Kubernetes Engine.

Arquitectura básica de una página web implementada en GCP (Plataforma de la nube de Google) con Kubernetes Engine (figura 3.8)

1. El código HTML esta contenerizado en un contenedor linux, donde posteriormente asociaremos a un Pod.yaml para ser implementado en la oferta de Kubernetes Engine.

2. El sistema de Kubernetes Engine tiene la permisividad y gestión que se tiene con Compute Engine debido a que nuestro contenedor está asociado a un nodo (VM).
3. La configuración y administración sobre el contenedor se hace por medio de código Yaml y gracias a la declaración de estados en los archivos Yaml, el sistema de Kubernetes Engine mantendrá dicho estado. Si se declara que se debe mantener un tráfico óptimo en el servicio y por alguna razón el tráfico aumenta y satura la página web, el sistema Kubernetes ejecutara un balanceo de carga automático creando una copia del contenedor igual en un nodo distinto, pero con una rapidez considerable a la de Compute Engine debido a que el nodo dedica su capacidad de cómputo a los procesos del contenedor exclusivamente.
4. A diferencia de App Engine son muy pocos los servicios externos a Kubernetes Engine que deben ser asociados para que el servicio sea eficiente, si se requieren servicios estos simplemente se configuran por medido de archivos Yaml.
5. Una vez expuesto el servicio este puede ser asociado a red privada, pública u otros proveedores de nube.
6. Finalmente el servicio es ofrecido como una página web, la administración y el control del servicio se hace por medio del sistema de Kubernetes Engine.

Casos de uso de App Engine

La plataforma de Google Cloud establece 3 casos por defecto en el cuales es más eficiente utilizar la oferta de Kubernetes Engine:

- Aplicaciones contenerizadas
- Sistemas distribuidos de la nube nativa
- Aplicaciones híbridas

SCIENTIÆ · PATRIÆ · FAMILIÆ
UNIVERSIDAD MILITAR
NUEVA GRANADA

4. Caracterización de la arquitectura

Kubernetes es un sistema que permite una administración completa de la infraestructura, debido a la declaración de estados por parte del administrador y de Kubernetes. Los estados declarados permitirán que el objeto Pod, pueda ejecutar y desplegar con éxito los contenedores. El plano de control otorgará soporte a la arquitectura del clúster realizando procesos internos para ejecutar y mantener activos los Pods, procesos que son liderados por un recurso denominado el nodo Maestro.

Kubernetes permite desplegar objetos dentro del clúster por medio de archivos programados con lenguaje Yaml. Un tipo de lenguaje que tiene la característica tener un sentido gramatical, posibilitando la lectura por parte de administrador y también del sistema Kubernetes. Los espacios de nombre son una etiqueta declarada dentro del archivo Yaml de un objeto para posicionarlo en una sección específica del clúster, generando diferentes espacios de nombre con la posibilidad de seccionar el aplicativo dentro del clúster.

Cuando los objetos y los recursos están desplegados en el clúster de Kubernetes, el mismo sistema presenta el comando Kubectl para monitoreo de toda la infraestructura. Kubectl tiene una sintaxis de 4 secciones que permiten comprender fácilmente la acción a realizar sobre los recursos u objetos. El monitoreo de Kubectl se realiza por simples consultas que presentan detalles sobre lo que está pasando con el recurso u objeto declarado en el comando Kubectl.

4.1 Conceptos básicos de Kubernetes

Este gestor permite automatizar el despliegue, el escalado, el equilibrio de la carga, el registro, vigilancia y otras características de gestión de las aplicaciones basadas en contenedores. Estas son características de soluciones típicas de PaaS y también de las características de IaaS, permitiendo cualquier tipo de preferencia y flexibilidad de configuración para los usuarios. Kubernetes soporta configuraciones declarativas, es decir; el administrar la infraestructura de forma declarativa, describiendo un estado que se desee alcanzar en los objetos dentro de Kubernetes, en lugar de emitir una serie de órdenes para lograr ese estado deseado.

4.1.1 Estado de los objetos

El contenedor es el elemento más pequeño dentro de infraestructura de Kubernetes, entonces el siguiente elemento es el que contiene al contenedor y son los llamados **Objetos**, estos objetos pueden administrar y contener más de un contenedor. Al observar la figura 4.1, los objetos son entidades persistentes que representan el estado de un grupo en este caso el de los contendores. Dentro del objeto existen dos tipos de características: la especificación del objeto y el estado del objeto.

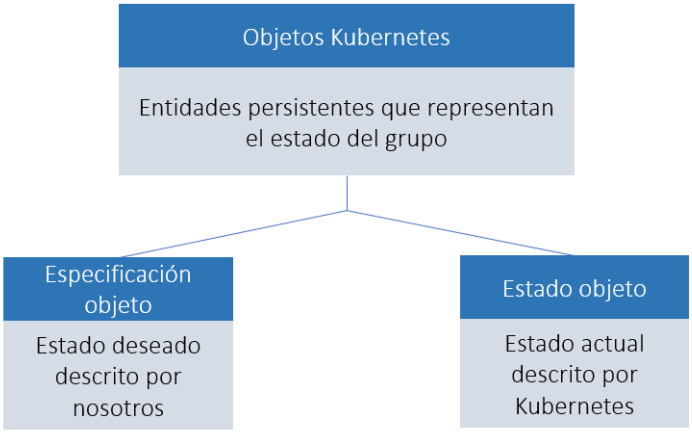

Figura 4.1: Elementos de un objeto en Kubernetes.

4.1.2 Objeto básico (Pod)

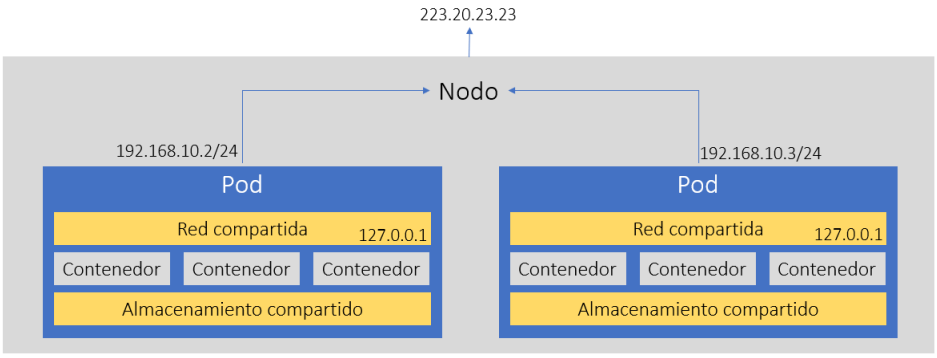

Figura 4.2: Elementos del Pod y conexión del Pod.

En capítulos anteriores se ha comentado que el contenedor es el elemento más pequeño dentro de la infraestructura de Kubernetes, pero esto no es así, ya que este es un elemento externo del sistema. El Pod es realmente el objeto más pequeño del sistema de Kubernetes, debido a que es la base de la construcción de otros objetos. En la figura 4.2 se observa que el Pod puede mantener dentro de si uno o varios contenedores, cuando hay más de un contenedor dentro de un Pod dichos contenedores están vinculados entre sí compartiendo recursos, almacenamiento y la red. El sistema de Kubernetes otorga una IP única a cada Pod, así que cada contenedor dentro

del Pod compartirá las mismas direcciones de IP y puertos de red, debido a ello los contenedores del mismo Pod pueden establecer una comunicación por medio del host local 127.0.0.1 [12].

[12] Google Cloud "Kubernetes Engine", Concepto de Pod en Kubernetes Engine, 2020.

4.1.3 Plano de control y procesos internos de la arquitectura

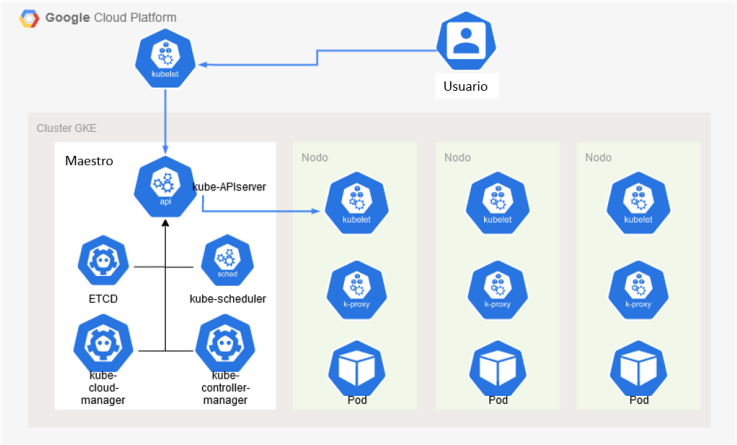

Figura 4.3: Arquitectura de Kubernetes Engine.

En la figura 4.3 se puede observar que existen dos tipos de áreas con diferentes componentes; uno de ellos es el **Maestro**, el encargado de coordinar los **Nodos**, estos a su vez son los encargados de ejecutar los Pods. Existen varios componentes dentro del Maestro y este grupo de componentes se le denominan **Plano de control de Kubernetes**[13].

[13] Google Cloud "Kubernetes Engine", Clústeres (Arquitectura de clúster), 2020.

kube-apiserver

Para lograr interactuar con kube-apiserver existe el comando kubectl, el trabajo de este comando es conectar con kube-apiserver y comunicarse con él usando la API de Kubernetes, así que kube-apiserver se encarga de:

- Aceptar comandos para ver o cambiar el estado del clúster.
- Verificar y aceptar el desplegué de Pods.
- Autenticar las solicitudes entrantes, determinar si está autorizada y si es válida.
- Gestionar el control de admisión.

etcd

Siendo la base de datos del clúster, su trabajo es almacenar de forma fiable el estado del mismo. Esto incluye todos los datos de configuración del clúster, que nodos forman parte del clúster, qué Pods deberían estar funcionando, y dónde deberían estar siendo ejecutados. El único que interactúa con etcd, es kube-apiserver haciéndolo en nombre del resto del sistema.

kube-scheduler

Programa los Pods en los nodos, evaluando los requisitos de cada Pod individual y seleccionado el nodo más adecuado. Cuando kube-scheduler descubre que el Pod aún no tiene asignado a un nodo, este le elige un nodo y posteriormente escribe el nombre de ese nodo en la asignación del Pod.

kube-controller-manager

Kube-controller-manager indefinidamente estará monitoreando el estado de un grupo a través del Kube-APIserver, siempre que el estado actual del clúster no coincida con el estado deseado y kube-controller-manager intentará hacer cambios para lograr el estado deseado.

kube-cloud-manager

Se encarga de manejar los controladores que interactúan con proveedores de nubes subyacentes. Por ejemplo, si se despliega manualmente un clúster de Kubernetes en otras ofertas de la nube de Google, kube-cloud-manager se encargaría de introducir a la infraestructura característica como balanceadores de carga y volúmenes de almacenamiento cuando se necesiten.

Plano de control en los nodos

Cuando el kube-apiserver quiere iniciar un pod en un nodo, se conecta a la kubelet de ese nodo. **Kubelet** usa el tiempo de ejecución del contenedor para iniciar el Pod y monitoriza su ciclo de vida. Y **kube-proxy** se encargará de mantener la conectividad de la red entre los Pods de un grupo.

4.2 Gestión básica de contenedores con Kubernetes

4.2.1 Código YAML para definir objetos e integrar los contenedores

Pod.yaml

Por medio de los archivos manifiestos de tipo Yaml o JSON se logra definir los objetos que deseamos que Kubernetes cree y administre. A diferencia de Json, Yaml es más legible por los humanos y más fácil de editar [14].

[14] Ben-Kiki, O., Evans, C., & Ingerson, B. (2009). Yaml ain't markup language.

```
apiVersion: apps/v1
kind: Pod
metadata:
    name: nginx
    labels:
        app: nginx
spec:
    containers:
    - name: nginx
        image:
        nginx:latest
```

Figura 4.4: Archivo YAML de un Pod.

En la figura 4.4 se puede observar la descripción del objeto más básico de Kubernetes, la primera línea de código describe la versión de API de Kubernetes que se utiliza para crear el objeto, con **kind** en la segunda línea de código podemos identificar el tipo de objeto, en la **metadata** se establece el nombre del objeto, un número de identificación único. Finalmente en la última sección del archivo se establecen los nombres e imágenes de los contenedores, que tiene asignado el Pod para ser ejecutados. Se debe tener en cuenta que se pueden definir varios objetos dentro de un solo archivo Yaml y para ello se establecen las carga de trabajo de Kubernetes un tema que se tratara a profundidad en el siguiente capítulo.

Identificación y nombre de un objeto

Cualquier objeto dentro de Kubernetes permite ser nombrado con caracteres alfanuméricos, guiones, puntos y con un máximo de 253 caracteres. Una vez creado el objeto, Kubernetes le asigna un número **UID** que es único e irrepetible mientras este activo el clúster. No pueden existir dos objetos iguales con nombres iguales, pero si un objeto es borrado su nombre puede ser utilizado en otro objeto.

Ciclo de vida del Pod

Dentro del ciclo de vida de un Pod no está estipulada su reparación esto debido a que el Pod está diseñado para ser efímero y desechable, así que una vez que el Pod presentar un fallo este es dado de baja, el plano de control se encargara crear y activar una copia igual del Pod que fallo.

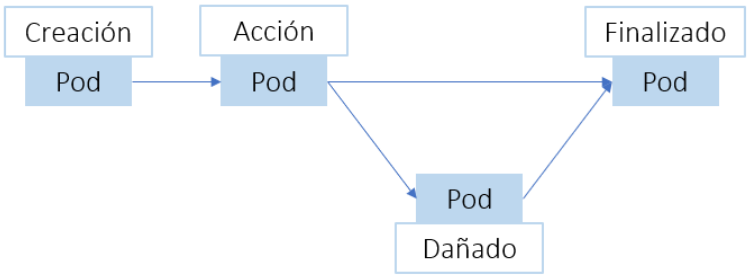

Figura 4.5: Ciclo de vida del Pod.

4.2.2 Espacios de nombres: Creación de áreas de trabajo

Por medio de los **Espacios de nombres** de Kubernetes se puede crear el mismo objeto con el mismo nombre múltiples veces en un solo nodo, con la condición de dichos objetos no deben estar en el mismo Espacios de nombres [15]. A modo de ejemplo en la figura 4.6 se establece un clúster con tres Espacios de nombres: Test, Stage, Prod. No se puede crear un **nginx Pod** en el nodo central debido a que pertenece al mismo Espacio de nombre en el que está el **nginx Pod** del nodo izquierdo. La principal función de los Espacios de nombres es permitir implementar cuotas de recursos, para definir los límites de consumo de recursos en cada Espacio de nombre. También permiten recrear una gestión por áreas de trabajo a la hora de implementar los objetos, dado el caso se puede establecer un Espacio de nombre para hacer pruebas de rendimiento con los mismos objetos que están en otro Espacio de nombre, en el cual está implementada la aplicación. No es necesario ni obligatorio utilizar los Espacios de nombres en la implementación de Kubernetes.

[15] namespaces(7) - Linux manual page, 2021.

4.2.3 Otros objetos de Kubernetes

Una de las cualidades que queremos aprovechar cuando realizamos implementaciones en la nube, es la escalabilidad de la infraestructura. Si observamos la figura 4.7 se determina que hay un Pod escalado 2 veces, esto debido a que se hicieron copias exactas del objeto dentro de la infraestructura. La duplicación de los Pods debe realizarse declarando archivos YAML

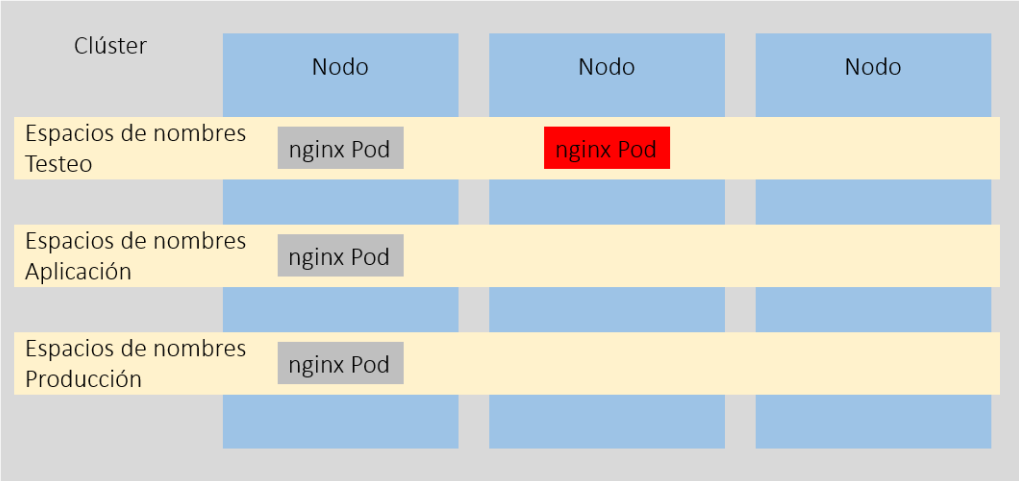

Figura 4.6: Distribución de objetos por medio de Espacios de nombres.

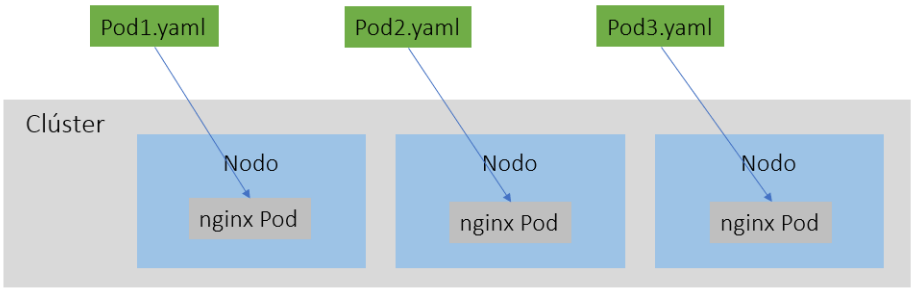

Figura 4.7: Despliegue de Pod uno a uno.

manualmente, como se observa en la figura 4.7. Si por alguna razón se desea aumentar el número de Pods iguales, este método se vuelve extenso y no óptimo.

Para estos casos intervienen objetos más avanzados que el Pod, pero basados en el Pod, estos nuevos objetos permiten manejar múltiples Pods por medio de un solo archivo declarativo tipo yaml:

- Con **ReplicaSet** aseguramos que un grupo de Pods, iguales entre sí, estén funcionando todas al mismo tiempo.
- Los **Despliegue** permiten hacer actualizaciones declarativas para crear, actualizar, retroceder y escalar Pods. Y por medio de ReplicaSet, los Despliegues controlan los grupos de Pods.
- Con los **Trabajos** se crean uno o más Pods necesarios para ejecutar una tarea, una vez se finaliza la tarea los Pods son finalizados. También se pueden determinar el tiempo que puede estar activo el Pod y a este objeto se le denomina un **Crono-trabajo**.

La figura 4.8 muestra que la arquitectura de Kubernetes esta divido en dos grandes secciones, la primera sección, que gestiona los procesos dentro del máster y de los nodos. La segunda sección, son los objetos que ejecutan y administran los contenedores por medio de archivos YAML.

Universidad Militar Nueva Granada

Estado deseado (Objetos de Kubernetes)	Plano de control de Kubernetes (Procesos de gestión)
Pod	API Server
ReplicaSet	Kube-scheduler
Despliegue	Kube-controller-manager
Servicio	Kube-cloud-manager
StatefulSet	Kubelet
DaemonSet	Kube-proxy
Trabajo	
Crono-trabajo	

Figura 4.8: Componentes y objetos de la arquitectura de Kubernetes.

4.3 Supervisión de objetos en Kubernetes

Kubectl es el principal comando de acciones de supervisión para Kubernetes, con dicho comando podremos realizar acciones sobre clúster y los objetos de Kubernetes [16].

[16] Buchanan, S., Rangama, J., & Bellavance, N. (2020). kubectl Overview. In Introducing Azure Kubernetes Service (pp. 51-62).

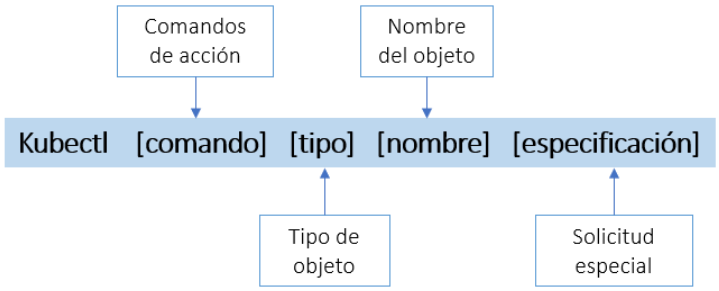

Figura 4.9: Estructura del comando kubectl.

Son cuatro las partes que conforman la sintaxis del comando kubectl, en la primera parte existen cuatro principales opciones de acción, como lo son get, describe, exec y logs, algunos comandos permitirán la visualización de información y otros cambiar la configuración del clúster. En la segunda parte **tipo**, se podrá seleccionar el tipo de objeto, al cual le aplicaremos la acción seleccionada en la sección de **comando**. En la tercera parte **nombre**, se hace la denotación específica de a que objeto en particular se le hará la acción, si quisiéramos ver el estado de todos los Pods no se debería agregar un nombre al comando kubectl y en consecuencia el comando kubectl nos mostraría una lista de todos los Pods con sus estados. Finalmente en la parte de **especificación** se agrega una petición especial, como podría ser la forma de visualizar o guardar la información que se obtuvo con la acción de kubectl.

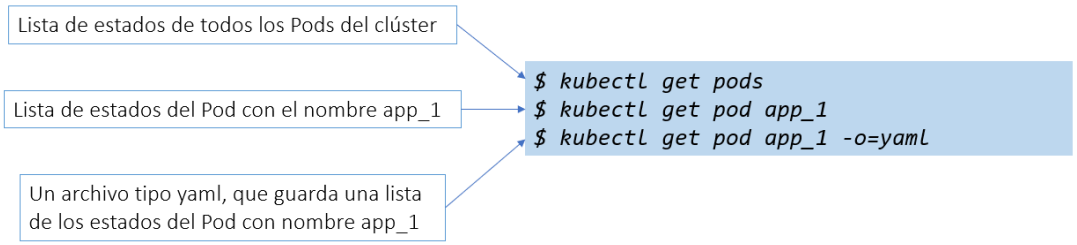

Figura 4.10: Ejemplo de utilización del comando kubectl.

