Informe Completo - Balanceo de Servicios con Terraform, Docker y HAProxy

Alumno: Antonio Marbán Regalado // Gabriel Gonzalez

Fecha: 9 de noviembre de 2025

Materia: Gestión de Infraestructura y Sistemas

---

1. Objetivo

El objetivo de la práctica es diseñar, desplegar y verificar el funcionamiento de una arquitectura de servicios balanceados utilizando Terraform como herramienta de infraestructura como código y Docker como entorno de despliegue. Se implementaron dos escenarios:

- **Escenario 1:** Balanceo HTTP entre dos aplicaciones Java Spring Boot.
- **Escenario 2:** Balanceo TCP entre dos servicios Java basados en sockets.

El balanceo de carga se realiza mediante un contenedor HAProxy configurado en modo HTTP o TCP según el escenario.

2. Arquitectura Implementada

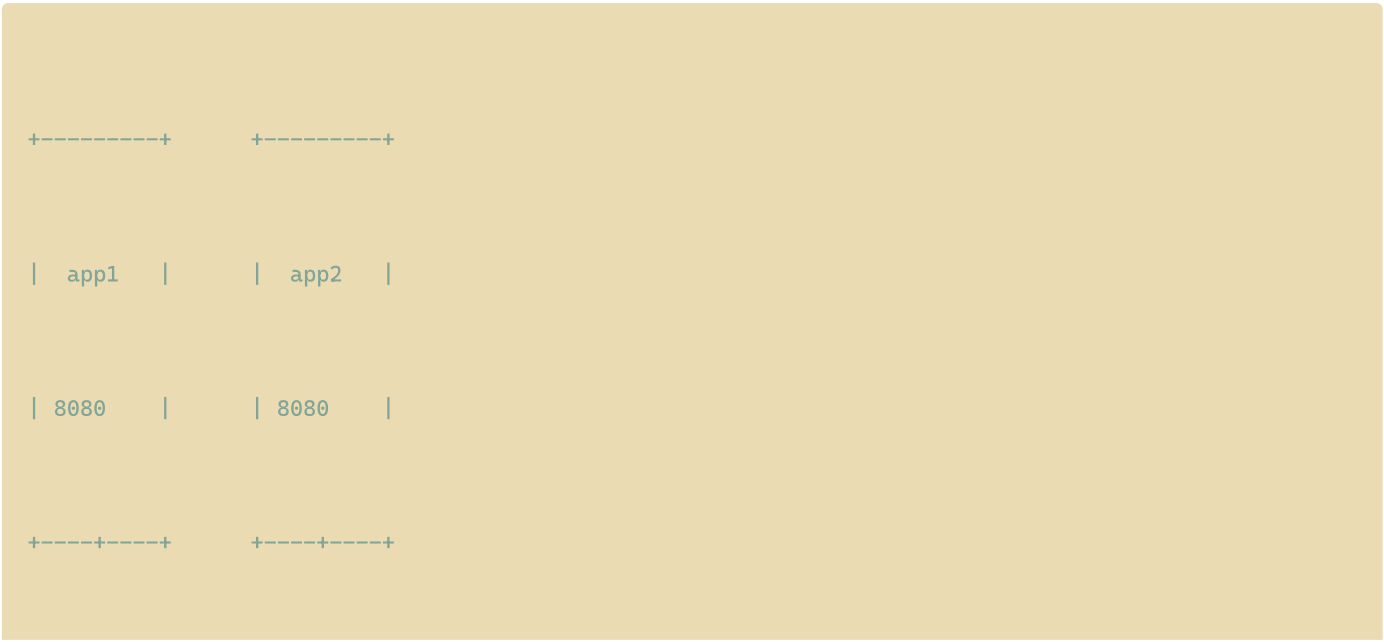
Se construyeron dos redes de contenedores independientes:

- `http_network`: utilizada por las aplicaciones HTTP (app1, app2 y HAProxy).
- `tcp_network`: utilizada por los servicios TCP (srv1, srv2 y HAProxy).

