Universidad Nacional de San Agustín de Arequipa

Escuela Profesional de Ciencia de la Computación
Cloud Computing

IaC: Pulumi en Google Cloud Platform

Presentado por
Juan Manuel Soto Begazo

Docente
Alvaro Henry Mamani Aliaga

Descripción del proyecto

El presente trabajo describe el despliegue de un sistema de inventario MERN (MongoDB, Express, React y Node.js) mediante **Infraestructura como Código** en **Google Cloud Platform**. La infraestructura se define con Pulumi y se ejecuta sobre Google Kubernetes Engine (GKE). La aplicación permite registrar, listar y eliminar productos, mostrando su nombre, costo y cantidad disponible. Para asegurar una respuesta elástica ante variaciones de carga, se habilita el escalamiento automático tanto a nivel de aplicaciones (HPA) como a nivel de infraestructura (crecimiento del grupo de nodos del clúster).

El código fuente del proyecto, que incluye tanto la aplicación MERN como el programa de aprovisionamiento en Pulumi, se encuentra disponible en el siguiente repositorio de ***GitHub***.

Arquitectura en GCP e IaC

Previo al despliegue, se construyeron las imágenes de los dos componentes principales de la aplicación: el **backend** y el **frontend**. Ambas fueron compiladas localmente y publicadas en el **Google Container Registry (GCR)** del proyecto, quedando disponibles en la nube con las etiquetas correspondientes para su posterior uso por Kubernetes. A partir de este punto, dichas imágenes se consideran elementos ya preparados y estables, de modo que la atención se centra en la infraestructura y el aprovisionamiento del entorno donde serán ejecutadas.

La solución se modela como **infraestructura como código (IaC)** empleando Pulumi, lo que permite definir de manera declarativa y reproducible todos los recursos necesarios dentro de Google Cloud. A través de este enfoque, se automatiza tanto la creación del clúster de Kubernetes como la configuración de los servicios internos que dan soporte a la aplicación.

En términos generales, el código de aprovisionamiento realiza las siguientes tareas principales:

- Crea un **clúster de Google Kubernetes Engine (GKE)** en la zona seleccionada, con un plano de control administrado por Google y asociado a un canal de actualizaciones regular.
- Define un **grupo de nodos (Node Pool)** con instancias de tipo `e2-medium`, configuradas con disco estándar y capacidad de **autoescalado** entre uno y cuatro nodos según la demanda de recursos.
- Configura un **proveedor de Kubernetes** dentro de Pulumi, generando dinámicamente el archivo de credenciales (*kubeconfig*) para la conexión con el clúster recién creado.

- Despliega los tres **microservicios** del sistema: MongoDB, Backend y Frontend, cada uno con su respectivo conjunto de *pods*, *services* y políticas de recursos.