4.3.1 Estados con get

Se utiliza el comando get para ver un resumen de que está pasando con el Pod, no se observan detalles solo una palabra que describe un estado dentro del ciclo de vida del Pod, por ejemplo:

Figura 4.11: Comando get.

- Pending: Pendiente significa que el Pod ha sido aceptado por Kubernetes, pero aún falta que sea programado en un nodo.
- Running: el Pod está funcionando y ha sido asignado con éxito a un nodo y todos sus contenedores han sido creados, pero el estado del contenedor es pendiente.
- Succeeded: significa que todos los contenedores han terminado de funcionar con éxito dentro del Pod.
- Failed: significa que un contenedor ha fallado, y no puede ser reiniciado.
- Unknown: el estado del Pod simplemente no puede ser reconocido por Kubernetes, causado por un fallo de comunicación en Kubernetes.
- CrashLoopBackOff: significa que el Pod no está configurado correctamente.

4.3.2 Detalles con describe

Describe es la versión en detalle de get, dando información sobre el Pod y sus contenedores, detallando la información del contenedor y del Pod, como el estado, la preparación, el recuento de reinicios, y los eventos.

4.3.3 Interacciones con exec

Este comando permite interactuar con la consola del Pod, permitiendo la ejecución de comando Shell, es útil cuando se requiere resolver un problema de configuración interna del contenedor, de conectividad u otras aplicaciones.

```
$ kubectl describe pod [Nombre_del_Pod]
```

Pod:
- Nombre
- Namespace
- Nombre del nodo
- Etiquetas
- Estatus
- IP address

Figura 4.12: Comando describe.

```
$ kubectl exec [Nombre_del_Pod] -- [Comando_bash]
$ kubectl exec app_1 -- ls /
bin
boot
dev
etc
home
lib
lib64
media
mnt
opt
proc
root
```

Figura 4.13: Comando exec.

4.3.4 Histórico con logs

Se pueden ver los registros de un Pod y de los contenedores dentro, los registros contienen tanto la salida estándar como los mensajes de error estándar que tienen las aplicaciones dentro del contenedor.

```
$ kubectl logs [Nombre_del_Pod]
```

Los registros del Pod incluyen:
- stdout: Salida estándar en la consola
- stderr: Mensajes de error

Figura 4.14: Comando logs.

SCIENTIÆ · PATRIÆ · FAMILIÆ
UNIVERSIDAD MILITAR
NUEVA GRANADA

5. Gestión de objetos y clúster en Kubernete

El despliegue es el objeto de Kubernetes que permite implementar varios Pods iguales al mismo tiempo. La gestión conjunta de todos los Pods de un Despliegue permite aplicar escalado o autoescalado a los Pods que se administran, además de permitir diferentes tipos de actualización del Despliegue según los requerimientos del aplicativo.

Los trabajos y crono-trabajos son objetos de Kubernetes que se caracterizan por realizar tareas temporales o periódicas. Existen subcategorías de estos objetos que permiten realizar varias tareas para reducir la utilización de recursos o dedicar varios Pods a la misma tarea para generar redundancia en el aplicativo.

La gestión del clúster permite la asignación automática o no, de recursos computacionales a los Pods según los requerimientos de la implementación. La sincronización de la gestión del clúster con la de los objetos, permitirá que los objetos puedan ser desplegados con eficiencia. Además de permitir que los objetos se puedan conectar a otros objetos o a diferentes tipos de redes.

5.1 Administración de múltiples Pods

La implementación de una aplicación por medio de Pods individuales puede ser riguroso, ya que se debe declarar los estados de cada Pod y luego incorporarlo al clúster, Pod por Pod. Para aplicaciones que son constituidas con pocos contenedores es rentable, pero para aplicaciones robustas se vuelve un poco ineficiente, ya que tendríamos que escalar y actualizar individualmente cada Pod en aplicaciones que podrían llegar a tener más de 10 Pods. Existe un objeto llamado Despliegue, que permite administrar varios Pods al mismo tiempo, haciendo más fácil implementar aplicaciones que están fraccionadas en varios contenedores. El Despliegue como cualquier otro objeto de Kubernetes deben ser declarados por medio de un archivo Yaml, que debe contener las características de los Pods. Una vez que se activa el Despliegue (Objeto Kubernetes), el sistema de Kubernetes crea un controlador del Despliegue, que es el responsable de crear y mantener el estado a lo largo de la vida del clúster. Durante el proceso de creación de los Pod, se crea un ReplicaSet, que es el encargado de asegurarse que estén activos un cierto número de Pods de una versión especifica [17].

R [17] kubernetes.io, Documentación (Deployment), 2020.

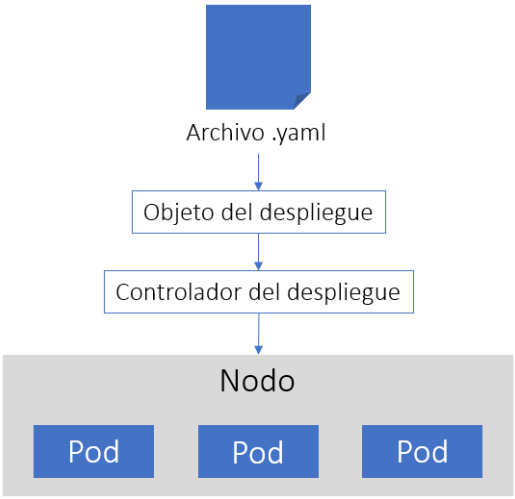

Figura 5.1: Estructura de un Despliegue.

5.1.1 Creación de un Despliegue

En la figura 5.2, se puede observar un ejemplo de un archivo .YAML para declarar estados por medio de un Despliegue, el nombre del Despliegue es **app_html**, con una cantidad de 3 Pods replicados, en la especificación del Pod se establece la imagen del contenedor que es extraída de **Google Container Registry**, la transmisión y recepción del tráfico se establece por medio del **puerto 8080**.

Archivo YAML

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app_html
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: app_html
    spec:
      containers:
      - name: app
        image: gcr.io/demo/app_html:1.0
        ports:
        - containerPort: 8080
```

Figura 5.2: Ejemplo de un archivo .YAML de un Despliegue

Formas de creación de un Despliegue

Ejecución de archivos YAML

En la plataforma de Google Cloud se puede crear un Despliegue de tres formas diferentes, la primera de ellas es teniendo un archivo Yaml del objeto del Despliegue, donde simplemente aplicamos el comando de la figura 5.3 para que Kubernetes cree el Despliegue.

```
$ kubectl apply -f [Archivo_del_Despliegue]
```

Figura 5.3: Comando de aplicación de un Despliegue por archivo.yaml.

Creación del archivo YAML desde shell

La segunda opción es crear el Despliegue desde la consola de comando shell que nos provee Google en la plataforma de Google Cloud, en el cual no es necesario tener un archivo Yaml del Despliegue, por este método se crea el archivo Yaml del Despliegue como se observa en la figura 5.4.

```
$ kubectl run [Nombre_del_Despliegue] \
--image [Imagen]:[Versión] \
--replicas 3 \
--labels [CLAVE]=[Valor] \
--port 8080 \
--generator deployment/apps.v1 \
--save-config
```

Figura 5.4: Comando de creación de un Despliegue desde comando shell.

Por plataforma GCP

La última opción es la más fácil, debido a que se hace directamente desde la API que nos otorga la plataforma de Google Cloud, como se observa en la figura 5.5, sin tener que interactuar con el comando shell. Debido a que la administración de contenedores por medio de Kubernetes se realiza con diferentes objetos, es importante aprender a identificar la estructura e implementación de un archivo Yaml, por ello se recomienda hacer la implementación de estos objetos por medio del comando shell para lograr aumentar y fortalecer los conocimientos sobre Kubernetes.

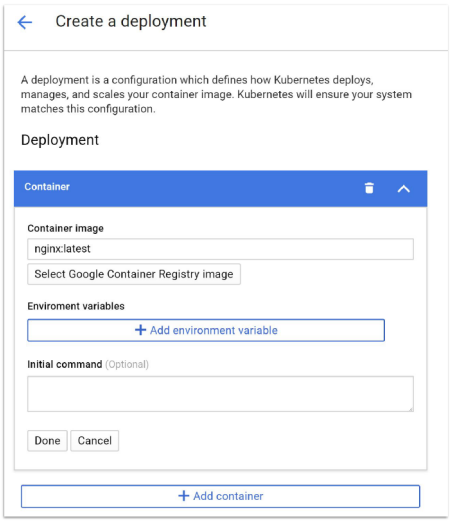

Figura 5.5: Creación de un Despliegue desde la API de la plataforma de Google Cloud.

5.1.2 Escalado y autoescalado

Una de las gestiones más básicas para los contenedores, es poder aumentarlos por medio del escalado. Permitiendo aumentar el número de Pods y contenedores de la infraestructura. El escalado es conveniente cuando se está monitoreando el consumo de procesamiento de la aplicación, debido a que se hace manualmente y la acción se ejecuta inmediatamente, el autoescalado es una cualidad de la nube nativa, ya que permite establecer un escalado automático, que es programado previamente y solo se efectúa si cumple con ciertos requisitos.

Escalado

Un Despliegue es el objeto de Kubernetes, en el cual se mantiene una cierta cantidad de Pods con un mismo estado, esto para evitar crear Pod a Pod. Es normal que una vez se crea un Despliegue con una cantidad de Pods, se desee aumentar dicha cantidad. Para ello se puede escalar la cantidad de Pods por tres métodos, editando el archivo YAML del Despliegue, con API de la plataforma de Google Cloud o simplemente aplicando el comando de la figura 5.6, donde se escoge la cantidad de réplicas de Pods.

```
$ kubectl scale deployment [Nombre_del_Despliegue]-
replicas=5
```

Figura 5.6: Comando para escalado de Pods en un Despliegue.

Autoescalado

Una de las cualidades de la nube nativa es la automatización de los procesos de administración, como el escalado de la infraestructura. Así que Kubernetes tiene integrado el auto-escalado de los Despliegues, el cual se hace con el comando de la figura 5.7, con 3 parámetros de configuración:

- Min = número de Pod activos en el Despliegue.
- Max = número de Pod que pueden ser activados dentro del Despliegue.
- Cpu-percent = porcentaje de utilización mínima de la CPU, para poder realizar el escalado.

Se debe de tener en cuenta que solo se está escalado una pequeña parte del clúster o para ser más específico; las aplicaciones que se están ejecutando con los contenedores, que están siendo gestionado con el Despliegue [18].

```
$ kubectl autoscale deployment [Nombre_del_Despliegue]
--min=5 --max=15 --cpu-percent=75
```

Figura 5.7: Comando para auto-escalado de un Despliegue.

[18] Sayfan, G. (2019). Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes (pp 21).

5.1.3 Actualización

Una vez que se implementa una aplicación, es normal que se tengan que realizar actualizaciones sobre la aplicación, pero depende de que tipo de aplicación o servicio se está ofreciendo, para lograr un eficiente administración se debe comprender que tipo de estrategia de actualización se debe aplicar según el tipo de servicio o aplicación se tiene, por ejemplo:

- Cuando es necesario que el servicio sea suspendido para realizar una actualización.
- Cuando no se puede suspender la aplicación para hacer actualizaciones.
- Cuando se desea que el tráfico de los usuarios se haga gradualmente entre la antigua y nueva versión de la aplicación durante la actualización.

para cada uno de estos casos existe un tipo de estrategia de actualización, pero primero se debe entender como se hace una actualización sin estrategia y que hace el sistema Kubernetes durante una actualización, para posteriormente comprender como aplicar estrategias de actualización.

Actualización sin estrategia

Las actualizaciones sin estrategia de una Implementación solo se pueden aplicar cuando los cambios se hacen en las especificaciones. Las actualizaciones se puede hacer de diferentes formas:

- Comando **apply**: la actualización se lleva a cabo con un nuevo archivo Yaml actualizado, permitiendo actualizar cualquier contendido de la especificación del Despliegue.
- Comando **set image**: se actualiza únicamente la imagen del contenedor.
- Comando **edit**: permitirá realizar cambios directamente sobre el archivo Yaml por medio del editor código de shell en la plataforma de Google Cloud, las actualizaciones solo se aplicarán una vez se guarden los cambios.

```
$ kubectl apply -f [Archivo_del_Deployment]
```

```
$ kubectl set image deployment
[Nombre_del_Deployment][Imagen][Imagen]:[Ve
rsión]
```

```
$ kubectl edit \
deployment/[Nombre_del_Deployment]
```

Figura 5.8: Métodos de actualización de un Despliegue.

Proceso de actualización

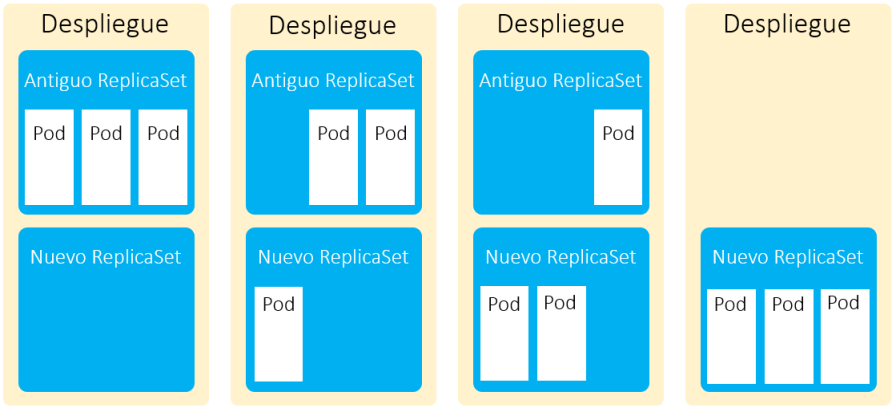

Figura 5.9: Proceso detrás de una actualización del Despliegue.

Recordemos que un Despliegue es el que permite realizar actualizaciones sobre el grupo de Pods, que se están desplegando, y el ReplicaSet es el encargado crear y mantener los Pods del

Despliegue, Así que, cada vez que se actualiza un Despliegue este activa un nuevo ReplicaSet con un nuevo conjunto de Pods. Como se observa en la figura 5.9, el proceso de actualización se hace gradualmente, activando Pods en el nuevo ReplicaSet y desactivando Pods en el antiguo ReplicaSet. La utilización de esta estrategia de actualización permite disponibilidad continua durante toda la actualización, pero cuando se tiene una gran cantidad de Pods, el proceso llegar a ser lento y no existe un control sobre el tráfico como con otras estrategias de actualización.

Actualización continua

La Actualización continua es una de las estrategias de actualización que son aplicadas al Despliegue, en esta estrategia se mantienen dos parámetros claves, **Max unavailabel** y **Max surge**, en el primer parámetro se establece la cantidad máxima de Pods que pueden ser creados al mismo tiempo en el nuevo ReplicaSet. En el segundo parámetro se establece la cantidad máxima de Pods (no disponibles) durante la actualización, independiente si son del antiguo o nuevo ReplicaSet. Los valores asignados a los parámetros pueden ser en números enteros o en porcentajes. Dentro de archivo Yaml del Despliegue podemos configurar la estrategia de Actualización continua, en las especificaciones del Pod, como se observa en la figura 5.10.

```
[...]
kind: deployment
spec:
  replicas: 10
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 5
      maxUnavailable: 10%
[...]
```

Figura 5.10: Archivo .YAML de un Despliegue con Actualización continua.

Si se implementara el archivo YAML de la figura 5.10, el proceso de actualización sería el que se observa en la 5.11, el proceso se inicia creando el nuevo ReplicaSet con la cantidad máxima de Pods que se permiten simultáneamente, con un maxsurge=5. Lo que se busca, es que el nuevo ReplicaSet tenga 10 Pods activos y eliminar todos los Pods del antiguo ReplicaSet, el siguiente paso es disminuir el número de Pods en el antiguo ReplicaSet. Para la desactivación Pods se debe tener en cuenta que el valor de MaxUnavailabel es del 10 %, es decir, que de los 10 Pods totales antes de la actualización, debe haber mínimo 9 Pods totales activos en proceso de actualización, por ello solo se desactivan 6 Pods del antiguo ReplicaSet, posteriormente se agregan 5 Pods al nuevo ReplicaSet, finalmente ya teniendo los 10 Pods en el nuevo ReplicaSet se pueden eliminar los Pods restantes del antiguo ReplicaSet.

Azul/Verde

Con la estrategia de actualización Azul/Verde, se crea un nuevo Despliegue con una nueva versión de la aplicación. En este caso, es app_html-v2. Una vez los Pods del nuevo Despliegue están listos, se cambia el tráfico de la versión azul a la nueva versión verde. Para lograr gestionar los flujos de tráfico, se utiliza un nuevo objeto de Kubernetes, llamado **Service**, el Service permite gestionar el flujo del tráfico por medio de etiquetas que tiene cada Pod. Cuando se define el objeto Servicio, los Pods son seleccionados basándose en un selector de etiquetas y versión, para la gestión de tráfico.

Universidad Militar Nueva Granada

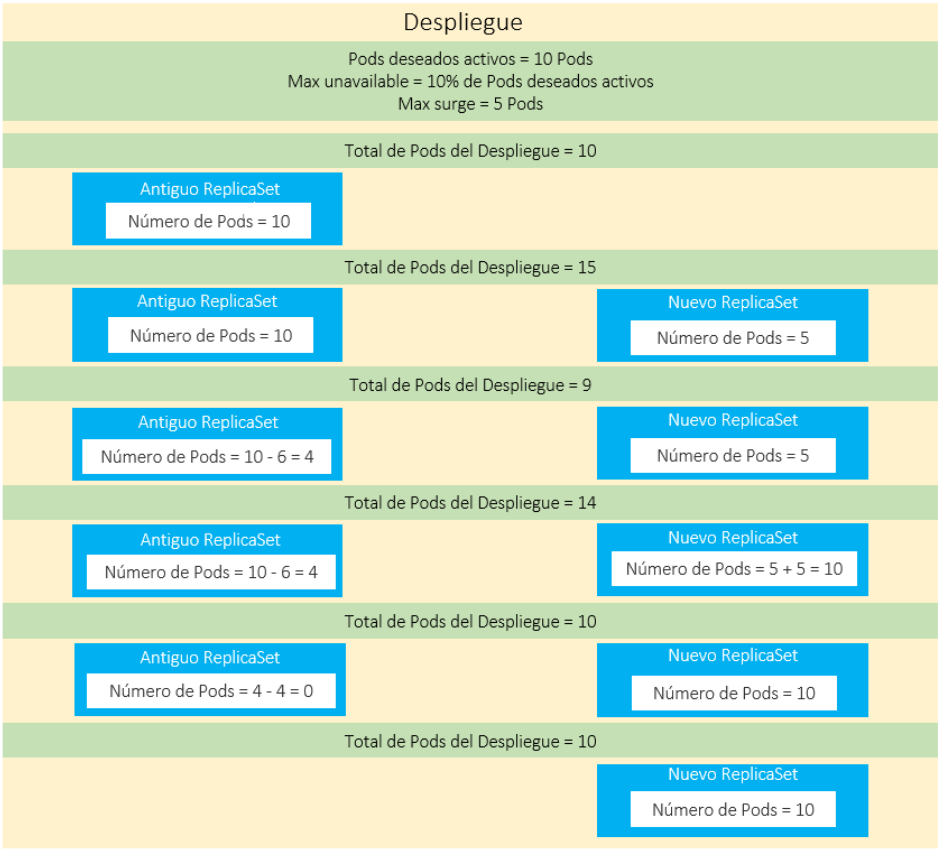

Figura 5.11: Ejemplo de la estrategia de Actualización continua.

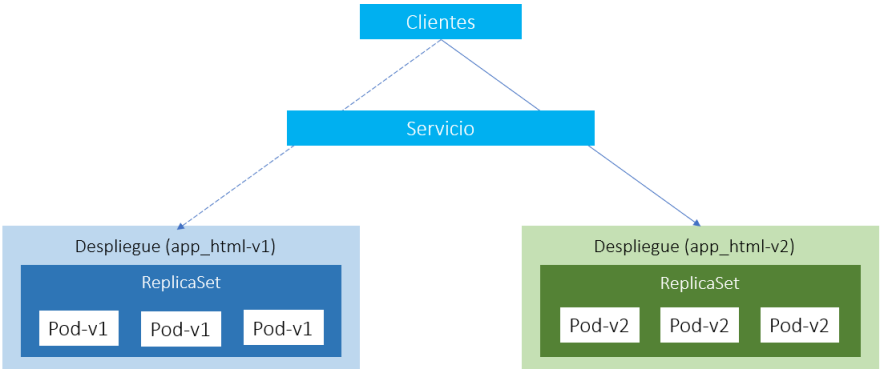

Figura 5.12: Estrategia de actualización Azul/Verde.

Como se observa en la figura 5.13, se crea una segunda versión del Despliegue con **apply**, pero para hacer el cambio del tráfico de la versión v1 a la v2, se aplica el comando de **patch service**, el cual modifica la versión dentro del service.yaml a la nueva versión que tendrá el 100 % del tráfico. El utilizar esta estrategia permite que la actualización sea instantánea, al igual que las versiones puedes ser probadas internamente sin afectar al usuario final de la aplicación, pero con la desventaja de que la utilización de los recursos se duplicara al tener varias versiones del Despliegue [19].

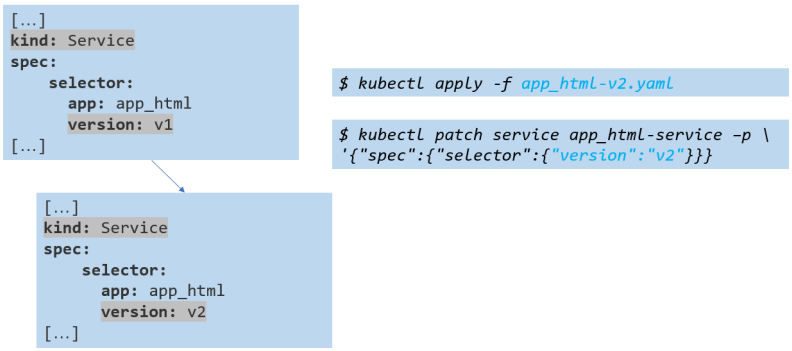

Figura 5.13: Service.yaml y comandos de aplicación de Azul/Verde.

[19] Yang, B., Sailer, A., & Mohindra, A. (2019, October). Survey and Evaluation of Blue-Green Deployment Techniques in Cloud Native Environments.

Canario

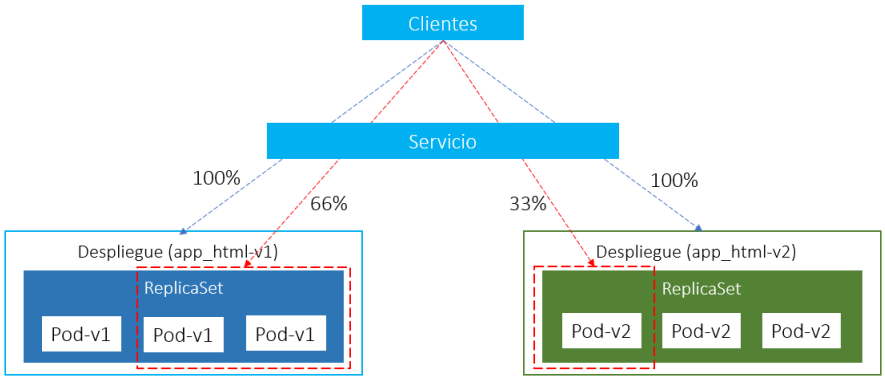

Figura 5.14: Estrategia de actualización Canario.

El método canario es otra estrategia de actualización basada en el método azul/verde, pero el tráfico se va desplazando gradualmente hacia la nueva versión. En la figura 5.14, el 100 % del tráfico de la aplicación se dirige inicialmente a app_html-v1. Cuando la estrategia canario comienza, el tráfico inicial en el Despliegue de la versión v1 disminuye hasta el 66 % y el tráfico del Despliegue de la versión v2 aumenta a un 33 %, dicho proceso se hace gradualmente hasta que el Despliegue de la versión v2 tenga el 100 % del tráfico.

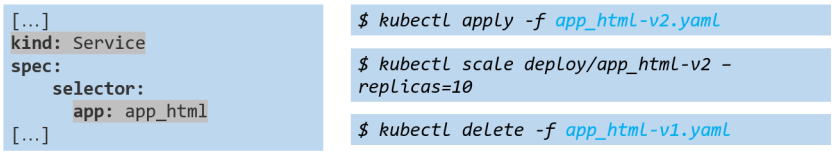

Figura 5.15: Archivo .YAML y comandos de aplicación de Canario.

El selector de **Servicio** se basa solo en la etiqueta de la aplicación y no en la etiqueta de la versión. El tráfico se envía a todos los Pods de la aplicación app_html, independientemente de

la etiqueta de la versión. La función del objeto Servicio de la figura 5.15, es dirigir el tráfico de los Pods de ambos Despliegues, inicialmente el Despliegue de la segunda versión escalará gradualmente su cantidad de Pods y el Despliegue de la primera versión disminuirá su cantidad de Pods, hasta el punto donde no queden Pods en el Despliegue de la primera versión, y todo el tráfico quede dirigido a los Pods de la segunda versión.

Recrear

En esta estrategia de actualización se eliminan todos los viejos Pods, antes de que se creen nuevos Pods. Este tipo de estrategia se utiliza en casos especiales donde se quiera hacer una ruptura limpia de los servicios que presta la aplicación, este tipo de estrategia se aplica directamente sobre el Despliegue.yaml, en el tipo de estrategia como se observa en la figura 5.16.