Cada escenario consta de tres contenedores: dos servicios backend y un balanceador HAProxy. Terraform se encargó de levantar las imágenes Docker, crear los contenedores y las redes necesarias.

Diagrama conceptual:

Escenario 1 (HTTP)



```
    \              /

     \            /

      \          /

       +---------+

       | HAProxy |

       |  Port80 |

       +---------+
```

Escenario 2 (TCP)

```
+---------+      +---------+

|  srv1   |      |  srv2   |

| 5000    |      | 5000    |

+----+----+      +----+----+

     \              /

      \            /
```

```
      \          /

       +---------+

       | HAProxy |

       | Port4000|

       +---------+
```

3. Escenario 1: Balanceo HTTP de aplicaciones Spring Boot

3.1 Configuración técnica

- Aplicaciones Java: `app1` y `app2` escuchan en puerto 8080.
- HAProxy: escucha en puerto 80, modo HTTP, algoritmo round-robin.
- Terraform: crea la red `http_network`, construye las imágenes `app1`, `app2` y `haproxy_http`, y levanta los contenedores.

Fragmento HAProxy.cfg:

```
frontend http_front

    bind *:80

    default_backend http_back


backend http_back

    balance roundrobin
```

```
    server app1 app1:8080 check

    server app2 app2:8080 check
```

Dockerfiles: basados en `openjdk:17-jdk-slim`, exponen puerto 8080 y ejecutan el jar de Spring Boot.

3.2 Verificación y resultados

Pruebas con `curl` sobre `http://localhost:80/hello`:

| Ejecución | Respuesta |

| --- | --- |

| 1 | Hola desde app1 |

| 2 | Hola desde app2 |

| 3 | Hola desde app1 |

| 4 | Hola desde app2 |

| 5 | Hola desde app1 |

Observaciones:

- El balanceo HTTP alterna correctamente entre las dos aplicaciones, demostrando el algoritmo round-robin.
- Si se detiene un contenedor, HAProxy sigue respondiendo con el otro servidor disponible, garantizando alta disponibilidad.

4. Escenario 2: Balanceo TCP con servicios Java por socket

4.1 Configuración técnica

- Servicios Java: `srv1` y `srv2` escuchan en puerto 5000 mediante sockets TCP.
- Protocolo: el cliente envía "HELLO" y el servidor responde "OLLEH SRV1" o "OLLEH SRV2" según el backend.
- HAProxy: escucha en puerto 4000, modo TCP, algoritmo round-robin.
- Terraform: crea la red `tcp_network`, construye las imágenes `srv1`, `srv2` y `haproxy_tcp`, y levanta los contenedores.

Fragmento HAProxy.cfg:

```
frontend tcp_front
```

```
    bind *:4000

    default_backend tcp_back



backend tcp_back

    balance roundrobin

    server srv1 srv1:5000 check

    server srv2 srv2:5000 check
```

4.2 Verificación y resultados

Pruebas con Python:

```
import socket

s = socket.socket()

s.connect(('localhost', 4000))

s.sendall(b'HELLO\n')

print(s.recv(1024).decode())

s.close()
```

| Ejecución | Respuesta |

| --------- | ---------- |

| 1 | OLLEH SRV2 |

| 2 | OLLEH SRV1 |

| 3 | OLLEH SRV2 |

| 4 | OLLEH SRV1 |

| 5 | OLLEH SRV2 |

Observaciones:

- El balanceo TCP alterna conexiones entre `srv1` y `srv2`.
- Cada conexión es persistente durante la ejecución del cliente, a diferencia de HTTP, donde cada petición es independiente.
- Si un servidor se detiene, nuevas conexiones se dirigen automáticamente al servidor activo.

5. Comparación entre balanceo HTTP y TCP

| Característica | HTTP (Escenario 1) | TCP (Escenario 2) |

| ---------- | ---------- | ---------- |

| Modo HAProxy | http | tcp |

| Puerto HAProxy | 80 | 4000 |

| Persistencia | Cada petición independiente | Conexión persistente |

| Algoritmo | Round-robin | Round-robin |

| Comportamiento ante fallo | Continúa respondiendo con servidor activo | Nuevas conexiones al servidor activo, conexiones existentes se interrumpen |

| Pruebas | `curl` alterna respuestas | Python socket alterna conexiones |

6. Gestión de contenedores con Terraform

Terraform permitió:

- Crear redes Docker aisladas (`docker_network`).
- Construir imágenes desde los Dockerfiles (`docker_image`).
- Levantar contenedores vinculados a las redes y con puertos expuestos (`docker_container`).
- Gestionar dependencias: HAProxy depende de los servicios backend.