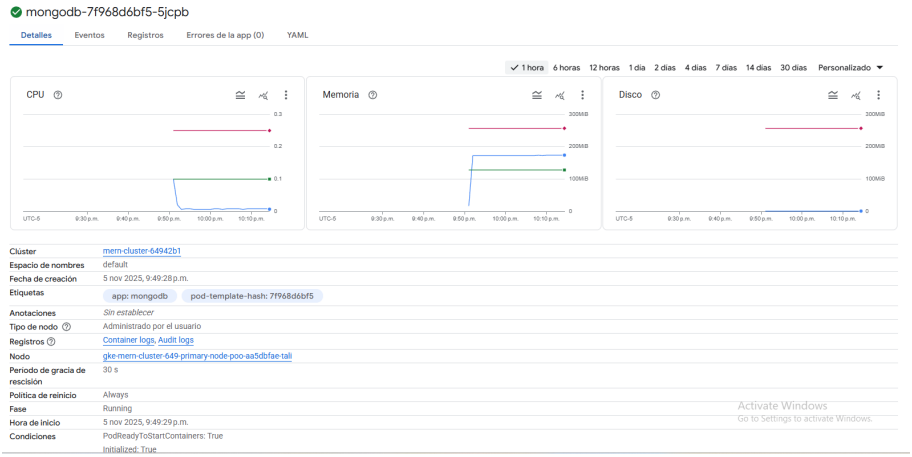

Figura 1: Consola de Google Cloud Platform - MongoBD

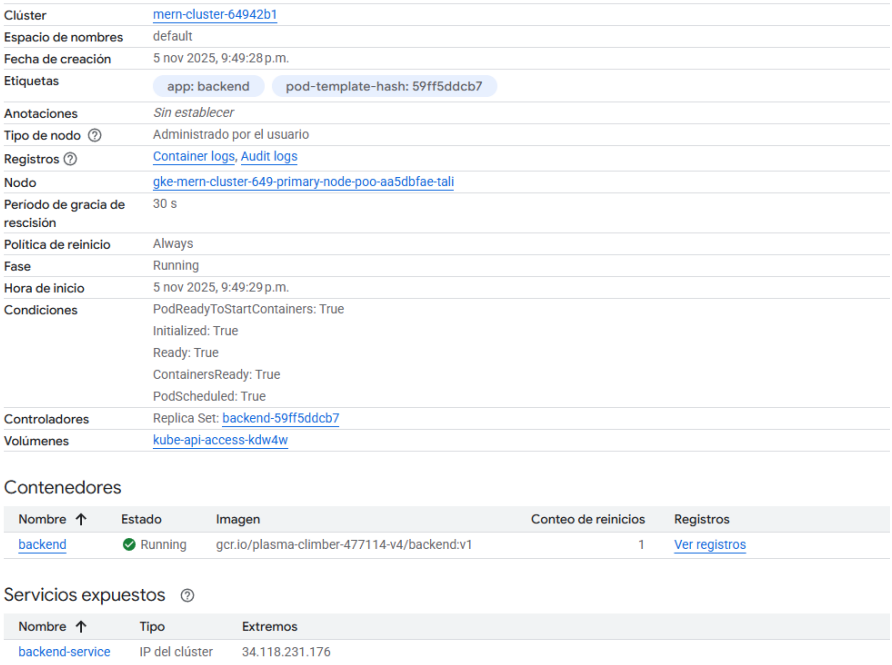

Figura 2: Consola de Google Cloud Platform - Backend

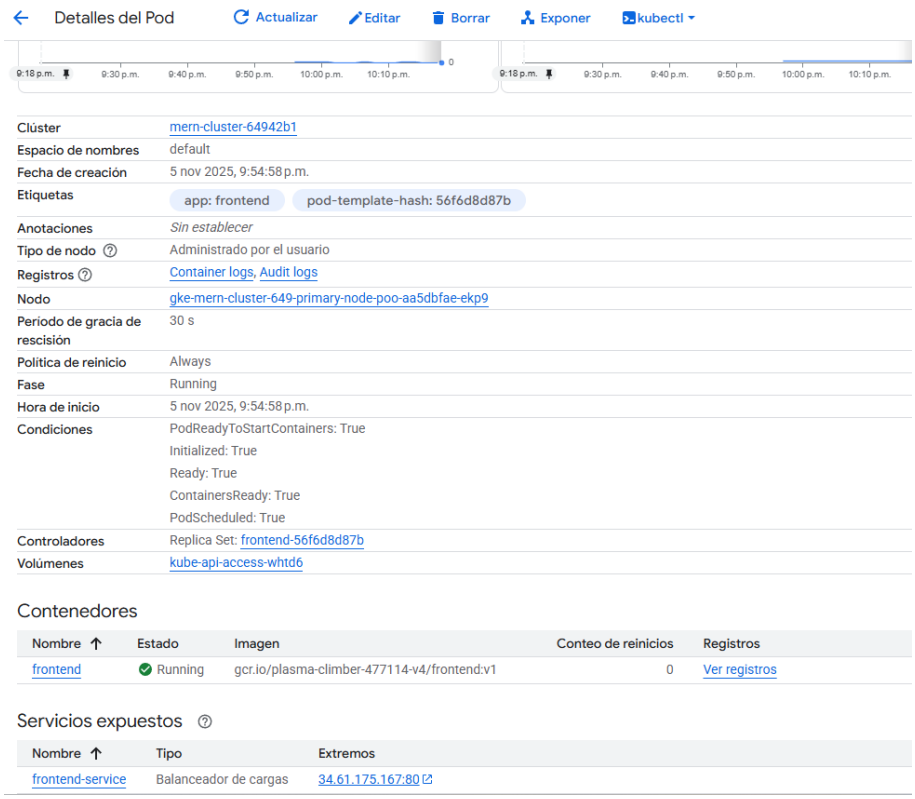

Figura 3: Consola de Google Cloud Platform - Frontend

- Asigna al **Frontend** un servicio de tipo *LoadBalancer* para exponer la aplicación de forma pública, mientras que el Backend y la base de datos operan mediante servicios internos.
- Implementa los **Horizontal Pod Autoscalers (HPA)** para el backend y el frontend, permitiendo que el número de réplicas varíe de forma automática según el uso de CPU y memoria.

