Lab GSP041 - Configuración de un Balanceador de Cargas de Aplicaciones Interno

Este documento detalla el trabajo realizado, los conceptos aprendidos y la arquitectura implementada durante el lab de Google Cloud Skills Boost "GSP041: Configurar un Balanceador de Cargas de Aplicaciones Interno", como parte de mi preparación para la certificación de Associate Cloud Engineer.

1. ¿Qué es un Balanceador de Cargas de Aplicaciones Interno?

Un Balanceador de Cargas de Aplicaciones (ALB) Interno de Google Cloud es un servicio administrado (basado en el proxy de Envoy) que opera en la capa 7 (HTTP/S) del modelo OSI. Su función principal es distribuir el tráfico de red *dentro* de tu red de Nube Privada Virtual (VPC).

A diferencia de un balanceador externo que expone servicios a Internet, un balanceador interno solo es accesible a través de una dirección IP interna de tu VPC.

2. ¿Para qué sirve y cuál es su objetivo?

Objetivo Principal: El propósito de un ALB Interno es proporcionar una arquitectura de microservicios segura, escalable y de alta disponibilidad.

Casos de uso clave:

- **Seguridad:** Permite que los servicios de "backend" (como bases de datos, APIs de procesamiento, o microservicios) permanezcan completamente privados, sin direcciones IP públicas. El único punto de acceso es la IP interna del balanceador.
- **Alta Disponibilidad:** Se integra con Grupos de Instancias Administrados (MIGs) y realiza verificaciones de estado. Si una VM (máquina virtual) de backend falla, el balanceador deja de enviarle tráfico automáticamente, redirigiéndolo a las instancias saludables.
- **Escalabilidad:** A medida que el MIG escala horizontalmente (agrega más VMs por alta demanda), el balanceador las registra automáticamente y distribuye la carga entre ellas.

- **Abstracción:** Las aplicaciones "frontend" no necesitan saber las direcciones IP individuales de las VMs del backend. Solo se comunican con una única dirección IP virtual (VIP) estática: la del balanceador.

3. Objetivo del Lab

El objetivo de este lab fue construir una arquitectura de aplicación de dos niveles (two-tier) muy común:

1. **Nivel Frontend (Público):** Un servidor web expuesto a Internet que actúa como la cara pública de la aplicación.
2. **Nivel Backend (Privado):** Un microservicio interno (una "calculadora de números primos") que realiza el trabajo pesado.
3. **Conexión:** El frontend se comunica con el backend de forma segura y escalable utilizando un Balanceador de Cargas de Aplicaciones Interno.

4. Arquitectura de la Solución y Detalles Técnicos

Implementamos la siguiente arquitectura para lograr el objetivo:

- **VPC:** Se utilizó la red `default`.
- **Nivel Frontend:**
  - **Instancia:** Una única VM de Compute Engine llamada `frontend`.
  - **Acceso:** Se le asignó una IP externa y una etiqueta (`frontend`) para una regla de firewall que permite el tráfico HTTP (puerto 80) desde *cualquier lugar* (`0.0.0.0/0`).
  - **Lógica:** Ejecutaba un script de Python (`frontend.sh`) que actúa como servidor web. Este script recibe solicitudes del usuario, contacta al backend a través de la IP del balanceador interno, y formatea la respuesta en HTML.
- **Nivel Backend:**
  - **Plantilla de Instancias:** Se creó una plantilla (`primecalc`) con la instrucción de *no* asignar IPs externas (`--no-address`).
  - **Grupo de Instancias Administrado (MIG):** Se creó un MIG llamado `backend` basado en la plantilla, desplegando 3 instancias. Esto garantiza la alta disponibilidad.

    - **Lógica:** Cada instancia ejecutaba un script de Python (`backend.sh`) que respondía `True` o `False` a solicitudes sobre si un número era primo.
    - **Acceso:** Se protegió con la etiqueta `backend` para una regla de firewall que *solo* permite tráfico interno (desde el balanceador y las verificaciones de estado) en el puerto 80.
- **Balanceador de Cargas Interno:**
    - **Verificación de Estado (`ilb-health`):** Una comprobación HTTP que el balanceador usa para preguntar a cada VM de backend si está "viva" antes de enviarle tráfico.
    - **Servicio de Backend (`prime-service`):** El componente que define *cómo* se balancea el tráfico. Se vinculó a nuestro MIG `backend` y a la verificación de estado `ilb-health` . Se configuró con `--load-balancing-scheme internal` .
    - **Regla de Reenvío (`prime-lb`):** Este es el componente que crea la **dirección IP interna estática** (ej. `10.138.0.10`). Toda solicitud a esta IP en el puerto 80 se "reenvía" al `prime-service` .

> T65P3C60nkaqN_RQOvlfQaZp8TByJRKw_jVISJW9yMM=.png

5. Resumen de Tareas: ¿Qué hice exactamente?

El lab se ejecutó en 5 tareas principales:

Tarea 1 y 2: Creación del Backend Privado

Creé el servicio de "calculadora de números primos".