Comandos usados:

```
terraform init

terraform apply -auto-approve
```

```
terraform destroy
```

7. Conclusiones

- Se logró desplegar con éxito dos escenarios de balanceo de carga usando Docker, HAProxy y Terraform.
- El balanceo HTTP distribuye peticiones individuales de forma round-robin, mientras que el TCP distribuye conexiones persistentes.
- HAProxy garantiza alta disponibilidad: si un backend falla, el tráfico se redirige automáticamente al servidor activo.
- Terraform simplifica la creación de infraestructuras reproducibles y gestionables como código.

8. Output

Ejecución de `terraform apply` (Escenario 1).

```
PS C:\Users\anton\Desktop\plantilla_practica\escenario1> terraform apply -auto-approve


Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
  + create


Terraform will perform the following actions:


  # docker_container.app1 will be created
  + resource "docker_container" "app1" {
      + attach                                      = false
      + bridge                                      = (known after apply)
      + command                                     = (known after apply)
      + container_logs                              = (known after apply)
      + container_read_refresh_timeout_milliseconds = 15000
      + entrypoint                                  = (known after apply)
      + env                                         = (known after apply)
      + exit_code                                   = (known after apply)
      + hostname                                    = (known after apply)
      + id                                          = (known after apply)
      + image                                       = "app1"
      + init                                        = (known after apply)
```

```
      + ipc_mode                                = (known after apply)
      + log_driver                              = (known after apply)
      + logs                                    = false
      + must_run                                = true
      + name                                    = "app1"
      + network_data                            = (known after apply)
      + network_mode                            = "bridge"
      + read_only                               = false
      + remove_volumes                          = true
      + restart                                 = "no"
      + rm                                      = false
      + runtime                                 = (known after apply)
      + security_opts                           = (known after apply)
      + shm_size                                = (known after apply)
      + start                                   = true
      + stdin_open                              = false
      + stop_signal                             = (known after apply)
      + stop_timeout                            = (known after apply)
      + tty                                     = false
      + wait                                    = false
      + wait_timeout                            = 60

      + healthcheck (known after apply)

      + labels (known after apply)

      + networks_advanced {
          + aliases      = []
          + name         = "http_network"
            # (2 unchanged attributes hidden)
        }

      + ports {
          + external = (known after apply)
```

```
      + internal = 8080
      + ip       = "0.0.0.0"
      + protocol = "tcp"
    }
}

# docker_container.app2 will be created
+ resource "docker_container" "app2" {
    + attach                                      = false
    + bridge                                      = (known after apply)
    + command                                     = (known after apply)
    + container_logs                              = (known after apply)
    + container_read_refresh_timeout_milliseconds = 15000
    + entrypoint                                  = (known after apply)
    + env                                         = (known after apply)
    + exit_code                                   = (known after apply)
    + hostname                                    = (known after apply)
    + id                                          = (known after apply)
    + image                                       = "app2"
    + init                                        = (known after apply)
    + ipc_mode                                    = (known after apply)
    + log_driver                                  = (known after apply)
    + logs                                        = false
    + must_run                                    = true
    + name                                        = "app2"
    + network_data                                = (known after apply)
    + network_mode                                = "bridge"
    + read_only                                   = false
    + remove_volumes                              = true
    + restart                                     = "no"
    + rm                                          = false
    + runtime                                     = (known after apply)
    + security_opts                               = (known after apply)
```

```
      + shm_size                                     = (known after apply)
      + start                                        = true
      + stdin_open                                   = false
      + stop_signal                                  = (known after apply)
      + stop_timeout                                 = (known after apply)
      + tty                                          = false
      + wait                                         = false
      + wait_timeout                                 = 60

      + healthcheck (known after apply)

      + labels (known after apply)

      + networks_advanced {
          + aliases      = []
          + name         = "http_network"
            # (2 unchanged attributes hidden)