Con ello, el despliegue completo se gestiona mediante un único programa de Pulumi, capaz de crear el clúster, provisionar los recursos y establecer las reglas de escalabilidad sin necesidad de editar archivos YAML manuales. Esta aproximación garantiza consistencia, trazabilidad y la posibilidad de replicar el entorno en cualquier momento o proyecto de GCP con tan solo ejecutar una actualización del código.

Componentes lógicos y comunicación

La aplicación se organiza en tres componentes principales que interactúan dentro del clúster de Kubernetes: el **frontend**, el **backend** y la **base de datos MongoDB**. Cada uno de ellos se despliega en un contenedor independiente, con su propio conjunto de recursos y políticas, lo que permite escalarlos o reiniciarlos sin afectar al resto del sistema.

El **frontend** corresponde a la interfaz web desarrollada en React, responsable de mostrar los productos del inventario y permitir la interacción con el usuario. Este componente es el único que se expone públicamente, mediante un servicio de tipo *LoadBalancer* que asigna una dirección IP externa. De este modo, cualquier usuario puede acceder a la aplicación desde un navegador web, mientras que la comunicación con los demás módulos se mantiene restringida al interior del clúster.

El **backend** implementa la API de inventario en Node.js con Express y actúa como intermediario entre el frontend y la base de datos. Su servicio es de tipo interno, accesible solo dentro del entorno de Kubernetes. Se configuraron **comprobaciones de salud** (*health probes*) y de disponibilidad (*readiness probes*) que permiten al orquestador verificar que el contenedor se encuentre en condiciones adecuadas antes de recibir tráfico. Estas comprobaciones contribuyen a mantener la estabilidad del sistema, especialmente durante despliegues o reinicios de pods.

Por su parte, la **base de datos MongoDB** se despliega como un contenedor independiente con persistencia lógica a nivel de pod. Opera de manera interna y únicamente responde a solicitudes provenientes del backend, evitando cualquier exposición directa a internet. Esta separación de capas permite reforzar la seguridad y la modularidad del sistema.

Para garantizar la correcta comunicación entre los servicios y comprobar el funcionamiento integral del despliegue, se realizaron las siguientes pruebas:

- **Verificación de despliegue**: se listaron los pods, servicios y réplicas para confirmar que cada componente estuviera en ejecución y con el estado `Running`.

```
kubectl get pods
NAME                        READY   STATUS    RESTARTS   AGE
backend-59ff5ddcb7-kgbw7    1/1     Running   0          41h
frontend-56f6d8d87b-vzttc   1/1     Running   0          41h
mongodb-7f968d6bf5-66stw    1/1     Running   0          41h
```

Figura 4: Pods en estado de Running

- **Prueba de despliegue del backend**: Se muestra la descripción del backend, así como su conexión con la base de datos en MongoDB.