1. **Script de Inicio (`backend.sh`):** Definí un script de Python que inicia un servidor web simple en el puerto 80.
2. **Plantilla de Instancias (`primecalc`):** Creé la plantilla para las VMs de backend. La configuré con `--no-address` para asegurar que no tuvieran IP pública y le asigné la etiqueta `backend` para el firewall.
3. **Regla de Firewall (`http`):** Creé una regla de firewall para permitir el tráfico HTTP (tcp:80) *solo* desde rangos de IP internos (en este lab, `10.138.0.0/20`) y dirigido a las instancias con la etiqueta `backend` .
4. **Grupo de Instancias Administrado (`backend`):** Creé el MIG que automáticamente desplegó 3 VMs basadas en la plantilla.

> 

Tarea 3: Configuración del Balanceador de Cargas Interno

Este fue el núcleo del lab. Conecté los componentes del balanceador.

1. **Verificación de Estado (`ilb-health`):** Creé una verificación de estado HTTP.
2. **Servicio de Backend (`prime-service`):** Creé el servicio de backend, especificando el esquema `internal` y vinculándolo a la verificación de estado.
3. **Vinculación:** Agregué mi MIG `backend` al `prime-service`.
4. **Regla de Reenvío (`prime-lb`):** Creé la regla de reenvío, que generó la IP interna estática (`10.138.0.10`) para el balanceador.

> 

Tarea 4: Prueba del Backend y el Balanceador

Antes de construir el frontend, verifiqué que el servicio interno funcionaba.

1. **Creé una VM de prueba (`testinstance`)** dentro de la misma VPC.
2. **Me conecté por SSH** a esta instancia.
3. **Usé `curl`** para hacer solicitudes directamente a la IP del balanceador (`curl 10.138.0.10/2`, `curl 10.138.0.10/4`).
4. **Resultado:** Recibí las respuestas correctas (`True`, `False`), confirmando que el balanceador estaba distribuyendo el tráfico a mi backend privado.

> 

Tarea 5: Creación del Frontend Público

Finalmente, creé el servidor web público que consumiría el servicio interno.

1. **Script de Inicio (`frontend.sh`):** Escribí un script de Python que *hardcodea* la IP del balanceador (`http://10.138.0.10/`) para hacer sus consultas.
2. **Instancia (`frontend`):** Creé la VM `frontend`, esta vez *con* una IP pública y la etiqueta `frontend`.

3. **Regla de Firewall (`http2`):** Creé una segunda regla de firewall para permitir el tráfico HTTP (tcp:80) desde *Internet* (`0.0.0.0/0`) a la etiqueta `frontend` .
4. **Prueba Final:** Accedí a la IP pública del `frontend` desde mi navegador y vi la tabla de números primos, demostrando que el flujo completo (Navegador -> Frontend (Público) -> ALB Interno (Privado) -> Backend (Privado)) estaba funcionando.

> 
>
> 
>
> 

6. Análisis de Comandos `gcloud` Clave

- `gcloud compute instance-templates create primecalc ... --no-address --tags backend`
  - `--no-address` : Flag de seguridad crítico. Asegura que las VMs del backend nunca sean accesibles desde Internet.
  - `--tags backend` : Etiqueta fundamental para vincular la VM a la regla de firewall del backend.
- `gcloud compute firewall-rules create http ... --source-ranges 10.138.0.0/20 --target-tags backend`
  - `--source-ranges` : Restringe el tráfico. En un escenario real, aquí se incluirían los rangos de IP de las subredes de tu aplicación y los rangos de las verificaciones de estado de Google (`130.211.0.0/22` , `35.191.0.0/16`).
  - `--target-tags` : Aplica la regla solo a las VMs con esta etiqueta.
- `gcloud compute backend-services create prime-service ... --load-balancing-scheme internal --region=...`
  - `--load-balancing-scheme internal` : El comando más importante. Define este servicio de backend como interno para la VPC, en lugar de externo.
- `gcloud compute forwarding-rules create prime-lb ... --load-balancing-scheme internal --address 10.138.0.10`
  - `--load-balancing-scheme internal` : Nuevamente, especifica que esta es una IP interna.

    - `--address`: Asigna la IP estática que servirá como el punto de entrada (VIP) para todos los servicios internos.

- `gcloud compute firewall-rules create http2 ... --source-ranges 0.0.0.0/0 --target-tags frontend`
    - `--source-ranges 0.0.0.0/0`: El "0.0.0.0/0" significa "cualquier dirección IP", es decir, abre el puerto 80 a todo Internet.
    - `--target-tags frontend`: Aplica esta regla solo a nuestra VM pública.

7. Logros y Conclusión

¿Qué logré? Completé con éxito la implementación de una arquitectura de aplicación de dos niveles, segura y con alta disponibilidad. Demostré habilidades en:

- **Google Compute Engine:** Creación y gestión de VMs, plantillas de instancias y grupos de instancias administrados (MIGs).
- **Redes VPC:** Creación y gestión de reglas de firewall de VPC, comprensión del direccionamiento IP interno vs. externo y el uso de etiquetas.
- **Balanceo de Carga:** Configuración completa de un Balanceador de Cargas de Aplicaciones Interno, incluyendo verificaciones de estado, servicios de backend y reglas de reenvío.

Conclusión: Este lab es una demostración práctica de un patrón de arquitectura fundamental para la seguridad en la nube. Al aislar el backend de Internet, se reduce drásticamente la superficie de ataque. El balanceador de cargas interno actúa como la "puerta de entrada" controlada y escalable a los microservicios, un concepto clave para la certificación de Associate Cloud Engineer y para el diseño de cualquier aplicación moderna y robusta en GCP.