        }

      + ports {
          + external = (known after apply)
          + internal = 8080
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_container.haproxy will be created
  + resource "docker_container" "haproxy" {
      + attach                                       = false
      + bridge                                       = (known after apply)
      + command                                      = (known after apply)
      + container_logs                               = (known after apply)
      + container_read_refresh_timeout_milliseconds = 15000
```

```
+ entrypoint                                = (known after apply)
+ env                                       = (known after apply)
+ exit_code                                 = (known after apply)
+ hostname                                  = (known after apply)
+ id                                        = (known after apply)
+ image                                     = "haproxy_http"
+ init                                      = (known after apply)
+ ipc_mode                                  = (known after apply)
+ log_driver                                = (known after apply)
+ logs                                      = false
+ must_run                                  = true
+ name                                      = "haproxy_http"
+ network_data                              = (known after apply)
+ network_mode                              = "bridge"
+ read_only                                 = false
+ remove_volumes                            = true
+ restart                                   = "no"
+ rm                                        = false
+ runtime                                   = (known after apply)
+ security_opts                             = (known after apply)
+ shm_size                                  = (known after apply)
+ start                                     = true
+ stdin_open                                = false
+ stop_signal                               = (known after apply)
+ stop_timeout                              = (known after apply)
+ tty                                       = false
+ wait                                      = false
+ wait_timeout                              = 60

+ healthcheck (known after apply)

+ labels (known after apply)
```

```
      + networks_advanced {
          + aliases      = []
          + name         = "http_network"
            # (2 unchanged attributes hidden)
        }

      + ports {
          + external = 80
          + internal = 80
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_image.app1 will be created
  + resource "docker_image" "app1" {
      + id          = (known after apply)
      + image_id    = (known after apply)
      + name        = "app1"
      + repo_digest = (known after apply)

      + build {
          + cache_from     = []
          + context        = "./app1"
          + dockerfile     = "Dockerfile"
          + extra_hosts    = []
          + remove         = true
          + security_opt   = []
          + tag            = []
            # (13 unchanged attributes hidden)
        }
    }

  # docker_image.app2 will be created
```

```
+ resource "docker_image" "app2" {
    + id          = (known after apply)
    + image_id    = (known after apply)
    + name        = "app2"
    + repo_digest = (known after apply)

    + build {
        + cache_from     = []
        + context        = "./app2"
        + dockerfile     = "Dockerfile"
        + extra_hosts    = []
        + remove         = true
        + security_opt   = []
        + tag            = []
          # (13 unchanged attributes hidden)
      }
  }

# docker_image.haproxy will be created
+ resource "docker_image" "haproxy" {
    + id          = (known after apply)
    + image_id    = (known after apply)
    + name        = "haproxy_http"
    + repo_digest = (known after apply)

    + build {
        + cache_from     = []
        + context        = "./haproxy"
        + dockerfile     = "Dockerfile"
        + extra_hosts    = []
        + remove         = true
        + security_opt   = []
        + tag            = []
```

```
            # (13 unchanged attributes hidden)
        }
    }


  # docker_network.net will be created
  + resource "docker_network" "net" {
      + driver      = (known after apply)
      + id          = (known after apply)
      + internal    = (known after apply)
      + ipam_driver = "default"
      + name        = "http_network"
      + options     = (known after apply)
      + scope       = (known after apply)


      + ipam_config (known after apply)
    }


Plan: 7 to add, 0 to change, 0 to destroy.

docker_network.net: Creating...

docker_image.app2: Creating...

docker_image.app1: Creating...

docker_image.haproxy: Creating...

docker_image.haproxy: Creation complete after 2s
[id=sha256:0cc3e405924924f2f9db0f596fd636a97091fdb672aa7a10fde3501b99ba12d8haproxy_http]

docker_network.net: Creation complete after 2s
[id=3aba87289528f2970bbbbce335554d440d9b37eca03fff5fe63899ab9c47e878]

docker_image.app1: Creation complete after 5s
[id=sha256:a5f64465e352d20aa00f64364c0f3874c62a42d8b3d34e6f8b7d0e8dfa6bd00fapp1]

docker_container.app1: Creating...

docker_image.app2: Creation complete after 5s
[id=sha256:ca6c759c24948a4ad640d14169e34295ef7c0a35012f01655cb32dcf987b7c9capp2]

docker_container.app2: Creating...

docker_container.app1: Creation complete after 2s
[id=17c251df94806cf8c4cfa08c4d36164534b90979a6e16c9956938eeec7b3a4a6]

docker_container.app2: Creation complete after 2s
[id=ca8306700b1f5fd4bc3b3db5fd5639fbd9083165c57e9fc22c243e8d042e2894]

docker_container.haproxy: Creating...
```

```
docker_container.haproxy: Creation complete after 1s
[id=29768276cc0b42dadb19783c8be6b26ac3e72ad169393f5b8371c17f6456acc2]


Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```

Ejecución de `terraform apply` (Escenario 2).