```
[...]
kind: deployment
spec:
    replicas: 10
    strategy:
        type: Recreate
[...]
```

Figura 5.16: Archivo .yaml de un Despliegue con Recrear.

¿Qué estrategia de actualización aplicar?

Cuando gestionamos y aplicamos actualizaciones sobre los contenedores que administramos con el sistema de Kubernetes, se busca tener presente que sucede con tres factores claves:

- Los servicios que prestamos con los contenedores, que se actualizaran.
- los recursos que utilizamos en la nube, donde están los contenedores que actualizaremos.
- los usuarios o el tráfico producido por ellos hacia los contenedores que se actualizaran.

	Actualización continua	Azul/Verde	Canario	Recrear
¿Qué sucede con los servicios durante la actualización?	No son interrumpidos	No son interrumpidos	No son interrumpidos	Son interrumpidos
¿Qué sucede con los recursos durante la actualización?	Los recursos aumentan, pero son configurables.	Los recursos se duplica	Los recursos se duplica	Los recursos se mantienen iguales
¿Qué sucede con los usuarios durante la actualización?	Se distribuyen en la versión antigua y la actualizada	Se mantienen en la versión antigua, solo hasta que la actualización finalice	Se distribuyen en la versión antigua y la actualizada	Sin servicio, no hay usuarios

Figura 5.17: Comparación de las estrategias de actualización para los Despliegues

En la figura 5.17 se puede observar que sucede con cada uno de los factores respecto a cada una de las actualizaciones ya antes explicadas. Con la intención de facilitar la comprensión de cuando se debe aplicar un tipo de estrategia de actualización, se crea el diagrama de flujo de la figura 5.18, que permite escoger que tipo de actualización se debe aplicar según los requerimientos de la aplicación, los requerimientos están relacionados con el servicio, los recursos y los usuarios.

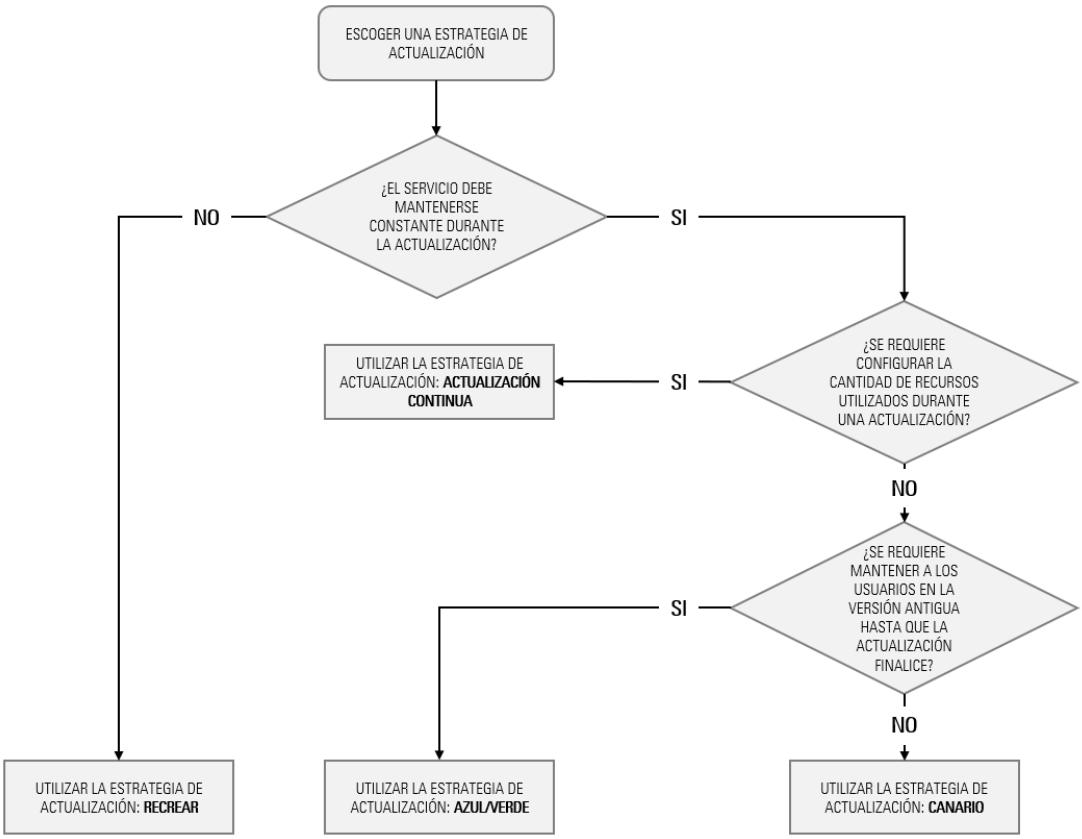

Figura 5.18: Diagrama de flujo para escoger el tipo de actualización más adecuado según los requerimientos de los servicios, recursos y usuarios

Retroceso de actualizaciones

En la mayoría de las estrategias de actualización, el Despliegue con la versión antigua es eliminada y en algunas ocasiones cuando se tiene una actualización ya implementada se desea volver a la versión anterior de dicha actualización. Así que, por medio de los comandos de la figura 5.19, se puede retroceder a las versiones anteriores de un Despliegue, si se especifica con **–to-revision** se volverá a la versión referida, y con **history** se puede inspeccionar el historial del Despliegue [20].

[20] kubernetes.io, Documentación (Deployment), 2020.

```
$ kubectl rollout undo deployment [Nombre_del_Deployment]

$ kubectl rollout undo deployment [Nombre_del_Deployment] --to-revision=2

$ kubectl rollout history deployment [Nombre_del_Deployment] --revision=2
```

Figura 5.19: Comandos de aplicación para retroceder o ver actualizaciones anteriores.

5.1.4 Ciclo de vida

Estado en proceso

Este estado indica que se está realizando una tarea sobre el Despliegue, algunas de las tareas pueden ser:

- Crear un nuevo ReplicaSet.
- Escalar un ReplicaSet.
- Reducir un ReplicaSet.

Estado completo

Este estado indica que cada una de las nuevas réplicas han sido actualizadas con última versión y están disponibles, y las réplicas de la versión anterior no están activas.

Estado fallido

Este estado indica que ocurrió un error en la creación de un nuevo ReplicaSet, y no pudo ser completado el Despliegue.

5.1.5 Acciones

`$ kubectl rollout pause deployment [Nombre_del_Deployment]`	Pausar
`$ kubectl rollout resume deployment [Nombre_del_Deployment]`	Subir cambios
`$ kubectl rollout status deployment [Nombre_del_Deployment]`	Monitorear
`$ kubectl delete deployment [Nombre_del_Deployment]`	Eliminar

Figura 5.20: Comandos de acción sobre Despliegue.

Cuando frecuentemente se hacen actualizaciones o se editan los Despliegue, esto desencadena un despliegue (rollout) automático [14]. Se vuelve complejo identificar la relación de que rollout pertenece a que Despliegue, así que existen diferentes acciones que se pueden realizar con los rollout, como se observa en la figura 5.20:

- Rollout pause: el Despliegue continuará funcionando, pero las nuevas actualizaciones del Despliegue no tendrán ningún efecto mientras el rollout está en pausa.
- Rollout resume: los cambios solo se implementarán cuando el rollout del Despliegue se reanude.
- Rollout status: funciona para monitorear el estado de las actualizaciones del Despliegue.
- El último comando de la figura 5.20 simplemente es para borrar el Despliegue. Cuando se borra un Despliegue, Kubernetes borra los Pods y libera o elimina los recursos gestionados por los mismos.

Universidad Militar Nueva Granada

5.2 Administración de Pods temporales

Figura 5.21: Estructura de un Despliegue vs. la estructura de un Trabajo.

El **Trabajo** es otro de los objetos de Kubernetes, este a diferencia del Despliegue no mantiene un estado abierto indefinidamente, así que la función del Trabajo es llegar a un estado y una vez llega a ese estado, el Pod se finaliza. Kubernetes se encargara de mantener un Pod activo hasta que finalice la tarea y si es el caso creara más Pods hasta que la tarea sea finalizada [21].

[21]Galantino, S. (2020). Enabling Job-aware scheduling on Kubernetes clusters (Doctoral dissertation, Politecnico di Torino).

Kubectl con Trabajo

Como cualquier otro objeto de Kubernetes, se pueden aplicar ciertos comandos en la consola de la plataforma de Google Cloud para inspeccionar, visualizar estados o escalar el Trabajo [22], un ejemplo en la figura 5.22.

```
$ kubectl describe job [Nombre_del_JOB]
```

```
$ kubectl get pod -l [Nombre_del_JOB=app_html-job]
```

```
$ kubectl scale job [Nombre_del_JOB]
--replicas [Numero_de_replicas]
```

Figura 5.22: Comandos Kubectl que se pueden aplicar a un Trabajo.

[22] Google Cloud "Kubernetes Engine", Clústeres (Guias, Ejecutar un trabajo), 2020.

Eliminación

Si se elimina un Trabajo todos los Pods asociados son eliminados [22]. En la figura 5.23, los dos primeros comando eliminarán el Trabajo y los Pods, pero si no se quiere eliminar los Pods asociados se debe utilizar el último comando.

```
$ kubectl delete -f [Archivo_JOB]
```

```
$ kubectl delete job [Nombre_del_JOB]
```

```
$ kubectl delete job [Nombre_del_JOB]
--cascade false
```

Figura 5.23: Comandos Kubectl para eliminar un Trabajo.

5.2.1 Trabajos no paralelos

Un Trabajo no paralelo, es un Trabajo por defecto que crea un único Pod para iniciar la tarea, si el Pod falla, se reiniciara asegurándose que la tarea tenga otra oportunidad de obtener un estado exitoso [20]. Por ejemplo, en la figura 5.24 un solo Pod es activado por el Trabajo y dicho Pod deberá completar la tarea tres veces, debido a la asignación de **completions**.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: app_html-job
spec:
  completions: 3
  template:
    spec:
[...]
```

Figura 5.24: Archivo YAML de un Trabajo no paralelo.

5.2.2 Trabajos Paralelos

Los Trabajos paralelos activan varios Pods para ejecutar una o varias tareas, esto depende del tipo de paralelismo [23]. Los tipos de paralelismo son creados por la asignación de dos especificaciones, en la figura 5.25 se observan dos archivos YAML con una variación en sus especificaciones, en el lado izquierdo se combinan las especificaciones de **completions** y **parallelism**, creando un Trabajo con un contador de tareas. Del lado derecho se omite la especificación de **completions**, creando un Trabajo con Pods en cola.

R [23] kubernetes.io, Documentación (Jobs), 2020.

```
apiVersion: batch/v1
kind: Job
metadata:
  name: app_html-job
spec:
  completions: 3
  parallelism: 2
  template:
    spec:
[...]
```

```
apiVersion: batch/v1
kind: Job
metadata:
  name: app_html-job
spec:
  parallelism: 3
  template:
    spec:
[...]
```

Figura 5.25: Archivos YAML de los Tipos de Trabajos paralelos.

Trabajo con contador de tareas

Para crear este tipo de Trabajo se debe asignar dos especificaciones. **Completions**, es el número de tareas a realizar, y **parallelism**, es el número de Pods que pueden trabajar en paralelo [20]. Si aplicamos el código del lado izquierdo de la figura 5.25, el resultado del proceso de despliegue de Pods y ejecución de tareas de dicho código se puede observar en la figura 5.26. Con los valores de 3 para **completions** y 2 para **parallelism**, el proceso inicia con dos Pods

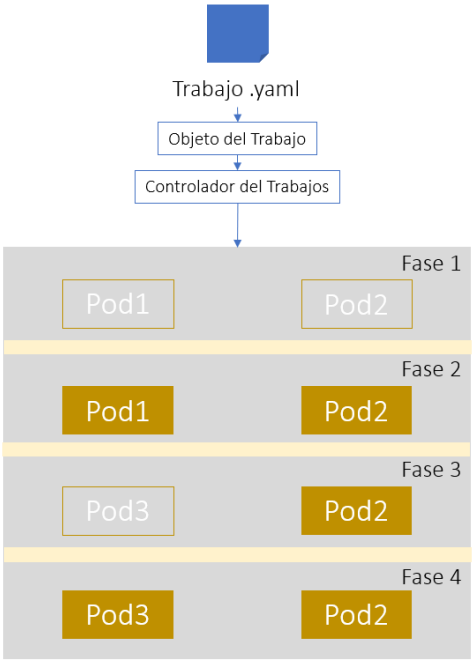

Figura 5.26: Ejemplo de los procesos de un Trabajo con contador de tareas.

trabajando en paralelo en la fase 1. En la fase 2, los dos Pods han finalizado sus tareas, pero como el Trabajo tiene asignado 3 tareas, faltaría un Pod para finalizar la última tarea. En la fase 3, se crea el Pod3. En la fase 4, solo se espera el que el Pod3 finalice su tarea para finalizar el Trabajo.

Trabajo con Pods en cola

A este tipo de Trabajo solo se le asigna una especificación, como se observa en el lado derecho de la figura 5.25. El Trabajo despliega una cantidad de Pods en paralelo para ejecutar una misma tarea, si alguno de los Pods finaliza la tarea los demás Pods son finalizados. En la

Universidad Militar Nueva Granada

figura 5.27, se muestra el proceso de un Trabajo cuando se asigna un valor de 3 a **parallelism**, en donde el Trabajo desplegara 3 Pods para que ejecuten una misma tarea, en la fase 2 un solo Pod finalizara la tarea, los dos Pods restantes se finalizaran automáticamente cuando detecten la tarea finalizada por el Pod1. Si el Pod1 falla, los demás Pods estarán en proceso de realización de la tarea, así que no será necesario reiniciar un nuevo Pod.

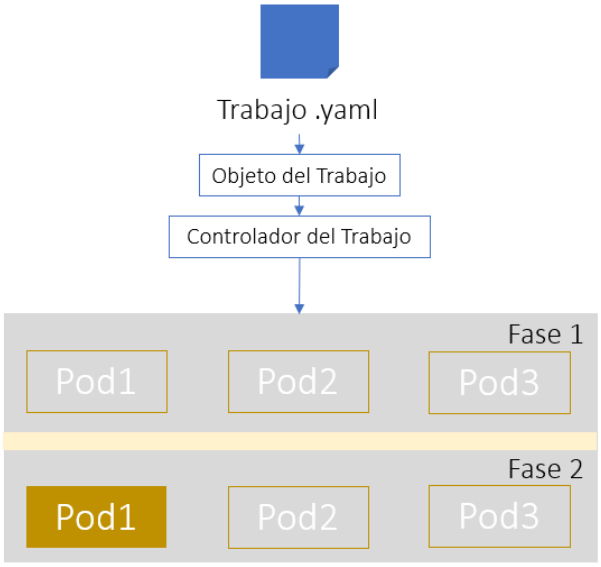

Figura 5.27: Ejemplo de los procesos de un Trabajo con Pods en cola.

5.2.3 Crono-Trabajos

Los trabajos pueden facilitar la implementación de contenedores en los cuales el objetivo sea realizar una tarea que deba ser realizada y finalizada, pero con la condición que dicha tarea deba ser ejecutada en el instante que se implementa el trabajo [24], con el crono-trabajo se puede dar una fecha exacta de ejecución del trabajo que a su vez administra y ejecuta los contenedores como se ejemplifica en la figura 5.28.

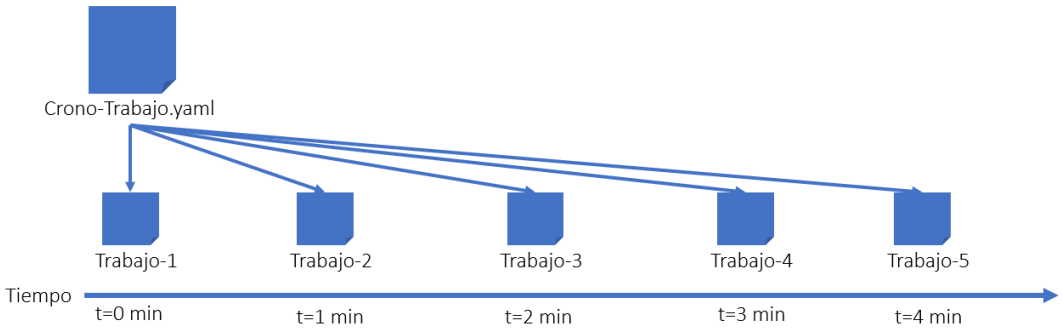

Figura 5.28: Ejecución de un crono-trabajo.

[24] Google Cloud "Google Kubernetes Engine (GKE)", Clústeres (Guias, Ejecutar un CronJob), 2020.

En la figura 5.29 se puede observar que en las especificaciones del objeto se puede declarar el minuto, hora, día del mes, mes y día de la semana que el crono-trabajo será ejecutado, que para el ejemplo de la figura se ejecutará cada 1 minuto. Este tipo de objetos nos permiten implementar contenedores que preste servicios que sea ejecutados a petición del usuario en el momento exacto que el usuario desee.

```
apiVersion: batch/v1
kind: CronJob
metadata:
  name: app_html-job
spec:
  schedule: "*/1 * * * *"
  jobtemplate:
    spec:
      template:
        spec:
[...]
```

Figura 5.29: Archivo.yaml de un crono-trabajo que se ejecuta cada 1 minuto.

5.3 Comparación de objetos de Kubernetes

Cuando se quiere implementar y administrar contenedores en Kubernetes se debe conocer cuáles son los objetos que controlaran y ejecutaran a los contenedores. Los objetos más básicos dentro del sistema de Kubernetes son los que se pueden observar en la primera columna de la figura 5.30. El objeto más básico y más cercano al contenedor es el Pod, este a su vez es administrado por otros objetos para agregar cualidades de la nube. en la figura 5.30 se puede observar cuál es el objeto administrador, el objeto administrado y la funcionalidad del objeto administrador. Se debe tener en claro que una aplicación puede ser implementada con múltiples contenedores, por ejemplo: una página web debe estar activa constantemente para no denegar el servicio a ninguno de sus clientes en internet, por ello lo más recomendable seria utilizar un despliegue, el autoescalado del despliegue permitiría aprovechar al máximo los recursos que consume el contenedor de la página web, permitiendo que si solo un cliente está utilizando la página web solo se utilicen los recursos web para un solo cliente. Cuando se utiliza un trabajo para administrar un contenedor, el contenedor solo se ejecutaría por un periodo de tiempo, así que sería recomendable utilizarlo cuando el contenedor ejecuta aplicaciones que realizan una tarea que debe finalizar. El crono-trabajo es solo agregarle a un trabajo cuando se desea realizar la ejecución del contenedor. Como se observa en la figura 5.30 solo un objeto administra directamente al contenedor, los demás objetos administrar otros objetos diferentes al contenedor, lo que permite asumir que estos otros objetos determinan la funcionalidad del contenedor.