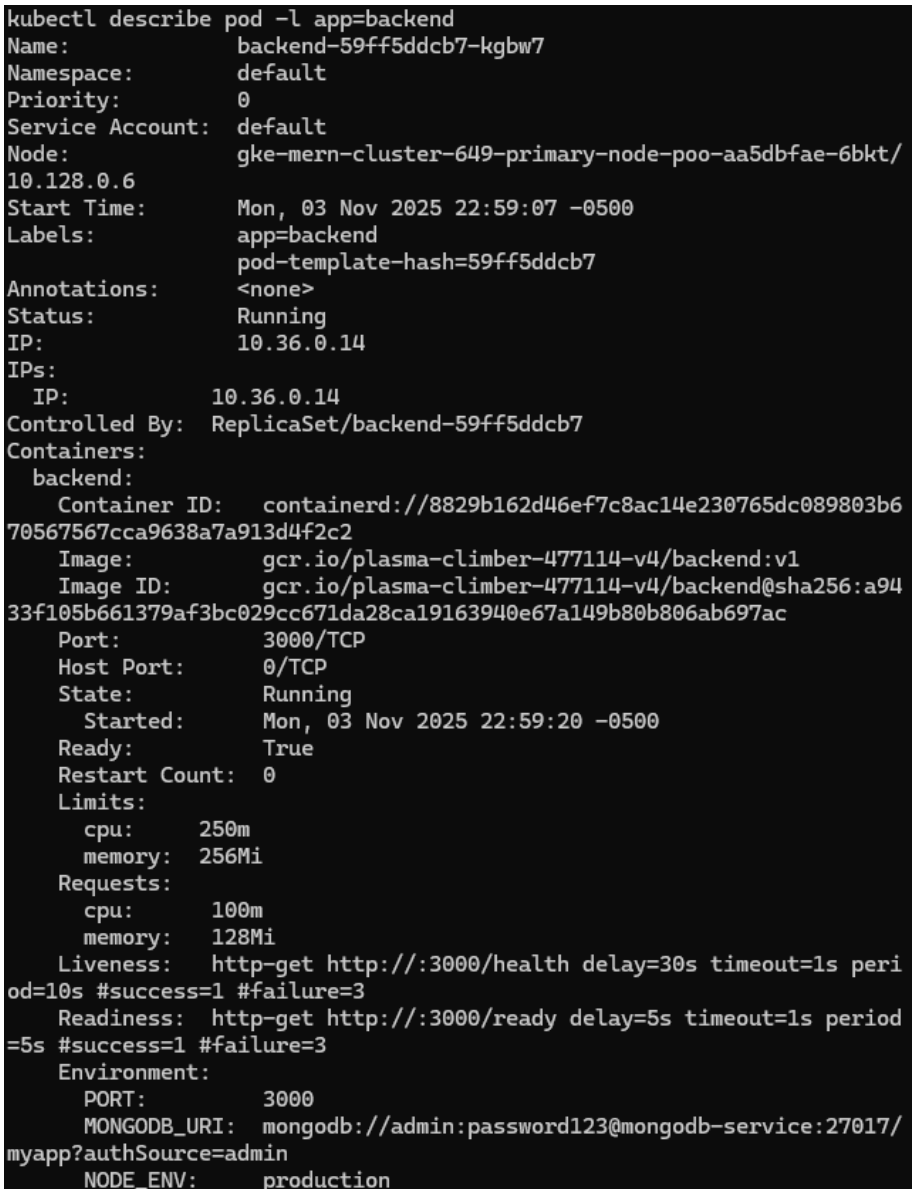

Figura 5: Descripción del backend

- **Comprobación de las probes del backend**: se consultaron los endpoints de salud y disponibilidad (`/health` y `/ready`) para confirmar que el orquestador respondía adecuadamente ante fallos simulados o reinicios de contenedores.

```
StatusCode        : 200
StatusDescription : OK
Content           : {"status":"ok","timestamp":"2025-11-05T21:48:37.067Z"}
RawContent        : HTTP/1.1 200 OK
                    Access-Control-Allow-Origin: *
                    Connection: keep-alive
                    Keep-Alive: timeout=5
                    Content-Length: 54
                    Content-Type: application/json; charset=utf-8
                    Date: Wed, 05 Nov 2025 21:48:37 GMT
                    ...
Forms             : {}
Headers           : {[Access-Control-Allow-Origin, *], [Connection, keep-alive], [Keep-Alive, timeout=
5],
                    [Content-Length, 54]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 54
```

Figura 6: Healty Probe

```
StatusCode        : 200
StatusDescription : OK
Content           : ready
RawContent        : HTTP/1.1 200 OK
                    Access-Control-Allow-Origin: *
                    Connection: keep-alive
                    Keep-Alive: timeout=5
                    Content-Length: 5
                    Content-Type: text/html; charset=utf-8
                    Date: Wed, 05 Nov 2025 21:48:52 GMT
                    ETag: W/...
Forms             : {}
Headers           : {[Access-Control-Allow-Origin, *], [Connection, keep-alive], [Keep-Alive,
                    timeout=5], [Content-Length, 5]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 5
```

Figura 7: Ready Probe

- **Consulta de logs**: se inspeccionaron los registros del backend, frontend y MongoDB para asegurar que no existieran errores de conexión, autenticación ni puertos bloqueados.
- **Validación funcional desde el frontend**: se ingresó a la dirección IP pública.