Universidad Militar Nueva Granada

Tipo de objeto		Sub-objeto que es administrado	Función del objeto
Pod		Contenedor	Administrar y ejecutar los contenedores dentro de él, como solo un objeto.
Despliegue		Pods	Administrar y ejecutar una cantidad n de Pods activos, durante un tiempo infinito.
Trabajo	No paralelo	Pods	Administrar y ejecutar un Pod para que realicen n tareas, una vez finaliza la tarea se elimina el Pod.
	Paralelo	Pods	Administrar y ejecutar n Pods para que realicen n tareas, una vez finalizan las tareas se eliminan los Pods.
Crono-trabajo		Trabajo	Desplegar y Ejecutar un trabajo cada cierto tiempo.

Figura 5.30: Objetos básicos de Kubernetes y sus funcionalidades.

Una forma más sencilla de entender la administración de los contenedores, es observando la figura 5.31 donde el objeto encapsulado es el objeto administrado por el objeto que lo encapsula, es decir; en la primera sección de la figura 5.31 el maestro es una máquina virtual que administra a los nodos, en las secciones 2, 3 y 4 los nodos darán la capacidad de cómputo y conectividad a los objetos, los objetos de los nodos administraran los objetos internos y finalmente el Pod será el objeto que administre la conectividad y ejecución del contenedor.

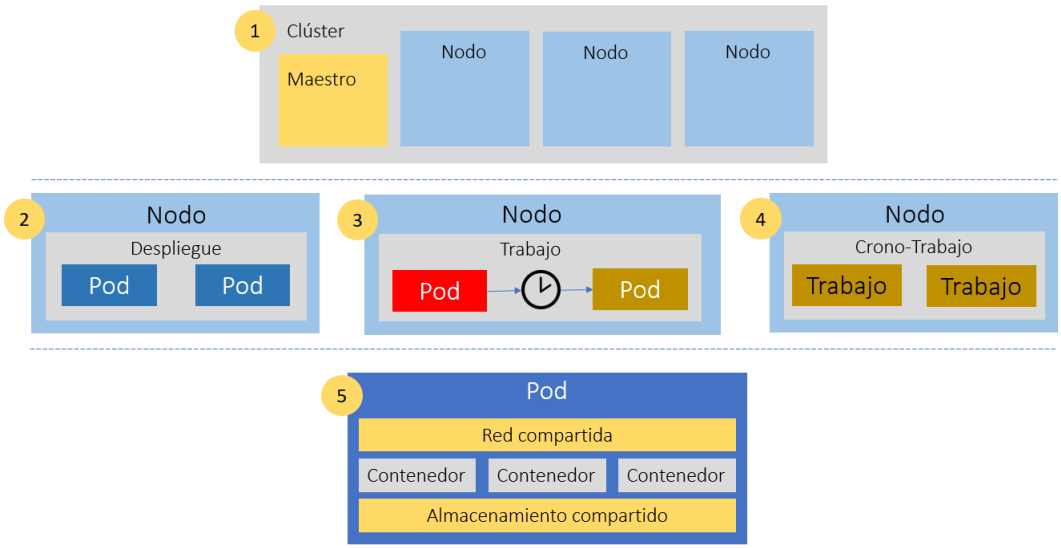

Figura 5.31: Dominio de administración y ejecución de los objetos en Kubernetes.

Debido a que el Pod es el objeto gestionado y dominado por los demás objetos, es esencial conocer los requerimientos de nuestro Pod para lograr seleccionar el mejor objeto que logre satisfacer las necesidades del Pod; debido a ello, se crea el diagrama de flujo que se observa en la figura 5.32, con la intención de que se logre escoger el objeto exacto dadas las características de los objetos que administran al Pod.

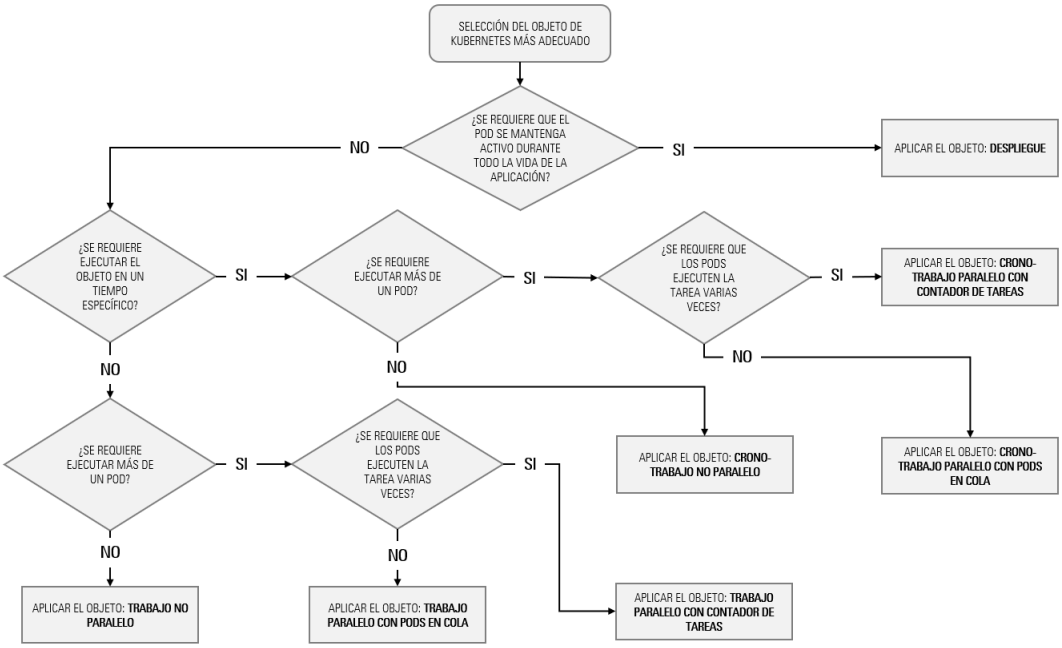

Figura 5.32: Diagrama de flujo para seleccionar el objeto administrador más adecuado a los requerimientos de un Pod.

5.4 Gestión del clúster

La gestión de los objetos permite administrar la parte lógica de las aplicaciones compuestas por los contenedores, dando lugar a que los contenedores se ejecuten dentro del sistema de Kubernetes. La gestión del clúster permite administrar los recursos que utilizaran los objetos, por ello es esencial sincronizar los recursos con los objetos. La habilitación del escalado de nodos permite dicha sincronización, por otra parte la red del clúster permitirá comprender la comunicación entre objetos y como se expone finalmente un objeto a internet.

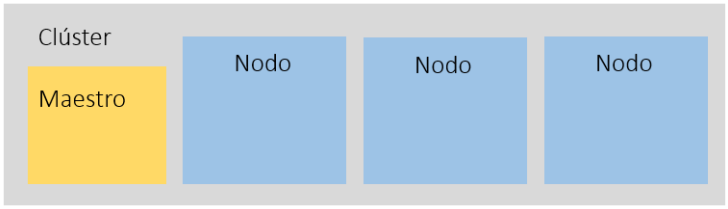

Figura 5.33: Composición predeterminada de un clúster de Kubernetes en la nube.

En la figura 5.33 se observa dos tipos de recursos, el maestro y el nodo. El maestro es un recurso de administración que gestiona los procesos internos de interacción entre el usuario, nodos y los objetos. Por el contrario el nodo es un recurso de ejecución para los objetos; así que, si queremos aumentar la cantidad de recursos para una aplicación debemos ejecutar un comando para aumentar el número de nodos del clúster. En la figura 5.34 se puede observar cuál debe ser el comando aplicado en la consola de la plataforma de Google Cloud para escalar el clúster a 5 nodos.

```
$ gcloud container clusters resize
project --node-pool default-pool \
--size 5
```

Figura 5.34: Comando para ejecutar en escalado del clúster, modificando el número de nodos.

5.4.1 Autoescalado de recursos

Él autoescalado permitirá escalar los nodos cuando surja un aumento de los objetos en el clúster, esto evita que se tenga que aplicar un comando para escalar nodos cada que se escalan los objetos. Una vez que se habilita el autoescalado este tiene inmerso un sistema de desescalado que permite la desactivación de nodos que tiene cierto umbral de utilización [25].

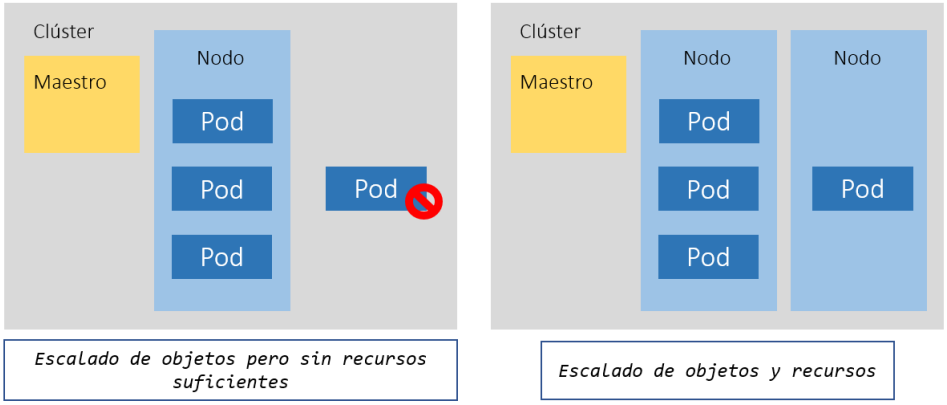

Figura 5.35: Dinámica de sincronización de escalado entre recursos y objetos, en el clúster de Kubernetes

[25] Wang, M., Zhang, D., & Wu, B. (2020, June). A Cluster Autoscaler Based on Multiple Node Types in Kubernetes. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 1, pp. 575-579). IEEE.

La sincronización entre recursos y objetos se establece cuando se habilita el autoescalado del clúster permitiendo que un nuevo Pod pueda tener los recursos suficientes para que se ejecute el contenedor. En la figura 5.35 se puede observar que cuando se escala un despliegue y el nodo no tiene recursos suficientes para el nuevo Pod, asi que el maestro tiene dos opciones:

1. No desplegar el Pod debido a que no hay más recursos y se tiene deshabilitado él autoescalado del clúster.
2. Autoescalar el clúster creando otro nodo y ubicando el nuevo Pod en dicho nodo, debido a que está habilitado el autoescalado del clúster.

Cuando se trata del desescalado, el maestro examina por 10 minutos los nodos que no sobrepasen la utilización de recursos en un 50 %, durante los 10 minutos el maestro reubica los Pods para optimizar la utilización de recursos y posterior a los 10 minutos son eliminados los nodos que no tengan objetos asignados. Si durante la operación de una aplicación desplegada en Kubernetes se tiene pensado el escalado de los objetos, es fundamental habilitar dicha característica; lo que evitará futuros conflictos en despliegue de objetos y permitirá un escalado optimizado a la aplicación.

Universidad Militar Nueva Granada

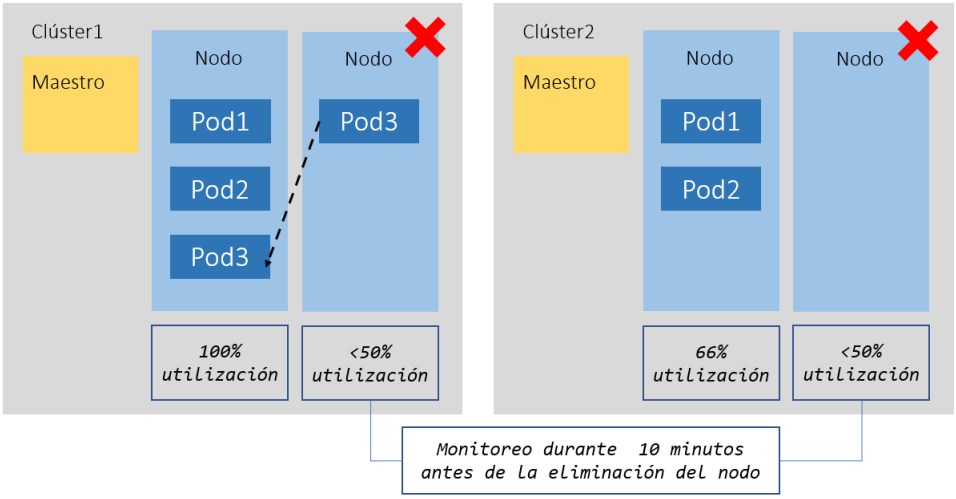

Figura 5.36: Desescalado del clúster

5.4.2 Tipos de habilitación de autoescalado

Según la disposición de la consola de la plataforma de Google Cloud, se pueden ejecutar 4 diferentes comandos relacionados a diferentes acciones con el autoescalado del clúster. Así que utilizando el diagrama de flujo de la figura 5.41 se establece cuál debe ser el comando utilizado según las decisiones en el diagrama de flujo.

```
$ gcloud container clusters create
[Nombre_clúster] --num-nodes 30 \
--enable-autoscaling --min-nodes 15
--max-nodes 50 [--zone Zona_computo]
```

Figura 5.37: Comando 1, para crear un nuevo clúster con autoescalado.

```
$ gcloud container node-pools create
[Nombre_pool] --cluster [Nombre_clúster]
--enable-autoscaling --min-nodes 15
--max-nodes 50 [--zone Zona_computo]
```

Figura 5.38: Comando 2, para crear un nuevo pool de nodos con autoescalado.

```
$ gcloud container clusters update
[Nombre_clúster] --enable-autoscaling \
--min-nodes 1 --max-nodes 10 --zone
[Zona_computo] --node-pool [Nombre_pool]
```

Figura 5.39: Comando 3, para habilitar el autoescalado en un pool de nodos, anteriormente creado.

```
$ gcloud container clusters update
[Nombre_clúster] --no-enable-autoscaling \
--node-pool [Nombre_pool] [--zone
[Zona_computo] --project [ID_projecto]]
```

Figura 5.40: Comando 4, para deshabilitar el autoescalado en un pool de nodos.

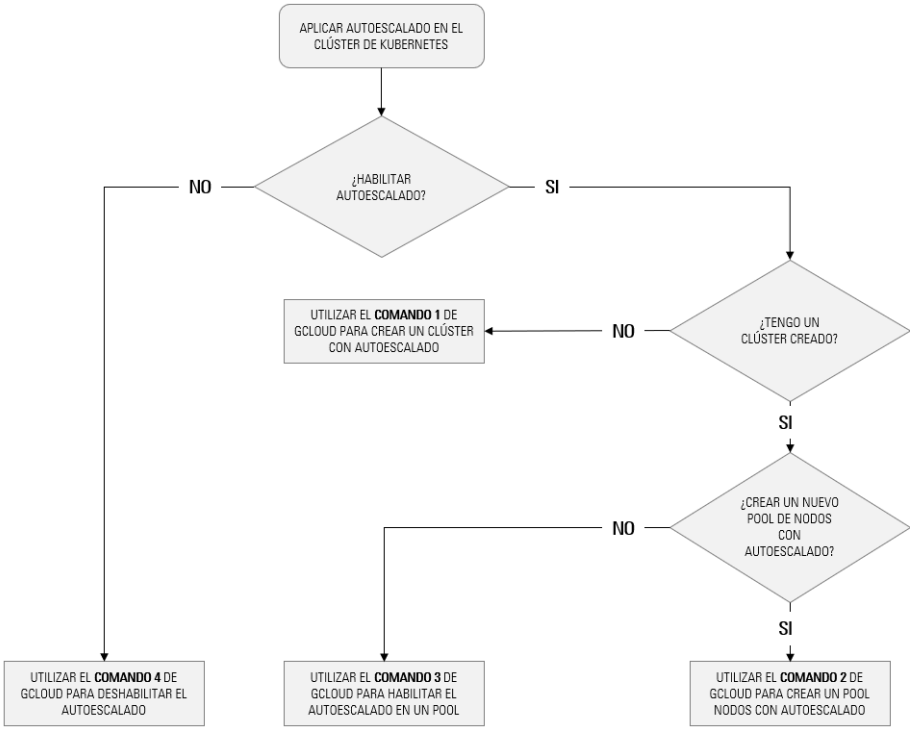

Figura 5.41: Diagrama de flujo para seleccionar el comando que permita habilitar o deshabilitar el autoescalado en el clúster.

5.4.3 Red del Clúster

Para lograr que un objeto cumpla con su función como un fragmento de una aplicación, el objeto debe ser expuesto en la red del clúster. La exposición de un objeto en la red se hace mediante el uso del comando `kubectl` que permite asignar un puerto específico de exposición [26].

[26] Liffredo, D. (2020). Analysis and Benchmarking of Kubernetes Networking (Doctoral dissertation, Politecnico di Torino).

Una vez que se expone un objeto este se conecta a una red como se puede observar en la figura 5.42. Cada objeto es conectado como un host de la red y cada contenedor tiene una conexión compartida por medio de la IP 127.0.0.1 dentro del nodo. La conexión entre los dos Pods de la figura 5.42 se establece por medio de interfaces virtuales que permite la conexión de objetos en un mismo nodo, si la conexión es entre objetos de diferentes nodos, GCP establece una nube virtual privada que otorga un pool de IPs a cada nodo que se relaciona con el número de identificación (NIC) de cada nodo. Conociendo las IPs establecidas para un determinado NIC,

la nube virtual privada logra hacer el enrutamiento, en la figura 5.43 se pueden observar los dos tipos de conexiones que existen un la red del clúster, la conexión de Pod-a-Pod establecida por los NICs y la exposición del Pod a internet por medio una asignación de un puerto específico y una IP pública. Si se deseara modificar el tipo de red o interacción de los objetos y recursos en la red, existe el objeto **Servicio**, este objeto no es analizado a profundidad debido a la cantidad de teoría que debe ser abarcada, así que solo es relacionado en la sección de actualizaciones de la administración de múltiples objetos del presente trabajo, por otro lado dicho objeto será la base de investigación para el desarrollo de próximos trabajos que tenga como base el presente trabajo.

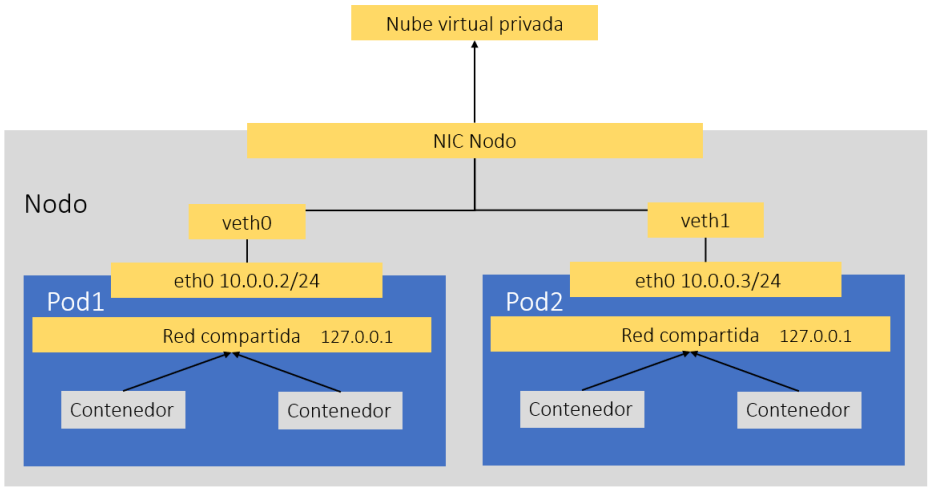

Figura 5.42: Topología de red del nodo.

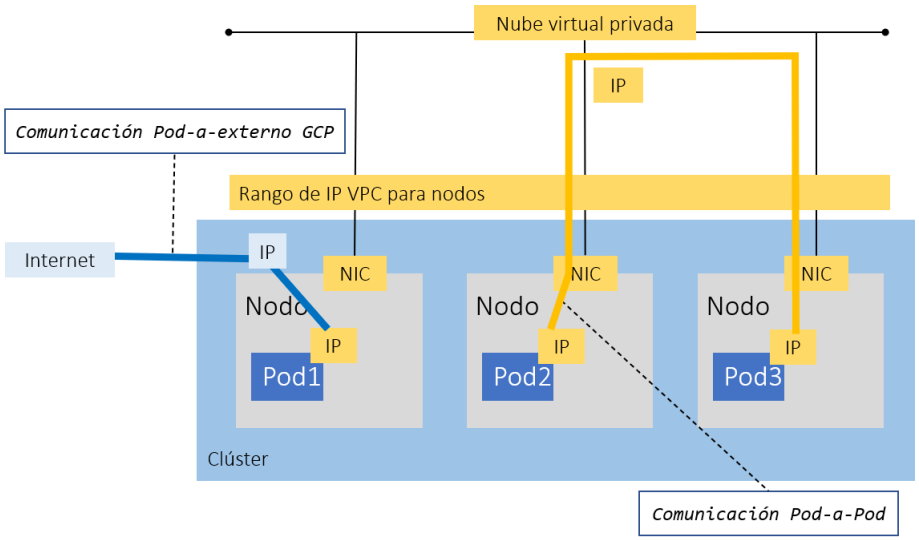

Figura 5.43: Diagrama de los tipos de comunicaciones en un clúster.

Universidad Militar Nueva Granada

Creación de un Clúster e implementación de un Pod
Implementar un Pod a partir de un archivo Yaml
Implementación de un Despliegue y exposición por medio de un Servicio
Implementación de trabajos y crono-trabajos
Implementación de un Despliegue con autoescalado

6. Implementaciones en Kubernetes Engine

Kubernetes Engine es el servicio de la plataforma de la nube de Google sobre la cual se realizó la exploración del servicio por medio del despliegue de diferentes implementaciones, cada una de las implementaciones son la aplicación de la teoría de los capítulos anteriores.

La primera implementación permite desplegar una página web básica por medio de un contenedor que es gestionado por la infraestructura de Kubernetes. Mostrando los comandos necesarios para modificar el clúster, como se habilitan los puertos de exposición del servicio web e interacción con los objetos de Kubernetes.

La segunda implementación permite desplegar una página web básica en la red interna que tiene el clúster una vez que es implementado. Mostrando los comandos de aplicación del archivo Yaml de un objeto Kubernetes, la interacción con la consola del objeto y la exposición local de la página web.

La tercera implementación permite desplegar un servicio web compuesto por varios objetos de Kubernetes, que son gestionados y desplegados al mismo tiempo. Mostrando los comandos para aumentar el número de objetos y la aplicación de un objeto dedicado a la exposición del servicio a internet.

La cuarta implementación permite desplegar y gestionar contenedores que realizan tareas temporales y periódicas. El primero gestiona un contenedor que realiza un cálculo matemático finito y el segundo imprime periódicamente una cadena de caracteres. Mostrando los comandos para observar las tareas realizadas por los objetos de Kubernetes.

La quinta implementación permite desplegar dos aplicaciones basadas en contenedores en un mismo clúster. La primera aplicación tiene contenedores de servicio web y la segunda tiene contenedores que generan peticiones a un URL específico. Mostrando los comandos para conectar los aplicativos y simular la actividad de una aplicación cuando se expone a internet.

6.1 Creación de un Clúster e implementación de un Pod

En la primera implementación se ejecutó un `Despliegue` que administrara un `Pod` con un contenedor de un servicio Web `nginx`, donde se programó una página web con un `hello word`

que es expuesto a internet. En la figura 6.1 se puede observar la contención del `Pod` por medio del `Despliegue` como también la adición de un nuevo nodo dentro del clúster en la topología de la implementación número 1.

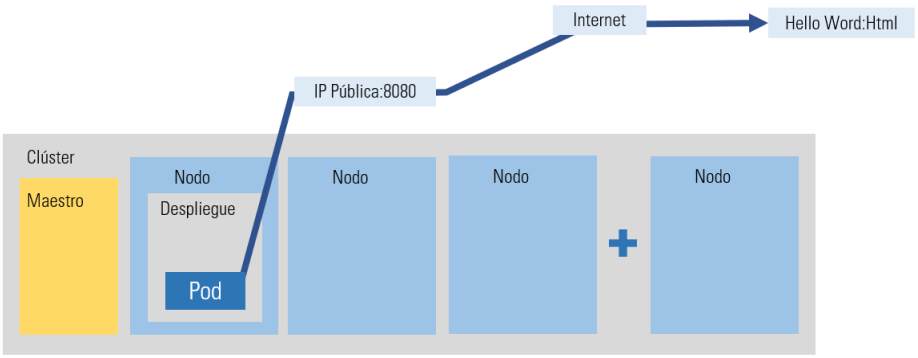

Figura 6.1: Topología de un clúster y un despliegue con un solo Pod expuesto a internet.

Se inició con la ejecución de las 4 siguientes líneas de comando en Cloud Shell, las dos primeras permiten asociar un valor a una variable y las dos últimas permiten observar el resultado de la variable, como se puede observar en la figura 6.2. Estas variables están relacionadas con la ubicación y el nombre del clúster.

```
1 export mi_zona=us-east1-b
2 export mi_cluster=standard-cluster-1
3 echo $mi_zona
4 echo $mi_cluster
```

```
u1401119@cloudshell:~ (second-pier-307520)$ echo $mi_zona
us-east1-b
u1401119@cloudshell:~ (second-pier-307520)$ echo $mi_cluster
standard-cluster-1
u1401119@cloudshell:~ (second-pier-307520)$
```

Figura 6.2: Variables de zona y nombre del clúster impresas en Cloud Shell.

Luego se creó él `Clúster` con las características de la topología que en este caso fueron de 3 nodos inicialmente, para ello se aplicó el siguiente comando:

```
1 gcloud container clusters create $mi_cluster --num-nodes 3 --zone $mi_zona --enable-ip-
    alias
```

En la figura 6.3 se puede observar la creación del `Clúster`, este proceso logra ser el más demorado de toda la implementación.

```
Creating cluster standard-cluster-1 in us-east1-b... Cluster is being health-checked (master is healthy)...done.
Created [https://container.googleapis.com/v1/projects/second-pier-307520/zones/us-east1-b/clusters/standard-cluster-1].
To inspect the contents of your cluster, go to: https://console.cloud.google.com/kubernetes/workload_/gcloud/us-east1-b/s
kubeconfig entry generated for standard-cluster-1.
NAME              LOCATION    MASTER_VERSION    MASTER_IP     MACHINE_TYPE  NODE_VERSION      NUM_NODES  STATUS
standard-cluster-1  us-east1-b  1.18.16-gke.2100  34.75.13.18  e2-medium     1.18.16-gke.2100  3          RUNNING
u1401119@cloudshell:~ (second-pier-307520)$
```

Figura 6.3: Creación del clúster.

Luego se habilitaron las credenciales al `Clúster` con el siguiente comando:

```
gcloud container clusters get-credentials $mi_cluster --zone $mi_zona
```

El siguiente comando permito hacer el escalado del `Clúster` adicionando un nuevo nodo:

```
gcloud container clusters resize $mi_cluster --zone $mi_zona --num-nodes=4
```

Luego se ejecutó el siguiente comando para observar la capacidad de cómputo y la cantidad de nodos que tiene el `Clúster`:

```
kubectl top nodes
```

En la figura 6.4 se observan los 4 nodos del `Clúster` creado, se debe tener en cuenta que él `Nodo Maestro` no se puede observar por medio de la ejecución del anterior comando.

```
u1401119@cloudshell:~ (second-pier-307520)$ kubectl top nodes
W0503 01:52:33.743673    1238 top_node.go:119] Using json format to get metrics. Next release will
NAME                                                CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
gke-standard-cluster-1-default-pool-f69d9932-2k2k   77m          8%     595Mi           21%
gke-standard-cluster-1-default-pool-f69d9932-6cvw   75m          7%     466Mi           16%
gke-standard-cluster-1-default-pool-f69d9932-g975   98m          10%    466Mi           16%
gke-standard-cluster-1-default-pool-f69d9932-mmsj   56m          5%     426Mi           15%
u1401119@cloudshell:~ (second-pier-307520)$
```

Figura 6.4: Visualización de nodos del clúster creado.

Una vez configurados los recursos de cómputo de la topología, se procedió a crear él `Despliegue` con la primera línea del siguiente comando, con la segunda línea se pudo observar las características básicas del `Pod` que se puede observar en la figura 6.5, debido a que el nombre del `Pod` es generado por el mismo sistema Kubernetes se guardó el nombre del `Pod` en una variable más sencilla para futuras acciones:

```
kubectl create deployment --image nginx nginx-1
kubectl get pods
export mi_nginx_pod=[nombre_del_pod]
```

```
u1401119@cloudshell:~ (second-pier-307520)$ kubectl create deployment --image nginx nginx-1
deployment.apps/nginx-1 created
u1401119@cloudshell:~ (second-pier-307520)$ kubectl get pods
NAME                       READY   STATUS              RESTARTS   AGE
nginx-1-7c87b4c649-ckhst   0/1     ContainerCreating   0          5s
u1401119@cloudshell:~ (second-pier-307520)$
```

Figura 6.5: Creación del Despliegue y estado del Pod

Luego se creó un archivo HTML con el siguiente comando, para crear la página web:

```
nano ~/test.html
```

Una vez creado el archivo HTML, se guardaron las siguientes tres líneas de comando dentro del archivo `test.html`:

```
<html><header><title>this is title</title></header>
<body> Hello World </body>
</html>
```

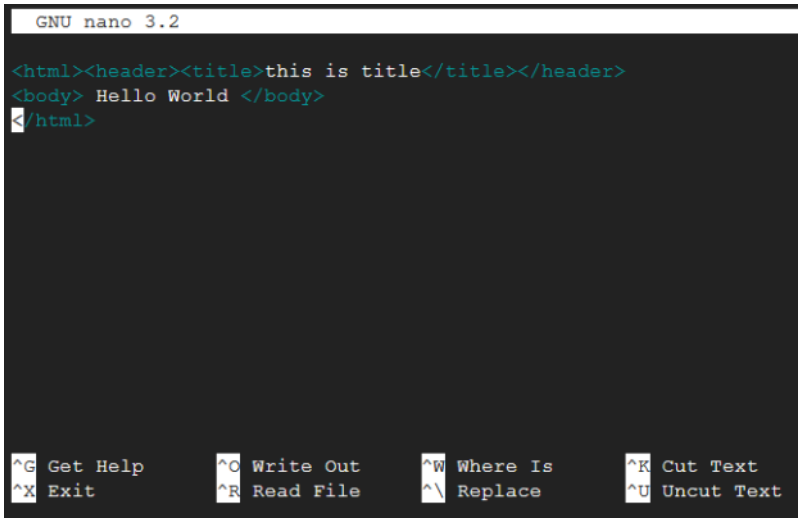

Figura 6.6: Código fuente de la página web en test.html.

Luego se copió el archivo `test.html` dentro del `Pod`, con el siguiente comando:

```
kubectl cp ~/test.html $mi_nginx_pod:/usr/share/nginx/html/test.html
```

Para que la página web creada pueda ser visualizada por un cliente con acceso a internet, se ejecutó el siguiente comando que permite exponer el `Pod` a internet por medio de una IP pública gracias a `LoadBalancer` y por el puerto 80:

```
kubectl expose pod $mi_nginx_pod --port 80 --type LoadBalancer
```

Luego se ejecutó el siguiente comando para visualizar las características de conectividad que tiene él `Clúster`:

```
kubectl get services
```

En la figura 6.7 se puede observar dos tipos de servicios, el primero referente a la conectividad del `Clúster`, que permite la conexión con otros servicios de la nube de Google. El segundo servicio permite la exposición del `Pod` a internet por medio de la `EXTERNAL_IP`.

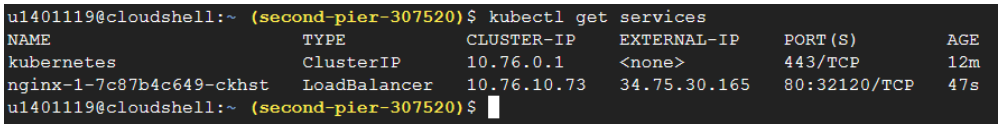

Figura 6.7: Servicios activos del clúster.

Finalmente la implementación fue desplegada con éxito y para comprobarlo se ejecutó el siguiente comando, que permite utilizar Cloud Shell como un buscador web:

```
curl http://[EXTERNAL_IP]/test.html
```

En las figuras 6.8 y 6.9 se puede observar el servicio activo de la página web que fue implementada por medio del sistema Kubernetes, con un `Despliegue` y un `Pod`.

```
u1401119@cloudshell:~ (second-pier-307520)$ curl http://34.75.30.165/test.html
<html><header><title>this is title</title></header>
<body> Hello World </body>
</html>
u1401119@cloudshell:~ (second-pier-307520)$
```

Figura 6.8: Visualización de código fuente de página web activa desde Cloud Shell.

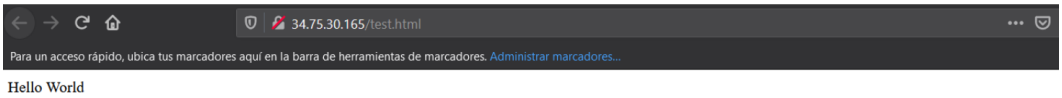

Figura 6.9: Visualización de página web desde un host conectado a internet.

Debido a que se implementó un `Despliegue` para administrar él `Pod`, se pueden ejecutar acciones de actualización o escalado sobre él `Pod` como se mencionó en el capítulo número 5 y sección 5.1. Las actualizaciones permitirán desplegar nuevas versiones del código de la página web y el escalado del `Pod` permitirá aumentar el número de clientes o peticiones que puede soportar la página web.

6.2 Implementar un Pod a partir de un archivo Yaml

Como se observa en la figura 5.2, en esta segunda implementación se desplegó el mismo tipo de servicios web `nginx`, pero con una exposición interna del `Pod` a la red del `clúster`, así que ningún usuario de internet podrá acceder el servicio, esto permite desarrollar aplicativos en redes privadas dentro del clúster o también realizar pruebas de testeo de páginas web antes de ser expuestas a los clientes. El desplegué del `Pod` se realizó por medio de un archivo `Yaml`, alojado y descargado de un repositorio público.

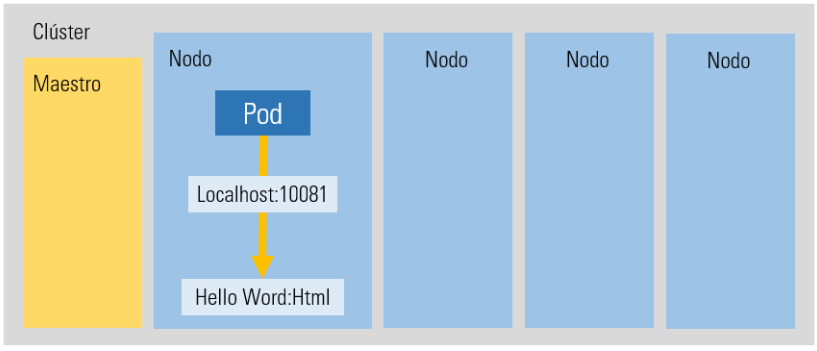

Figura 6.10: Topología de un clúster con un Pod expuesto a la red interna del clúster.

Para esta implementación se utilizó el mismo `Clúster` de la implementación anterior, si se desea utilizar un `Clúster` independiente se pueden utilizar los comandos utilizados anteriormente, se debe tener en cuenta que el nuevo `Clúster` debe tener un nombre diferente. Se inició con la descarga del repositorio de laboratorios que otorgar Google Cloud al público, para ello se utiliza el siguiente comando:

```
1 git clone https://github.com/GoogleCloudPlatform/training-data-analyst
```

Luego se ejecutó los siguientes comandos para ubicar el archivo `Yaml`, aplicarlo para crear el `Pod` y observarlo:

```
cd training-data-analyst/courses/ak8s/v1.1/GKE_Shell/
kubectl apply -f ./new-nginx-pod.yaml
kubectl get pods
```

En la figura 6.11 se puede observar la creación del `Pod` debido al resultado que otorga Cloud Shell con `pod/new-nginx`, en la misma figura podemos observar el `Pod` creado para esta implementación como para la anterior.

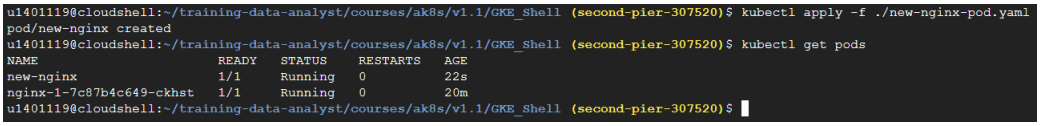

Figura 6.11: Aplicación y visualización del Pod.

Con el siguiente comando se accedió a la consola del `Pod` para posteriormente programar la página web:

```
kubectl exec -it new-nginx /bin/bash
```

En la figura 6.12 se puede observar el acceso la consola de`Pod` con permisos de administrador, sin tener que utilizar otra consola diferente a la de Cloud Shell.

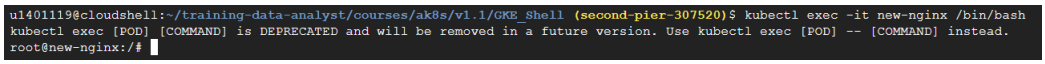

Figura 6.12: Ingreso a la consola del Pod.

Una vez dentro de la consola del `Pod` se ejecutaron los siguientes comandos para actualizar el `Pod`, instalar el editor de texto, crear el archivo fuente y programar la página web:

```
apt-get update
apt-get install nano
cd /usr/share/nginx/html
nano test.html
<html><header><title>this is title</title></header>
<body> Hello World </body>
</html>
exit
```

Luego se ejecutó el siguiente comando para exponer la página web a la red del `Clúster` para ello se declara el valor de `port-forward`, que permite exponer la página con la IP de `localhost` y un puerto de preferencia:

```
kubectl port-forward new-nginx 10081:80
```

En la figura 6.13 se puede observar la exposición exitosa del `Pod`, para poder seguir ejecutando comandos sin cancelar la exposición del `Pod` se debe abrir otra consola de Cloud Shell.

Universidad Militar Nueva Granada

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/GKE_Shell (second-pier-307520)$ kubectl port-forward new-nginx 10081:80
Forwarding from 127.0.0.1:10081 -> 80
```

Figura 6.13: Exposición de la página web a la red interna del clúster.

Finalmente se ejecutó el siguiente comando para visualizar el código fuente de la página web programada, finalizando así el despliegue de la implementación:

```
1 curl http://127.0.0.1:10081/test.html
```

```
u1401119@cloudshell:~ (second-pier-307520)$ curl http://127.0.0.1:10081/test.html
<html><header><title>this is title</title></header>
<body> Hello World </body>
</html>
u1401119@cloudshell:~ (second-pier-307520)$
```

Figura 6.14: Visualización del código fuente de la página web desde la consola de Cloud Shell.

La exposición de los objetos en Kubernetes a la red interna del `Clúster` permiten que los objetos logren comunicarse entre ellos según la programación de los mismos, en este caso es una página web con comunicación socket la cual no requiere sincronización, permitiendo una comunicación rápida entre los objetos de la red.

6.3 Implementación de un Despliegue y exposición por medio de un Servicio

En la tercera implementación se aplicó un `Despliegue` con el que gestiono tres `Pods` con un servicio de `nginx`, donde todos los `Pods` fueron expuestos a internet por medio de un objeto tipo servicio.

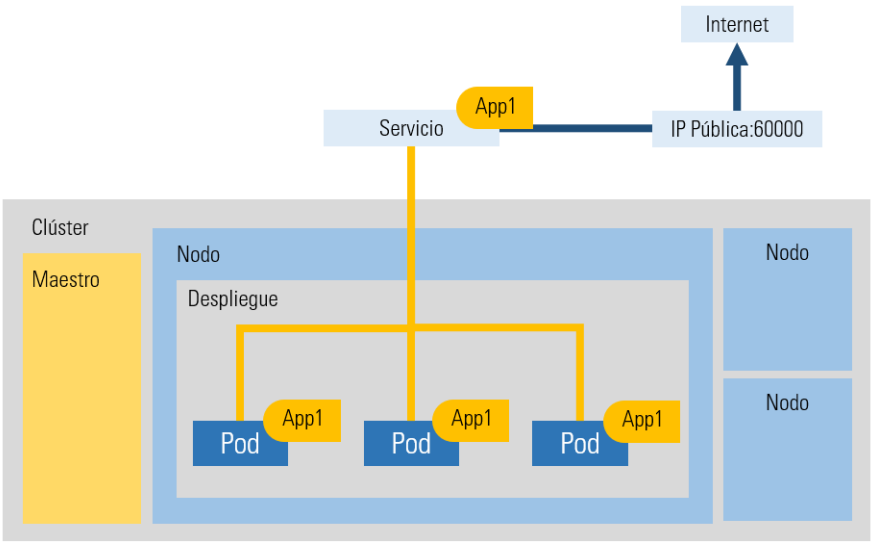

Figura 6.15: Topología de un clúster con un desplegué de múltiples Pods expuestos a internet.

Se inició la implementación ejecutando los siguientes comandos en Cloud Shell, que permiten la creación y adición de credenciales del `Clúster`:

```
export mi_zona=us-east1-b
export mi_cluster=standard-cluster-1
gcloud container clusters create $mi_cluster --num-nodes 3 --zone $mi_zona --enable-ip-
    alias
gcloud container clusters get-credentials $mi_cluster --zone $mi_zona
```

Luego se ejecutaron los siguientes comandos para aplicar el archivo `Yaml` del `Despliegue`, la última línea de comando permito observar la estructura del objeto aplicado:

```
cd training-data-analyst/courses/ak8s/v1.1/Deployments/
kubectl apply -f ./nginx-deployment.yaml
nano nginx-deployment.yaml
```

En la figura 6.16 se puede observar la estructura del `Despliegue`, que en este caso gestiona 3 `Pods` en los cuales existe un contenedor con servicio web `nginx`.

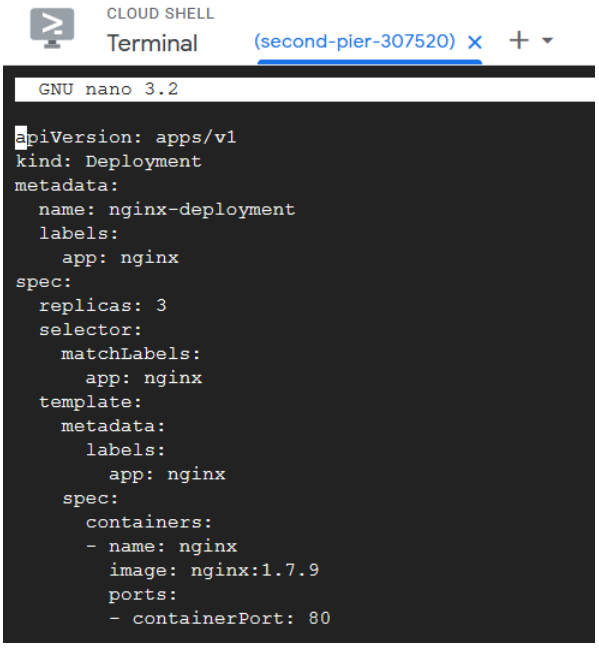

Figura 6.16: Archivo Yaml del despliegue de los Pods.

Posteriormente se ejecutó el siguiente comando con la intención de ver los estados de los `Pods`, en la figura 6.17 se puede observar el resultado de dicho comando:

```
kubectl get pods
```

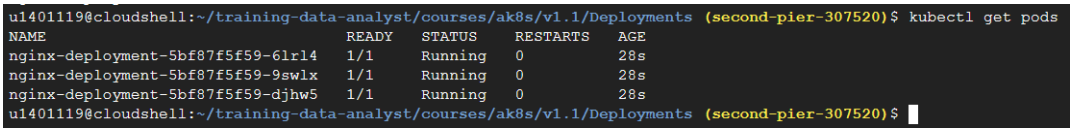

Figura 6.17: Visualización de los estados de los Pods.

Luego se realizó un desescalado y escalado del `Despliegue`, disminuyendo la cantidad total de `Pods` a 1 con la primera línea de comando y posteriormente con la segunda línea de comando se aumentó el `Despliegue` a 3 `Pods`:

```
kubectl scale --replicas=1 deployment nginx-deployment
kubectl scale --replicas=3 deployment nginx-deployment
```

En las dos anteriores implementaciones se utilizó un comando para exponer individualmente a los `Pods`, pero para esta implementación se utilizó un objeto que relaciona la etiqueta que tiene los `Pods` que son administrados por un mismo `Despliegue`, para aplicar y visualizar la estructura de dicho objeto se ejecutaron los siguientes comandos:

```
kubectl apply -f ./service-nginx.yaml
nano service-nginx.yaml
```

En la figura 6.18 se observa que el objeto `Servicio` expondrá todo aquel `Pod` que tenga como etiqueta `nginx` por medio de una IP pública otorgada por el valor de tipo de exposición `LoadBalancer`.

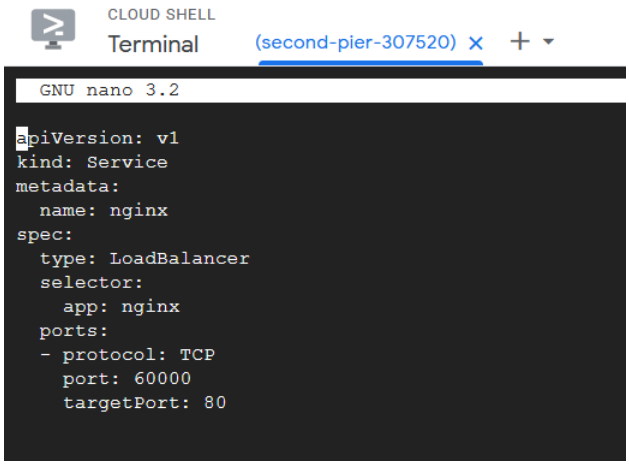

Figura 6.18: Archivo Yaml del servicio de exposición de la página web.

Luego se ejecutó el siguiente comando para visualizar la IP asignada por el objeto `Servicio`:

```
kubectl get service nginx
```

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Deployments (second-pier-307520)$ kubectl get service nginx
NAME    TYPE           CLUSTER-IP   EXTERNAL-IP      PORT(S)           AGE
nginx   LoadBalancer   10.76.5.38   35.243.220.215   60000:31133/TCP   54s
```

Figura 6.19: Visualización del servicio para exposición de los Pods.