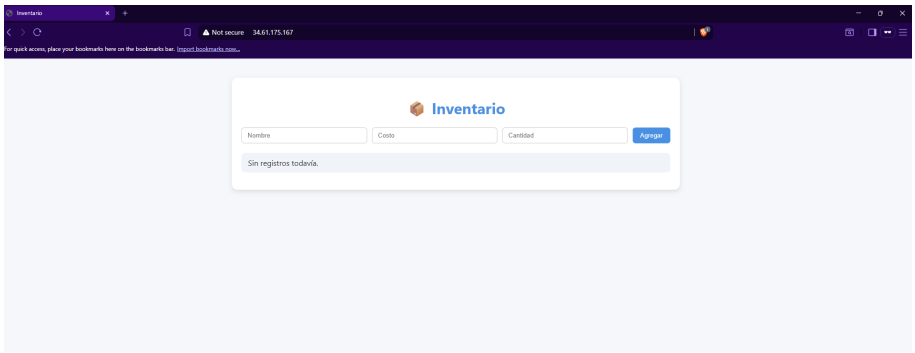

Figura 8: Frontend en la IP pública

- **Pruebas de Funcionamiento**: Se hicieron varias pruebas de inserción, listado y eliminación de elementos:

Figura 9: Prueba de Insertado

Figura 10: Objeto Insertado

Con estas pruebas se validó que los tres componentes del sistema se encuentran correctamente integrados, que la comunicación entre ellos fluye mediante los servicios internos definidos, y que la aplicación es accesible de forma estable desde el exterior a través del balanceador de carga. El entorno demostró una operación continua, con detección automática de fallos y recuperación inmediata ante interrupciones, confirmando la solidez del despliegue realizado.

Escalabilidad, resiliencia y validación de carga

La arquitectura del sistema fue diseñada con un enfoque centrado en la **elasticidad** y la **resiliencia**, garantizando que la aplicación pueda adaptarse automáticamente a las variaciones de demanda sin intervención manual. Esta capacidad se implementa en dos planos complementarios: el de las aplicaciones y el de la infraestructura.

En el primer plano, se configuraron **Horizontal Pod Autoscalers (HPA)** para los servicios del *backend* y el *frontend*. Estos mecanismos monitorean de forma continua el uso de recursos y ajustan dinámicamente el número de réplicas activas de cada servicio.

En el caso del backend, el HPA evalúa tanto el consumo de CPU como la utilización de memoria, pudiendo escalar entre una y diez réplicas según la carga. Para el frontend, el escalado se basa en el uso de CPU, permitiendo crecer hasta ocho réplicas en momentos de alta concurrencia de usuarios.

En el segundo plano, correspondiente a la infraestructura, se habilitó el **autoescalador del grupo de nodos (Node Pool Autoscaling)** de GKE, el cual permite aumentar o reducir la cantidad de instancias físicas del clúster dentro de un rango definido (de uno a cuatro nodos). Esto asegura que, cuando los HPA demanden más pods de los que caben en los nodos existentes, el clúster pueda expandirse automáticamente para alojarlos, y posteriormente liberar recursos al disminuir la carga.

Para verificar el funcionamiento de ambos mecanismos, se diseñó una **prueba de carga controlada** dirigida al backend, componente que concentra la mayor demanda de procesamiento. Se empleó un script de estrés que enviaba múltiples solicitudes HTTP de forma simultánea, simulando un escenario de alta concurrencia. A medida que el consumo de CPU y memoria se incrementaba, el HPA detectó el sobreuso de recursos y activó nuevas réplicas del backend, distribuyendo la carga de manera uniforme entre ellas.

Durante todo el experimento, el **frontend** permaneció estable y accesible desde la IP pública del balanceador, mientras que la **base de datos MongoDB** continuó respondiendo sin errores, demostrando la estabilidad de la comunicación interna entre servicios. El monitoreo en tiempo real permitió observar cómo, en paralelo, el **Node Pool** incrementaba temporalmente el número de nodos para sostener las nuevas réplicas, evidenciando la coordinación entre los dos niveles de escalamiento.

Al finalizar la prueba y reducir la carga, tanto las réplicas de los pods como las instancias del clúster regresaron progresivamente a su número inicial, confirmando el correcto funcionamiento del proceso inverso de *desescalado*. Este comportamiento comprobó que el sistema no solo puede crecer ante aumentos de tráfico, sino también optimizar el consumo de recursos al disminuir la demanda, manteniendo así un balance eficiente entre desempeño y costo operativo.