En la figura 6.19 se puede observar la IP pública en la columna de `EXTERNAL_IP`. Con el siguiente comando se logró observar el servicio web activo en internet:

```
http://[EXTERNAL_IP]:60000/
```

En la figura 6.20 se puede observar el resultado de ejecutar el comando anterior en un buscador web. Esta implementación permite mantener activo un servicio expuesto debido a que no se requiere exponer los `Pods` individualmente cada vez que escala el `Despliegue` o cuando un `Despliegue` gestiona una gran cantidad de `Pods` y se quiere evitar aplicar un comando de exposición para cada `Pod`.

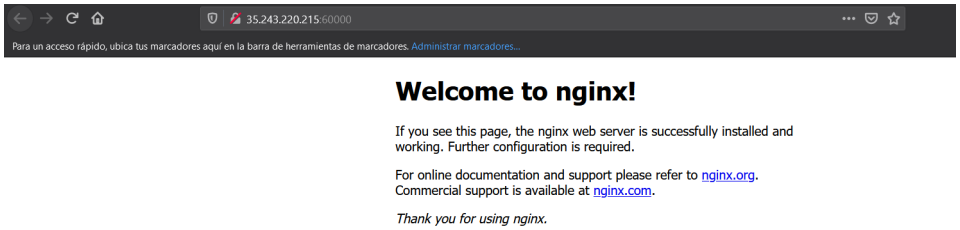

Figura 6.20: Visualización de servicio web activo desde un cliente en internet.

6.4 Implementación de trabajos y crono-trabajos

En la cuarta implementación se desplegaron dos objetos dentro de un solo `Clúster`, objetos relacionados con estados finitos; es decir, que los objetos realizaron tareas en vez de mantenerse activos durante toda la vida de la aplicación como en las tres anteriores implementaciones.

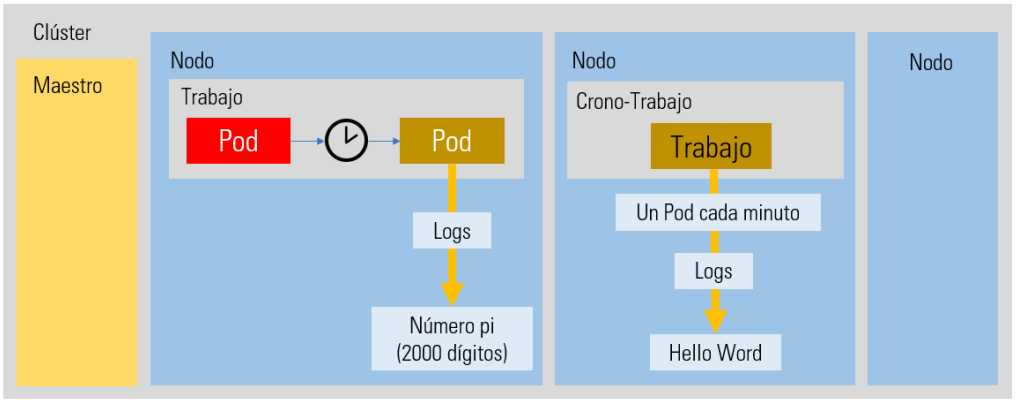

Figura 6.21: Topología de un clúster con un trabajo y un crono-trabajo.

El objeto `Trabajo` ejecutará el cálculo de los primeros 2000 dígitos de PI y el objeto `Crono-trabajo` ejecutará un `Trabajo` cada minuto para imprimir un `Hello Word`, Como se observa en la figura 6.21 ninguno de los dos objetos estará expuesto a internet ni a la red interna del `Clúster`. Se inició con la creación del `Clúster` y la asignación de credenciales con la ejecución de los siguientes comandos:

```
1 export mi_zona=us-east1-b
2 export mi_cluster=standard-cluster-1
3 gcloud container clusters create $mi_cluster --num-nodes 3 --zone $mi_zona --enable-ip-
      alias
4 gcloud container clusters get-credentials $mi_cluster --zone $mi_zona
```

Luego se buscó y se aplicó el archivo `Yaml` del objeto `Trabajo`, con la ejecución de los siguientes comandos:

```
1 cd training-data-analyst/courses/ak8s/v1.1/Jobs_CronJobs/
2 kubectl apply -f example-job.yaml
3 kubectl describe job example-job
```

En la figura 6.22 se puede observar la estructura del objeto `Trabajo` aplicado, donde por medio de lenguaje `Perl` calculara e imprimirá los primeros 2000 dígitos del número PI.

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Jobs_CronJobs (second-pier-307520)$ kubectl describe job example-job
Name:           example-job
Namespace:      default
Selector:       controller-uid=d6e06e1f-58d3-4e3b-a0ba-3d3e4a38c861
Labels:         controller-uid=d6e06e1f-58d3-4e3b-a0ba-3d3e4a38c861
                job-name=example-job
Annotations:    <none>
Parallelism:    1
Completions:    1
Start Time:     Tue, 04 May 2021 15:54:19 +0000
Pods Statuses:  1 Running / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  controller-uid=d6e06e1f-58d3-4e3b-a0ba-3d3e4a38c861
           job-name=example-job
  Containers:
   pi:
    Image:      perl
    Port:       <none>
    Host Port:  <none>
    Command:
      perl
    Args:
      -Mbignum=bpi
      -wle
      print bpi(2000)
    Environment:  <none>
    Mounts:       <none>
  Volumes:        <none>
Events:
  Type    Reason            Age   From            Message
  ----    ------            ----  ----            -------
  Normal  SuccessfulCreate  12s   job-controller  Created pod: example-job-k2nxt
```

Figura 6.22: Información del trabajo desplegado.

Para observar el estado del objeto `Trabajo` se ejecutó el siguiente comando y en la figura 6.23 se puede observar el resultado de la ejecución del comando:

```
1 kubectl get jobs
```

Como se puede observar en la figura 6.23, la tarea fue realizada con éxito y tuvo una duración de 31 segundo.

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Jobs_CronJobs (second-pier-307520)$ kubectl get jobs
NAME          COMPLETIONS   DURATION   AGE
example-job   1/1           31s        50s
```

Figura 6.23: Visualización de los Trabajos.

Para observar el estado del `Pod` contenido por el `Trabajo` se ejecutó el siguiente comando y en la figura 6.24 se puede observar el resultado de la ejecución del comando:

```
1 kubectl get pods
```

En las anteriores implementaciones los `Pods` han tenido el estado infinito `Running`, pero en este caso y como se puede observar en la figura 6.24 este estado cambia a `Completed`, lo que indica que la tarea que realizo ha terminado.

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Jobs_CronJobs (second-pier-307520)$ kubectl get pods
NAME                READY   STATUS      RESTARTS   AGE
example-job-k2nxt   0/1     Completed   0          3m9s
```

Figura 6.24: Visualización de los Pods.

Debido a que el objeto `Trabajo` no fue expuesto a ninguna red, no se podrá observar su resultado desde una búsqueda por IP como en anteriores implementaciones, por ello se ejecutó el siguiente comando y en la figura 6.25 se puede observar el resultado de la tarea realizada por el objeto:

```
1 kubectl logs [NOMBRE_POD]
```

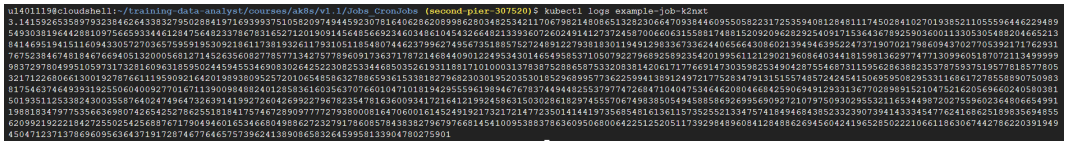

Figura 6.25: Visualización de los primeros 2000 dígitos de PI impresos por el Pod.

Luego se ejecutaron los siguientes comandos para aplicar y visualizar la estructura del objeto `Crono-trabajo`:

```
kubectl apply -f example-cronjob.yaml
nano example-cronjob.yaml
```

Dentro de la estructura del `Crono-trabajo` que se observa en la figura 6.26, está configurado para ejecutar un `Trabajo` que ejecutara en consola del contenedor un `"Hello, World!"`.

Figura 6.26: Archivo Yaml del crono-trabajo.

Luego se ejecutó el siguiente comando para visualizar la impresión del `"Hello, World!"`:

```
kubectl logs [NOMBRE_POD]
```

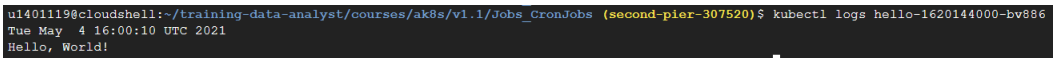

Figura 6.27: Visualización del resultado del Pod desplegado con el crono-trabajo.

Finalmente se ejecutó el siguiente comando para comprobar la ejecución periódica de los `Trabajos` cada minuto:

```
kubectl get jobs
```

En la figura 6.28 se puede observar la diferencia en los tiempos entre los `Trabajos`, que coinciden con la ejecución del cada un minuto del `Crono-trabajo`.

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Jobs_CronJobs (second-pier-307520)$ kubectl get jobs
NAME              COMPLETIONS   DURATION   AGE
hello-1620144060  1/1           2s         2m24s
hello-1620144120  1/1           2s         84s
hello-1620144180  1/1           2s         24s
```

Figura 6.28: Visualización de los trabajos desplegados con el crono-trabajo.

6.5 Implementación de un Despliegue con autoescalado

En la última implementación se ejecutaron dos `Despliegues`, uno que simula ser un servicio web que recibe tráfico y escala su estructura según la cantidad de tráfico que recibe, el otro `Despliegue` que genera tráfico por medio de peticiones por consola del los contenedores.

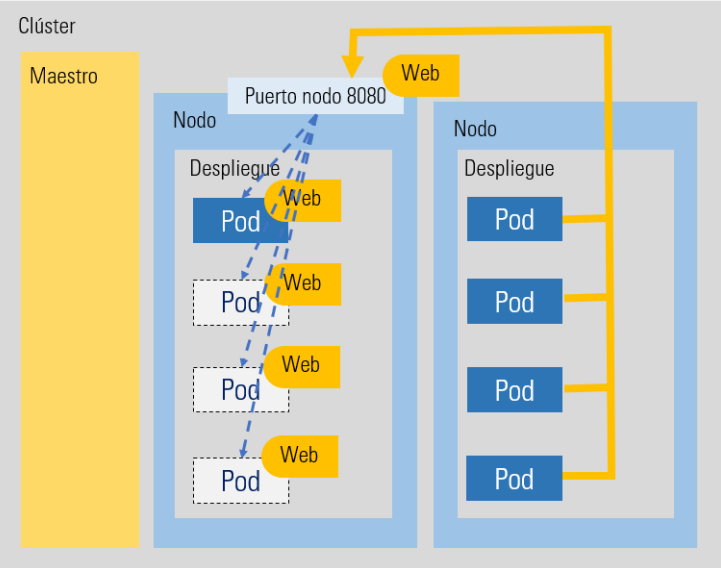

Figura 6.29: Topología de un clúster con un despliegue autoescalable y un despliegue generador de tráfico.

Se inició con la ejecución de los siguientes comandos para realizar la creación de `Clúster`:

```
export mi_zona=us-east1-b
export mi_cluster=standard-cluster-1
gcloud container clusters create $mi_cluster --num-nodes 2 --zone $mi_zona --enable-ip-
    alias
gcloud container clusters get-credentials $mi_cluster --zone $mi_zona
```

Luego se creó y se visualizó el estado del `Despligue` de la parte derecha de la topología con los siguientes comandos:

```
cd training-data-analyst/courses/ak8s/v1.1/Autoscaling
kubectl create -f web.yaml --save-config
kubectl get deployment
```

En la figura 6.30 se puede visualizar el resultado del último comando aplicado, como se puede observar el `Despliegue` solo tendrá un `Pod` sin importar la cantidad de tráfico que reciba el `Pod`.

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$ kubectl get deployment
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
web    1/1     1            1           39s
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$
```

Figura 6.30: Visualización del Despliegue.

Para exponer el `Despliegue` y lograr automatizar el escalado según el tráfico que reciba, se aplicaron los siguientes comandos:

```
kubectl expose deployment web --target-port=8080 --type=NodePort
kubectl autoscale deployment web --max 4 --min 1 --cpu-percent 3
```

Luego se ejecutó el siguiente comando para visualizar las características de autoescalado y capacidad de cómputo del `Despliegue`:

```
kubectl get hpa
```

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$ kubectl get hpa
NAME   REFERENCE        TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
web    Deployment/web   0%/3%     1         4         1          73s
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$
```

Figura 6.31: Autoescalado y Consumo de procesamiento del Despliegue.

Para comprobar el funcionamiento del autoescalado, se ejecutaron los siguientes comandos para aplicar el `Despliegue` generador de tráfico:

```
kubectl apply -f loadgen.yaml
nano loadgen.yaml
```

En la figura 6.32 se puede observar la estructura del generador de tráfico, que realiza peticiones hacia el puerto de exposición del `Despliegue` con autoescalado.

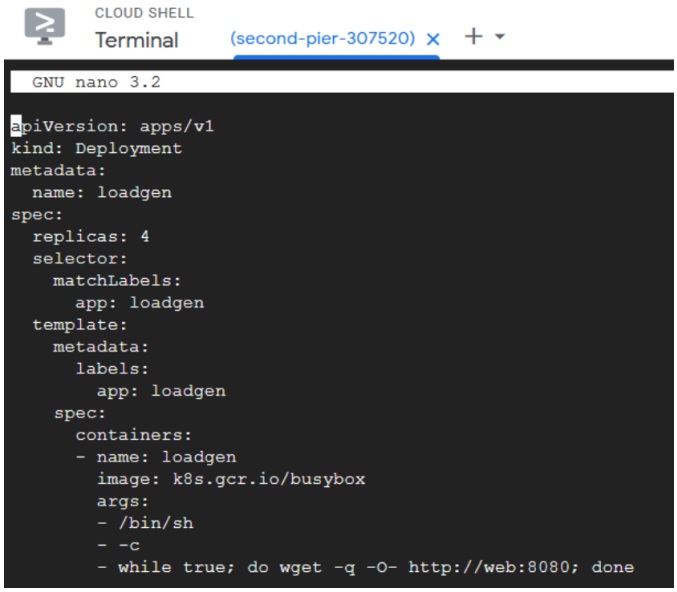

Figura 6.32: Archivo Yaml del Despliegue generador de tráfico.

Luego se aplicó el siguiente comando para visualizar el estado de los `Despliegues` y observar el número de `Pods` que tiene activo en el momento:

```
kubectl get deployment
```

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$ kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
loadgen   4/4     4            4           7s
web       1/1     1            1           5m10s
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$
```

Figura 6.33: Visualización de los Despliegues.

Se volvió a ejecutar el mismo comando para visualizar el aumento de número de `Pods`, que son añadidos por el autoescalado:

```
kubectl get deployment
```

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$ kubectl get deployment
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
loadgen   4/4     4            4           43s
web       2/4     4            2           5m46s
```

Figura 6.34: Visualización del autoescalado del Despliegue.

Luego se ejecutó el siguiente comando para visualizar el nuevo consumo de cómputo del `Despliegue` con autoescalado:

```
kubectl get hpa
```

```
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$ kubectl get hpa
NAME   REFERENCE        TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
web    Deployment/web   65%/3%    1         4         4          5m21s
u1401119@cloudshell:~/training-data-analyst/courses/ak8s/v1.1/Autoscaling (second-pier-307520)$
```

Figura 6.35: Aumento de consumo de procesamiento del Despliegue autoescalado.

Finalmente se ejecutó el siguiente comando para detener la generación de tráfico:

```
kubectl scale deployment loadgen --replicas 0
```

Esta implementación permite que se genere un escenario de estrés sobre una aplicación en concreto lo que permitió observar el funcionamiento de la característica de autoescalado, optimizando la aplicación para que tenga una respuesta correcta respecto a la cantidad de peticiones que recibe, lo que se traduce en un servicio continuo de la aplicación.

UNIVERSIDAD MILITAR NUEVA GRANADA

7. Monitoreo de una implementación

Uno de los principales procesos que se ejecutan cuando se despliega una implementación exitosa en Kubernetes, es el poder realizar un monitoreo efectivo que permita comprender que está sucediendo con el sistema y los objetos. Para ello se crea un Dashboard seccionado en métricas para los Nodos y métricas para los Pods. También se crean políticas de alarmas que envían el reporte a un canal de notificación diferente a GCP.

7.1 Topología de la implementación a monitorear

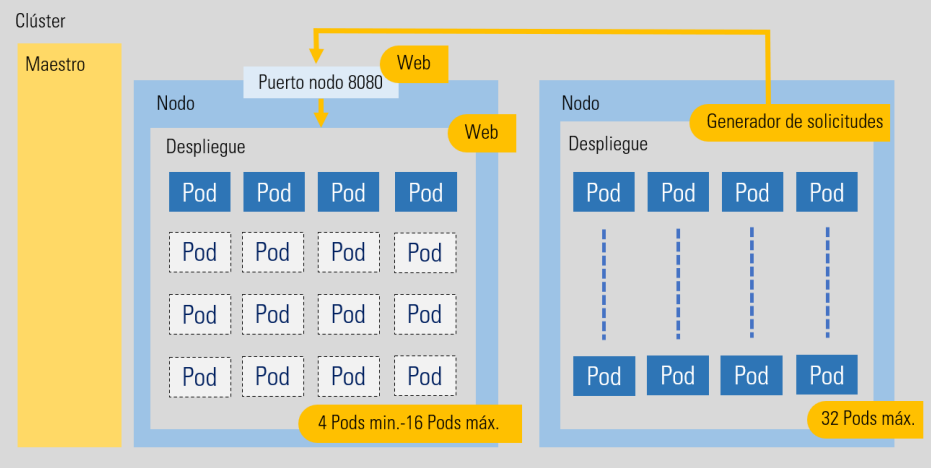

Figura 7.1: Topología implementada para hacer monitoreo desde GCP monitoring.

Para realizar monitoreo sobre una implementación se hizo un aumento en el número de Pod que tiene la implementación 5 del anterior capítulo. Para ello se mejoró la capacidad de cómputo de los nodos, como también se modificó el número de réplicas en los archivos Yaml de los despliegues. En la figura 7.1 se puede observar la cantidad de Pods que recibirán las solicitudes y los Pods que las enviaran.

7.2 Configuración y despliegue del clúster a monitorear

Para lograr aumentar el número de Pods del despliegue se buscó aumentar el número de CPUs, debido a que es un requerimiento para lograr que un Pod de web se ejecute. Teniendo en cuenta que cada Pod consume en su ejecución el 25 % de una CPU. En la figura 7.2 se puede observar el tipo y número de CPUs que tendrán los dos nodos del clúster.