- Se inició una carga controlada para hacer que el **Backend Service** escale.
- En la siguiente figura se observan las métricas capturadas por el HPA durante la prueba, donde se aprecia el aumento progresivo del consumo de CPU, hasta superar la metrica de 30 % y posteriormente a eso la creación de una nueva replica.

```
C:\Universidad\7mo-Año\Semestre B\Cloud\cloud-computing\02-Kubernetes>kubectl get hpa backend-hpa -w
NAME          REFERENCE            TARGETS                        MINPODS   MAXPODS   REPLICAS   AGE
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 26%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 26%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 26%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 26%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 27%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 28%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 28%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 28%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 28%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 1%/30%, memory: 28%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 16%/30%, memory: 31%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 50%/30%, memory: 31%/70%   1         10        1          47h
backend-hpa   Deployment/backend   cpu: 50%/30%, memory: 31%/70%   1         10        2          47h
backend-hpa   Deployment/backend   cpu: 50%/30%, memory: 36%/70%   1         10        2          47h
backend-hpa   Deployment/backend   cpu: 50%/30%, memory: 37%/70%   1         10        2          47h
```

Figura 11: Aumento de parámetros en el HPA

- En la figura a continuación se evidencia el estado de los pods luego del escalamiento, con varias instancias del backend activas simultáneamente.

```
C:\Universidad\7mo-Año\Semestre B\Cloud\cloud-computing\02-Kubernetes>kubectl get pods -l app=backend
-w
NAME                         READY   STATUS    RESTARTS       AGE
backend-59ff5ddcb7-5m8gt     1/1     Running   1 (14m ago)    15m
backend-59ff5ddcb7-d2kz9     0/1     Pending   0              0s
backend-59ff5ddcb7-d2kz9     0/1     Pending   0              0s
backend-59ff5ddcb7-d2kz9     0/1     ContainerCreating   0              0s
backend-59ff5ddcb7-d2kz9     0/1     Running             0              10s
backend-59ff5ddcb7-d2kz9     1/1     Running             0              17s
```

Figura 12: Escalamiento de Pods

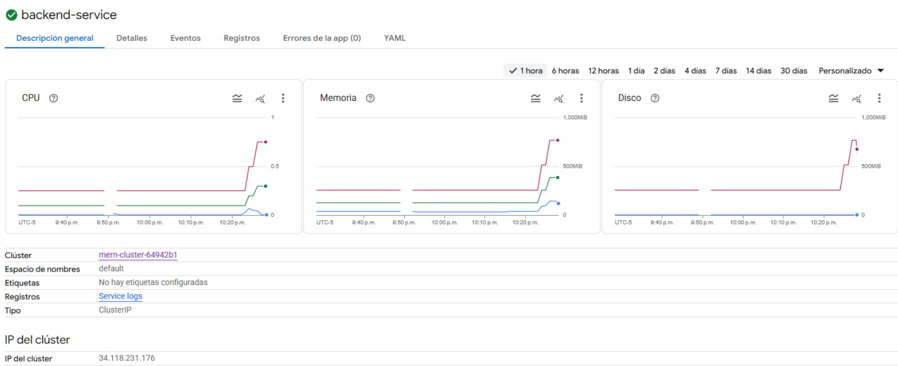

Figura 13: Incremento en las métricas de Backend Service

Implementaciones

Nombre	Estado	Pods
backend	OK	3/3

Pods activos

Nombre	Estado	Extremos	Reinicios	Fecha de creación ↑
backend-59ff5ddcb7-5m8gt	Running	10.36.2.4	1	5 nov 2025, 9:49:28 p.m.
backend-59ff5ddcb7-d2kz9	Running	10.36.0.10	0	5 nov 2025, 10:22:50 p.m.
backend-59ff5ddcb7-r562x	Running	10.36.2.12	0	5 nov 2025, 10:25:16 p.m.

Figura 14: Pod Activos en Google Cloud Platform

En conjunto, los resultados confirmaron que la arquitectura desplegada es **autosuficiente, elástica y resiliente**, capaz de responder de manera dinámica a las fluctuaciones de tráfico sin degradación perceptible en el rendimiento. La integración entre Kubernetes y los servicios gestionados de GKE permitió alcanzar un sistema flexible, eficiente y preparado para entornos productivos basados en los principios de **Cloud Computing**.

Conclusiones

La migración del laboratorio local a un enfoque de **IaC en GCP** permitió automatizar la infraestructura, reducir la configuración manual y mejorar la trazabilidad de cambios. La combinación de **HPA** en las aplicaciones y **autoscaling** del grupo de nodos demostró capacidad de adaptación ante picos de tráfico, manteniendo la disponibilidad. En síntesis, el uso de Pulumi y GKE facilitó un despliegue moderno, escalable y verificable, alineado con los principios de **Cloud Computing**.