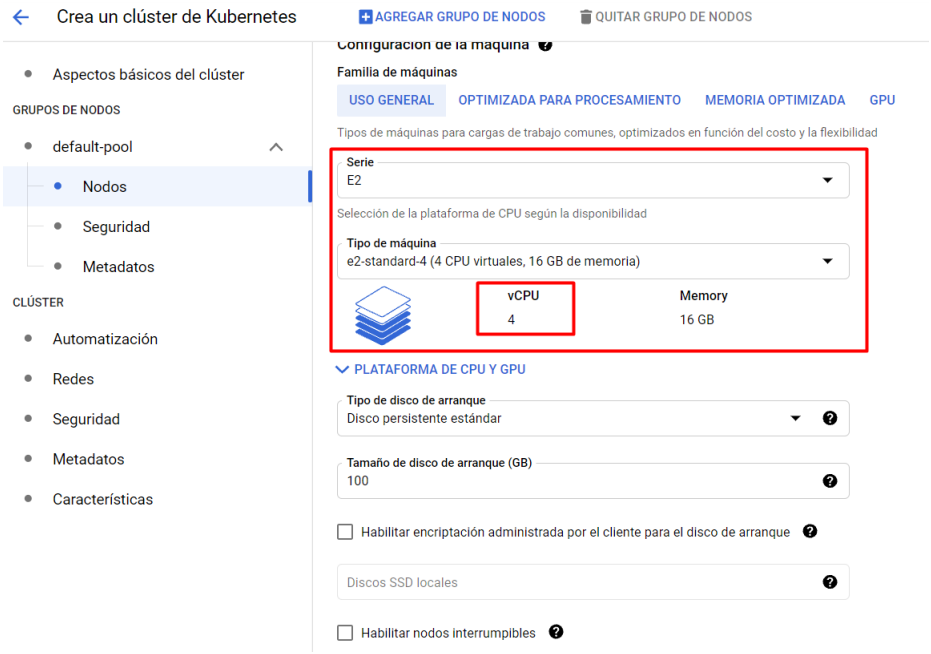

Figura 7.2: Selección de tipo y número de nodos del clúster.

Para lograr realizar un monitoreo y seguimiento por alarmar del clúster y sus objetos, es necesario habilitar el servicio de Cloud Monitoring, como se observa en la figura 7.3.

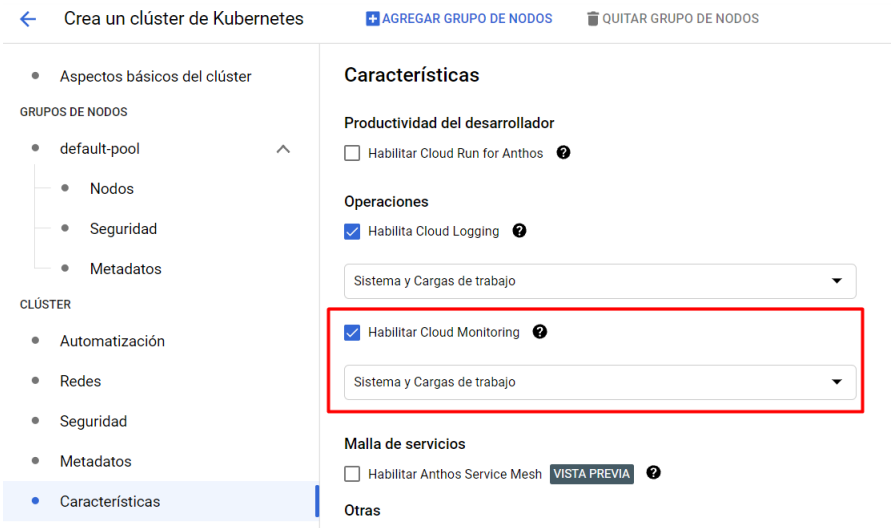

Figura 7.3: habilitación de monitoreo de objetos y nodos del clúster.

Finalmente en la figura 7.4 se puede observar el comando que se podría aplicar al Shell de

Google para crear el clúster con los nodos mejorados y con el monitoreo habilitado.

Figura 7.4: Comando completo para desplegar el clúster.

Una vez aplicado el clúster, se puede observar la no existencia de algún tipo de fallo en el clúster y su satisfactorio despliegue.

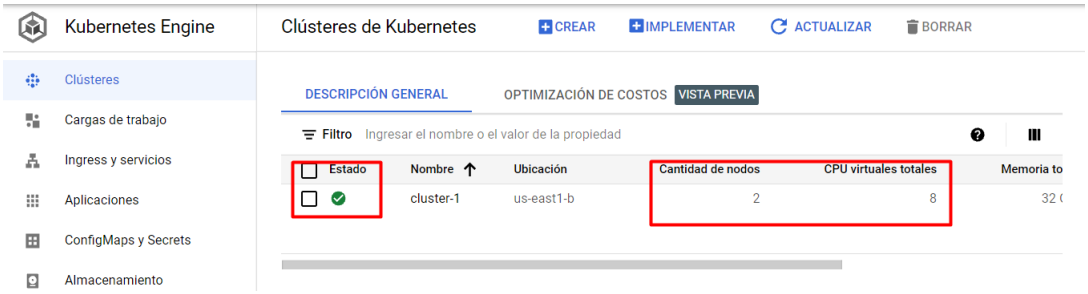

Figura 7.5: Clúster desplegado en Kubernetes Engine.

7.3 Aplicación de objetos del clúster a monitorear

Aplicando los siguientes comandos se lograron desplegar los dos archivos Yaml de las figuras 7.6 y 7.7, permitiendo dejar la implementación en modo espera para iniciar pruebas de estrés.

```
kubectl create -f web.yaml --save-config
kubectl apply -f loadgen.yaml
kubectl expose deployment web --target-port=8080 --type=NodePort
kubectl autoscale deployment web --max 16 --min 4 --cpu-percent 70
kubectl scale deployment loadgen --replicas 0
```

Los objetos no tuvieron grandes cambios respecto a la implementación 5 del capítulo anterior, únicamente se aumentó el número mínimo de 1 a 4 Pods activos mínimos del despliegue web.

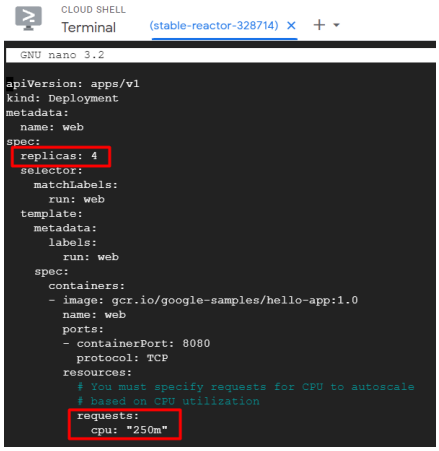

Figura 7.6: Archivo Yaml del despliegue web y cuota de consumo de CPU por cada Pod.

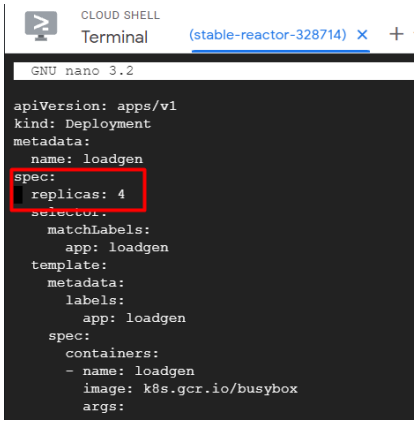

Figura 7.7: Archivo loadgen.yaml (Generador de peticiones).

Cuando se refiere a que la implementación está en modo espera, es cuando el despliegue Loadgen tiene 0 Pods emitiendo solicitudes, permitiendo que los Pods de web se mantenga en 4 mínimos Pods activos como se observa en la figura 7.8.

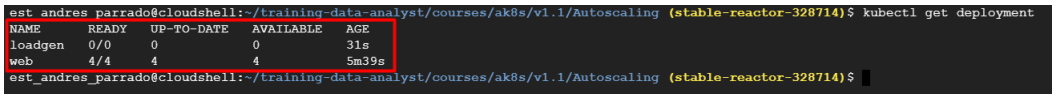

Figura 7.8: Objetos loadgen y web activos en espera de ejecución de escenario de estrés.

7.4 Monitoreo del clúster y objetos de Kubernetes

Una vez que se habilita el monitoreo del clúster de Kubernetes, se habilita la visualización de la actividad de diferentes métricas sobre los estados de los objetos y del sistema que integra la

implementación. Es necesario establecer un Dashboard que permita relacionar las métricas con los objetos y el sistema a monitorear.

7.4.1 Configuración de Dashboard y métricas

Para configurar el Dashboard del sistema u objetos a monitorear es necesario seguir la ruta de GCP de la figura 7.9

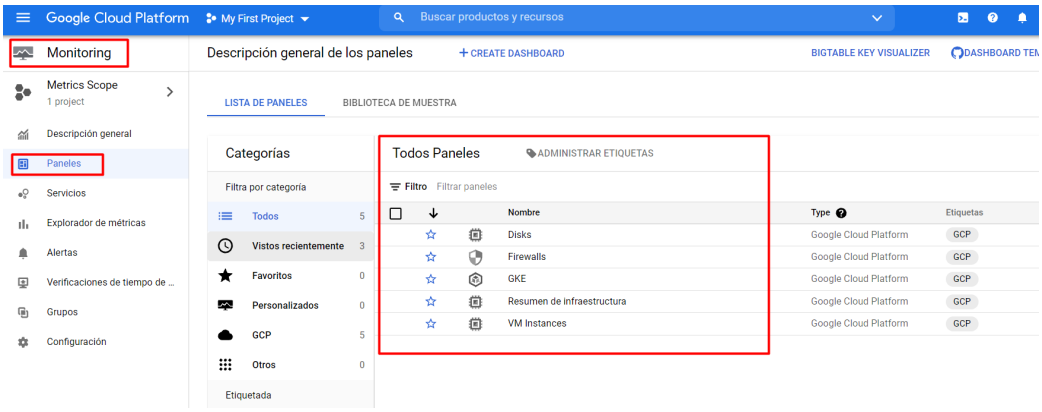

Figura 7.9: Ruta de visualización de Dashboards, presentes en el servicio de monitoreo de GCP.

El servicio de monitoreo de GCP, permite generar un dashboard totalmente personalizado con diferentes tipos de tablas y métricas sobre una cuadrícula para lograr una distribución deseada. En la figura 7.10 se puede observar los principales tipos de gráficas.

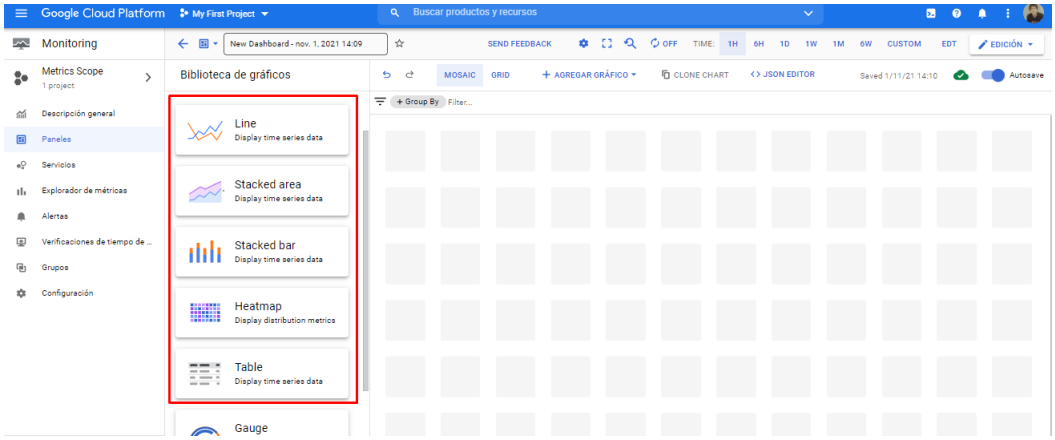

Figura 7.10: Cuadricula del Dashboard y tipos de gráficas.

Para generar monitoreo sobre una métrica se inicia con la selección del tipo de gráfica luego seleccionar el tipo de recurso y finalmente el recurso en específico, a continuación se mostraran algunas de las configuraciones que se realizaron para métricas específicas.

Métrica de uso promedio de memoria de todos los Nodos

Se selecciona una gráfica típica para monitorear la memoria en uso vs. tiempo, dicha métrica nos mostrará la memoria en uso promedio de todos los nodos de la implementación. En la figura 7.11 se puede observar la selección del tipo de recurso y recurso específico en la parte izquierda señalada, en la parte derecha señalada se puede observar la gráfica generada y en el medio se

Universidad Militar Nueva Granada

puede observar otra gráfica previamente configurada que servirá de referencia para saber cuál es el uso máximo posible de memoria.

Figura 7.11: Configuración de gráfica del Dashboard para monitorear la memoria en uso promedio por todos los Nodos.

Métrica de uso de CPU de cada Nodo

Se selecciona una gráfica típica para monitorear el uso de CPU vs. tiempo, esta métrica mostrará la cantidad porcentual de uso de CPU individualmente de los dos Nodos del clúster. En la parte derecha de la figura 7.12 se puede observar la configuración de la métrica, en la parte derecha la gráfica generada y en el medio se observa la cantidad promedio de CPU de cada Nodo y el valor exacto de CPU usable.

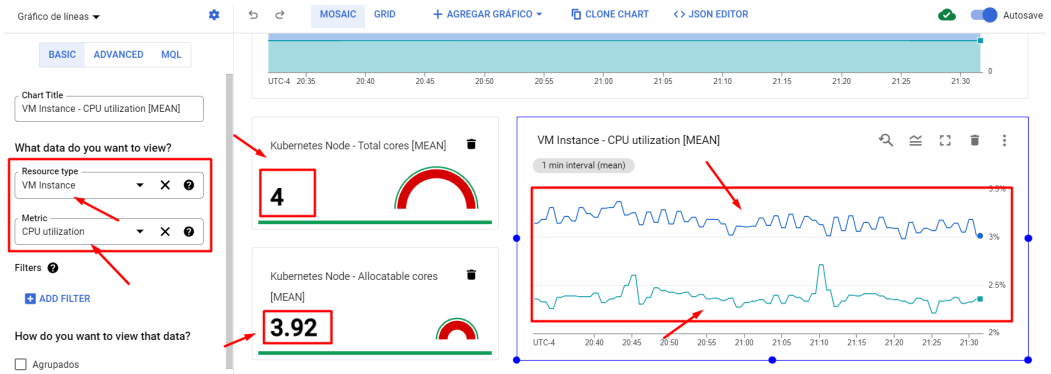

Figura 7.12: Configuración de gráfica del Dashboard para monitorear el uso de CPU por los dos Nodos.

Métrica de bits recibidos por cada uno de los Pods

Se selecciona una gráfica típica para monitorear la cantidad de bits recibidos vs. tiempo, esta métrica mostrará un promedio de bits recibidos por cada uno de los Pod del despliegue Web.yaml, este tipo de gráfica permite seleccionar que Pod queremos visualizar en la gráfica. Como se observa en la figura 7.13 y recordando que la implementación está en modo espera del escenario de estrés, no se puede observar bits recibidos por los Pods.

Universidad Militar Nueva Granada

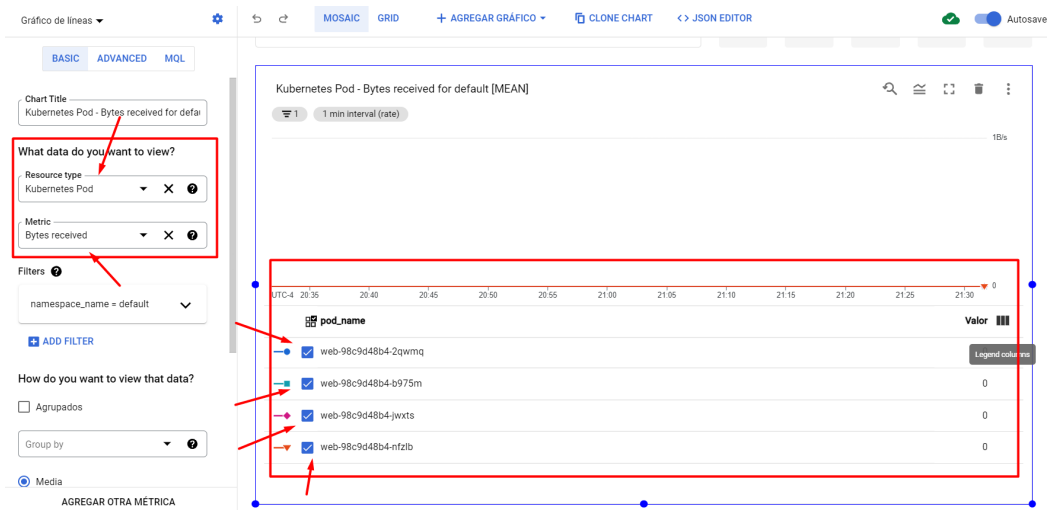

Figura 7.13: Configuración de gráfica del Dashboard para monitorear el uso de CPU por los dos Nodos.

Dashboard final para monitoreo de Nodos y Pods

El Dashboard fue seccionado en dos partes:

- Monitoreo del sistema del clúster: Nodos de la implementación
- Monitoreo de las cargas de trabajo del clúster: Pods de la implementación

En la figura 7.14 se puede observar las gráficas de métricas relacionadas con el monitoreo de Nodos de la implementación, en la parte izquierda se pueden visualizar gráficas con valores de referencia de métricas de monitoreo de los Nodos y en la parte derecha está el comportamiento de las métricas de los Nodos.

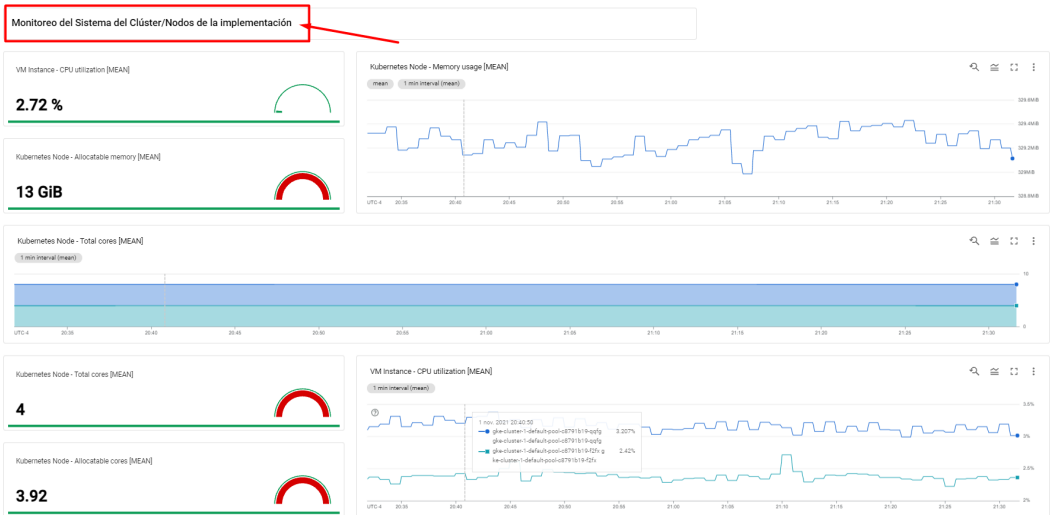

Figura 7.14: Sección parcial del Dashboard con métricas de monitoreo de los Nodos.

En la figura 7.15 se puede observar la última gráfica referente al monitoreo de Nodos y la primera gráfica de monitoreo de Pods. Por último, en la figura 7.16 se puede observar las tres últimas gráficas de monitoreo de Pods.

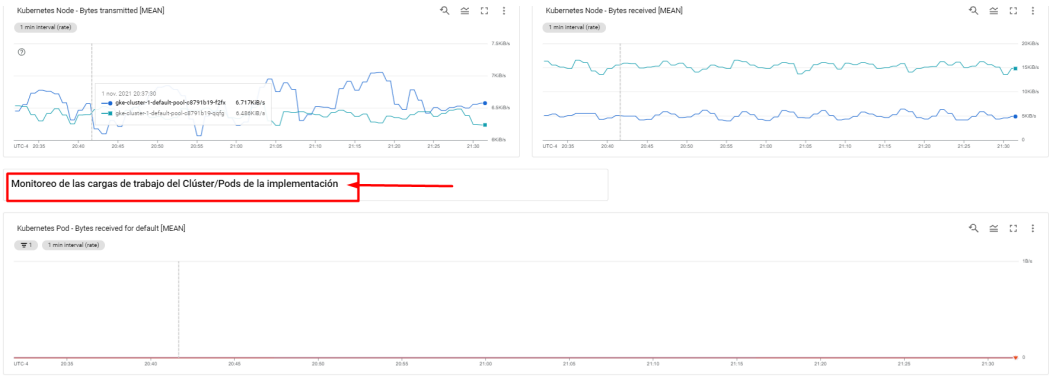

Figura 7.15: Sección mixta del Dashboard con métricas de monitoreo de los Nodos y los Pods.

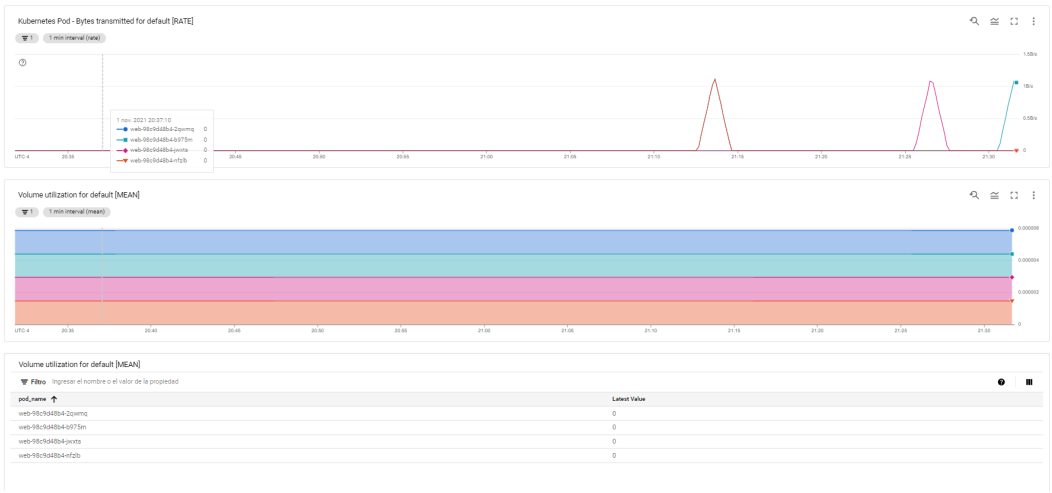

Figura 7.16: Sección parcial del Dashboard con métricas de monitoreo de los Pods.

7.4.2 Configuración de alarmas y canales de notificación

Una de las buenas prácticas al momento de monitorear servicios en cloud es habilitar un canal de notificaciones, que permita que el monitoreo se establezca sin mantener una presencia activa sobre CGP; es decir, automatizar la generación de incidentes evitando que exista un recurso humano que reporte estado de los objetos y del sistema. El establecimiento de políticas de alarmas se realiza con el mismo servicios de monitoreo de GCP, como se observa en la figura 7.17.

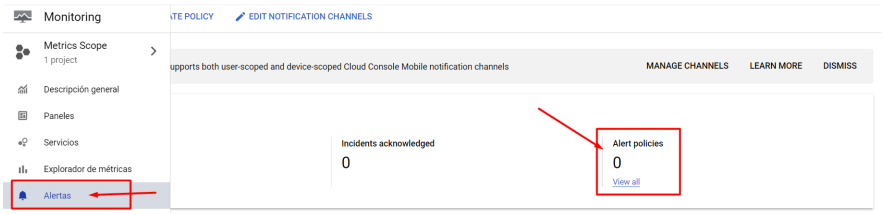

Figura 7.17: Ruta de visualización de políticas de alertas, presentes en el servicio de monitoreo de GCP.

Universidad Militar Nueva Granada

Configuración de alarmas

Para generar una política de alarma se debe relacionar una métrica con un umbral específico de activación y un canal notificación de alerta. En las figuras 7.18 y 7.19 se puede observar la configuración de una alarma relacionada con el uso excesivo de memoria por parte de los Nodos.

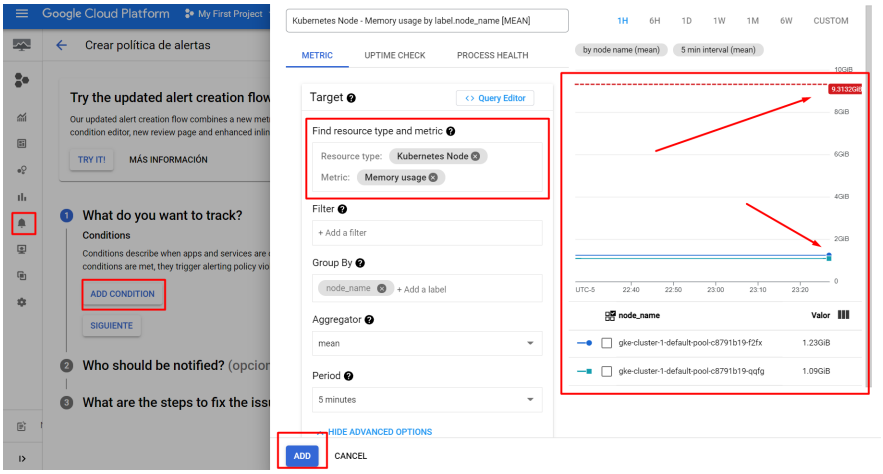

Figura 7.18: Configuración de la métrica para establecer alarma sobre uso de memoria del Nodo.

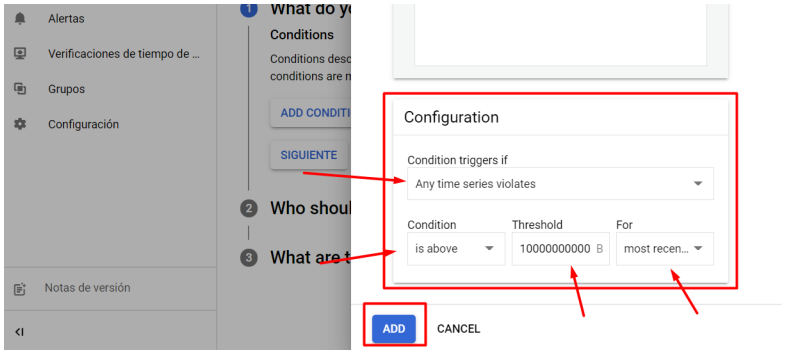

Figura 7.19: Configuración de umbral de activación de alarma.

Canal de notificaciones

El canal de notificaciones seleccionado fue por SMS, debido a que se podría conocer el estado del clúster desde el teléfono móvil, ya que desde el teléfono móvil no se tiene acceso óptimo a GCP.

Figura 7.20: Notificaciones de alertas configuradas para ser enviadas por SMS a número personal.

Alarmas configuradas

En la figura 7.21 se pueden observar 4 alarmas creadas, dos para el monitoreo de Nodos y dos para los Pod. Las alarmas fueron relacionadas con métricas de memoria y CPU debido a que son la principal característica para que se ejecuten los contenedores y se escalen automáticamente los Pods, que se ponen a prueba con un escenario de estrés.

Figura 7.21: Alarmas configuradas con canal de notificación habilitada.

7.5 Monitoreo y activación de escenario de estrés

La ejecución de escenario de estrés se realizó en 8 etapas:

- **Etapa 1**: Generación de tráfico mínimo, loadgen con 1 Pod por 10 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 1
```

- **Etapa 2**: Generación de tráfico, loadgen con 4 Pods por 10 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 4
```

- **Etapa 3**: Generación de tráfico, loadgen con 32 Pods por 15 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 32
```

- **Etapa 4**: Generación de tráfico, loadgen con 128 Pods por 5 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 128
```

- **Etapa 5**: Generación de tráfico, loadgen con 384 Pods por 12 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 384
```

- **Etapa 6**: Suspensión de tráfico, loadgen con 0 Pods por 11 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 0
```

- **Etapa 7**: Generación de tráfico, loadgen con 384 Pods por 7 min. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 384
```

- **Etapa 8**: Suspensión de tráfico, loadgen con 0 Pods. Comando de escalado aplicado:

```
kubectl scale deployment loadgen --replicas 0
```

Para verificar como aumenta y disminuye el uso de memoria, volumen y tráfico de transmisión o recepción en cada una de las etapas, se debe generar un lapso de tiempo específico a graficar. En la figura 7.22 se puede observar dicho lapso de tiempo.

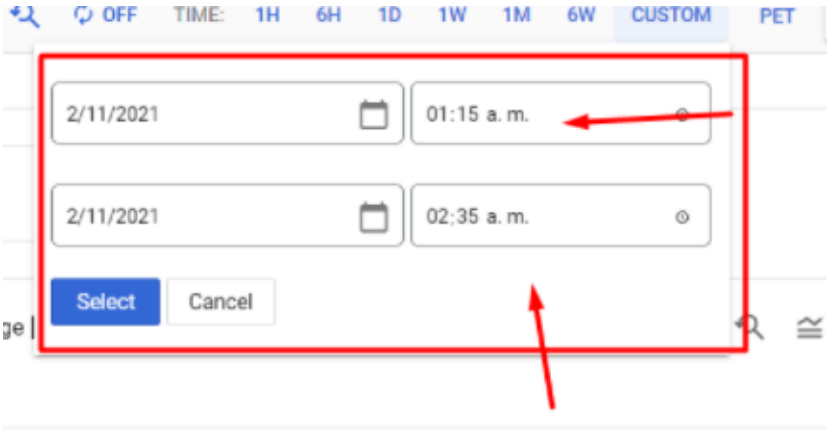

Figura 7.22: Especificación de tiempo de muestra de métricas en las gráficas.

Fueron 3 de las métricas más descriptivas en donde se puede observar cada uno de los escalones de aumento y disminución de las etapas, donde se aplicaba escalado de Pods.

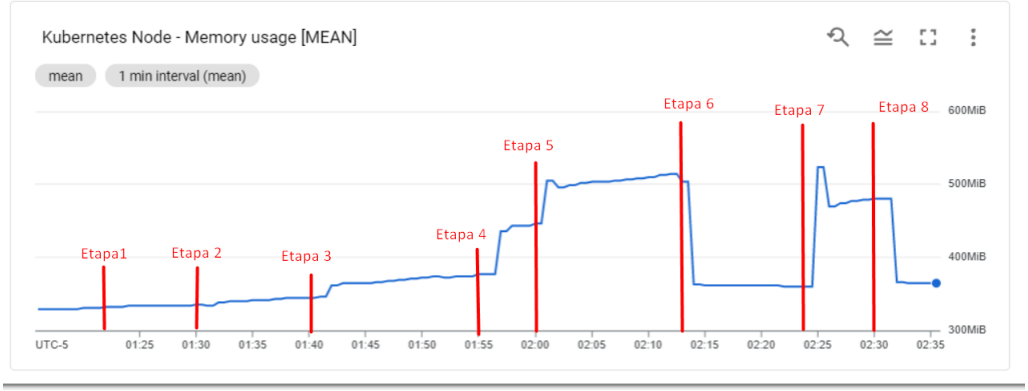

Figura 7.23: Gráfica de uso de memoria promedio de todos los nodos vs. tiempo.

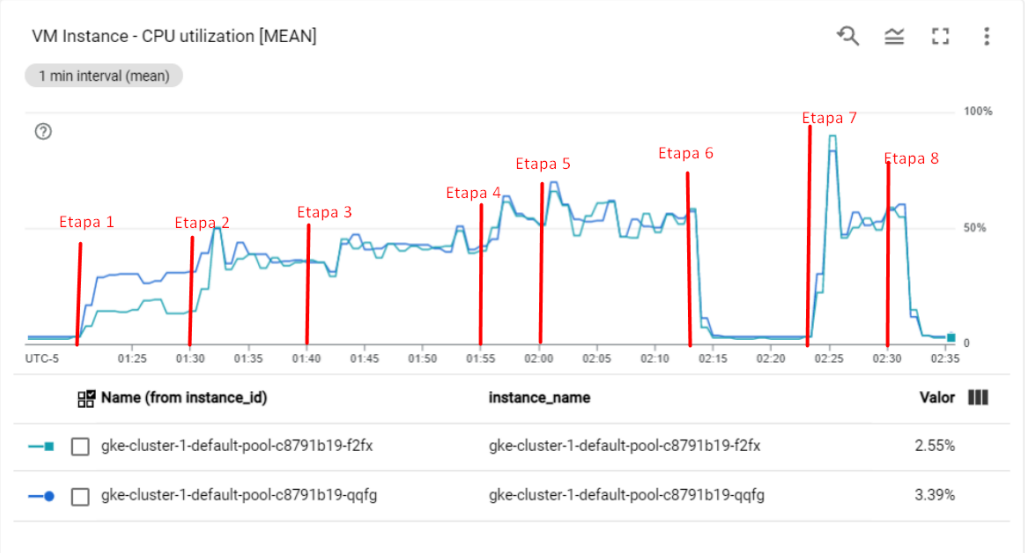

Figura 7.24: Gráfica de utilización de CPU por cada Nodo vs. tiempo.

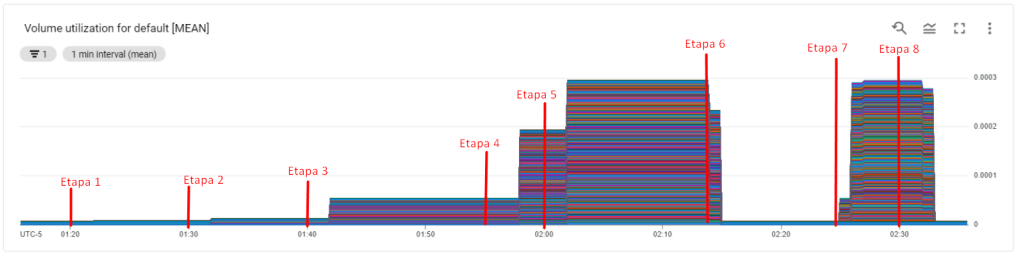

Figura 7.25: Gráfica de uso de memoria tipo volume de cada Pod vs. tiempo.

Recepción de notificaciones

Algunos de los valores de las métricas no superaron los umbrales de alarma causando que solo se lograra notificar por medio de SMS el uso de cores de CPU y el uso de memoria. Debido a ello en la figura 7.26 solo se puede observar dos clases de notificaciones.

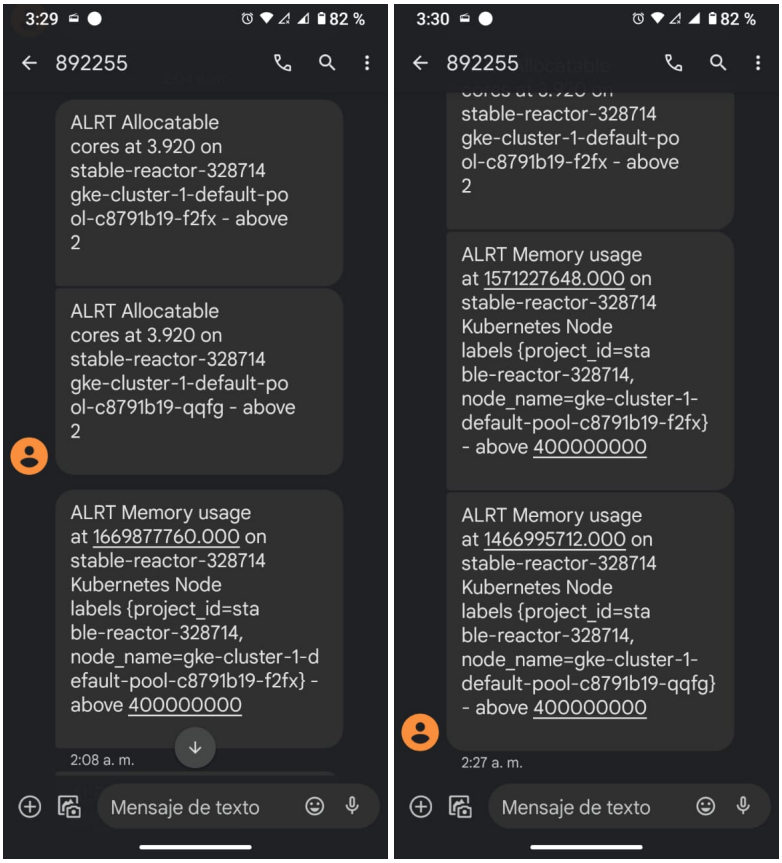

Figura 7.26: Notificación de utilización de cores de CPU por parte de los Nodos y disminución de uso de memoria por un Nodo.

8. Conclusiones

Conclusiones

- La disposición de créditos gratuitos de Google Cloud que pueden ser implementados específicamente en el servicio de Kubernetes en la nube, permite que el proveedor se destaque sobre otros proveedores cuando no se cuenta con presupuesto para realizar implementaciones, siendo el perfecto proveedor en el sector de implementaciones de carácter educativo e investigativo, donde se requieren hacer exploraciones y pruebas sobre el servicio de Kubernetes.
- El acceso a la documentación sobre el sistema de Kubernetes otorgada por el proveedor como también por el mismo sistema. Permitió comprender la teoría de Kubernetes en la nube, pero logro ser mucho más significativo el poder explorar y realizar implementaciones sobre la Plataforma de Google Cloud, debido a la complejidad de la teoría de los objetos y la estructura del clúster, así que una vez desplegados los objetos en una implementación real se logra comprender la operatividad del sistema de Kubernetes que se describe en la teoría.
- Debido a la sintaxis de kubectl y facilidad del lenguaje Yaml, sé logra comprender como el sistema de Kubernetes mantiene redundancia y disponibilidad de los aplicativos debido al establecimiento de estados de los objetos. La orquestación de múltiples objetos debido a utilización de etiquetas con Yaml y la facilidad de lectura del código permiten determinar una infraestructura por código.
- Los flujogramas y cuadros comparativos expuestos son el resultado de comprender cuáles son las características o necesidades de un aplicativo basado en contenedores con Kubernetes, que se deben tener en cuenta para administrar o implementar múltiples objetos, un objeto compuesto específico, una actualización o aplicar un comando en el sistema Kubernetes del aplicativo.
- El sistema Kubernetes se compone de muchos más objeto que no fueron caracterizados en este trabajo, como pueden llegar a ser los objetos relacionados con bases de datos y gestión de identidades del sistema Kubernetes. El caracterizar los objetos como bloques que pueden llegar a ser ubicados dentro de un clúster permite comprender como interactúan entre sí y reconocer como funcionan otros objetos sin tener que profundizar en su teoría, debido a la simplicidad y equidad a la hora de representar una arquitectura, topología, elementos u objetos del sistema de Kubernetes.

- Las implementaciones presentadas en un formato de tipo manual permiten generar una base de datos del conocimiento, para desarrollar futuras pruebas sobre aplicaciones con contenedores en el sistema Kubernetes en la nube, permitiendo la fomentación del despliegue de servicios con las tecnologías emergentes como lo es la computación en la nube nativa.
- Una gran cantidad de métricas que se pueden utilizar con GCP monitoring para los objetos en Kubernetes están en fase de experimentación. Por ello es mucho más fácil generar monitoreo sobre los Nodos y se vuelve más eficiente mantener un monitoreo con GCP monitoring y canales de notificación para los Nodos; por otro lado, se debería utilizar kubectl para los objetos de Kubernetes. Ya que los Pods están diseñados para ser desechables y fáciles de volver a desplegar, permitiendo tener un monitoreo menos estricto.

Bibliografías

[1] Gibson, J., Rondeau, R., Eveleigh, D., & Tan, Q. (2012, November). Benefits and challenges of three cloud computing service models. In 2012 Fourth International Conference on Computational Aspects of Social Networks (CASoN) (pp. 198-205). IEEE. https://ieeexplore.ieee.org/abstract/document/6412402

[2] G.ST.02 Guía de Computación en la nube - Ministerio de Tecnologías de la Información y las Comunicaciones, Mintic.gov.co, 2017. https://www.mintic.gov.co/arquitecturati/630/w3-article-75554.html

[3] Mohammadi, S., & Mohammadi, A. (2014). Effect of cloud computing in accounting and comparison with the traditional model. Research Journal of Finance and Accounting, 5(23), 104-114. https://core.ac.uk/download/pdf/234630329.pdf

[4] Gannon, D., Barga, R., & Sundaresan, N. (2017). Cloud-native applications. IEEE Cloud Computing, 4(5), 16-21. https://ieeexplore.ieee.org/abstract/document/8125550

[5] B. Burns, "What is a container?", azure.microsoft.com, 2019. https://azure.microsoft.com/en-us/overview/what-is-a-container/#overview.

[6] Brewer, E. A. (2015, August). Kubernetes and the path to cloud native. In Proceedings of the sixth ACM symposium on cloud computing (pp. 167-167). https://dl.acm.org/doi/abs/10.1145/2806777.2809955

[7] Peter M. Mell, Timothy Grance "The NIST Definition of Cloud Computing", Special Publication (NIST SP), 2011. https://www.nist.gov/publications/nist-definition-cloud-computing

[8] Google Cloud "Geografía y regiones", Información general de Google Cloud, 2020. https://cloud.google.com/docs/geography-and-regions

[9] Google Cloud "Productos", Compute Engine, 2020. https://cloud.google.com/compute?hl=es#section-5

[10] Google Cloud "Serverless computing", Documentación de Google App Engine, 2020. https://cloud.google.com/compute?hl=es#section-5

[11] Google Cloud "Kubernetes Engine", Descripción general de GKE, 2020. https://cloud.google.com/kubernetes-engine/docs/concepts/kubernetes-engine-overview?hl=es

[12] Google Cloud "Kubernetes Engine", Concepto de Pod en Kubernetes Engine, 2020. https://cloud.google.com/kubernetes-engine/docs/concepts/pod

[13] Google Cloud "Kubernetes Engine", Clústeres (Arquitectura de clúster), 2020. https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture

[14] Ben-Kiki, O., Evans, C., & Ingerson, B. (2009). Yaml ain't markup language. version 1.1. Working Draft 2008-05, 11.

[15] Man7.org. 2021. namespaces(7) - Linux manual page. [online] https://man7.org/linux/man-pages/man7/namespaces.7.html

[16] Buchanan, S., Rangama, J., & Bellavance, N. (2020). kubectl Overview. In Introducing Azure Kubernetes Service (pp. 51-62). Apress, Berkeley, CA.

[17] kubernetes.io,Documentación (Deployment), 2020. https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/

[18]Sayfan, G. (2019). Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes (pp 21). Packt Publishing Ltd. https://books.google.com.co/books

[19]B. Yang, A. Sailer, S. Jain, A. E. Tomala-Reyes, M. Singh and A. Ramnath, "Service Discovery Based Blue-Green Deployment Technique in Cloud Native Environments,"2018 IEEE International Conference on Services Computing (SCC), 2018, pp. 185-192, doi: 10.1109/SCC.2018.00031. https://ieeexplore.ieee.org/abstract/document/8456417

[20] kubernetes.io,Documentación (Deployment), 2020. https://kubernetes.io/es/docs/concepts/workloads/controllers/deployment/

[21]Galantino, S. (2020). Enabling Job-aware scheduling on Kubernetes clusters (Doctoral dissertation, Politecnico di Torino). https://webthesis.biblio.polito.it/15946/

[22] Google Cloud "Kubernetes Engine",Clústeres (Guias, Ejecutar un trabajo), 2020. https://cloud.google.com/kubernetes-engine/docs/how-to/jobs

[23] kubernetes.io, Documentación (Jobs), 2020. https://kubernetes.io/docs/concepts/workloads/controllers/job/

[24] Google Cloud "Google Kubernetes Engine (GKE)", Clústeres (Guias, Ejecutar un CronJob), 2020. https://cloud.google.com/kubernetes-engine/docs/how-to/cronjobs?hl=es-419

[25] Wang, M., Zhang, D., & Wu, B. (2020, June). A Cluster Autoscaler Based on Multiple Node Types in Kubernetes. In 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 1, pp. 575-579). IEEE. https://ieeexplore.ieee.org/abstract/document/9084706

[26] Liffredo, D. (2020). Analysis and Benchmarking of Kubernetes Networking (Doctoral dissertation, Politecnico di Torino). https://webthesis.biblio.polito.it/15948/