```
PS C:\Users\anton\Desktop\plantilla_practica\escenario2> terraform apply -auto-approve


Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the

following symbols:

  + create


Terraform will perform the following actions:


  # docker_container.haproxy will be created

  + resource "docker_container" "haproxy" {

      + attach                                      = false

      + bridge                                      = (known after apply)

      + command                                     = (known after apply)

      + container_logs                              = (known after apply)

      + container_read_refresh_timeout_milliseconds = 15000

      + entrypoint                                  = (known after apply)

      + env                                         = (known after apply)

      + exit_code                                   = (known after apply)

      + hostname                                    = (known after apply)

      + id                                          = (known after apply)

      + image                                       = "haproxy_tcp"

      + init                                        = (known after apply)

      + ipc_mode                                    = (known after apply)

      + log_driver                                  = (known after apply)

      + logs                                        = false

      + must_run                                    = true
```

```
      + name                                   = "haproxy_tcp"
      + network_data                           = (known after apply)
      + network_mode                           = "bridge"
      + read_only                              = false
      + remove_volumes                         = true
      + restart                                = "no"
      + rm                                     = false
      + runtime                                = (known after apply)
      + security_opts                          = (known after apply)
      + shm_size                               = (known after apply)
      + start                                  = true
      + stdin_open                             = false
      + stop_signal                            = (known after apply)
      + stop_timeout                           = (known after apply)
      + tty                                    = false
      + wait                                   = false
      + wait_timeout                           = 60

      + healthcheck (known after apply)

      + labels (known after apply)

      + networks_advanced {
          + aliases      = []
          + name         = "tcp_network"
            # (2 unchanged attributes hidden)
        }

      + ports {
          + external = 4000
          + internal = 4000
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
```

```
  }


# docker_container.srv1 will be created
+ resource "docker_container" "srv1" {
    + attach                                      = false
    + bridge                                      = (known after apply)
    + command                                     = (known after apply)
    + container_logs                              = (known after apply)
    + container_read_refresh_timeout_milliseconds = 15000
    + entrypoint                                  = (known after apply)
    + env                                         = (known after apply)
    + exit_code                                   = (known after apply)
    + hostname                                    = (known after apply)
    + id                                          = (known after apply)
    + image                                       = "srv1"
    + init                                        = (known after apply)
    + ipc_mode                                    = (known after apply)
    + log_driver                                  = (known after apply)
    + logs                                        = false
    + must_run                                    = true
    + name                                        = "srv1"
    + network_data                                = (known after apply)
    + network_mode                                = "bridge"
    + read_only                                   = false
    + remove_volumes                              = true
    + restart                                     = "no"
    + rm                                          = false
    + runtime                                     = (known after apply)
    + security_opts                               = (known after apply)
    + shm_size                                    = (known after apply)
    + start                                       = true
    + stdin_open                                  = false
    + stop_signal                                 = (known after apply)
```

```
      + stop_timeout                                 = (known after apply)
      + tty                                          = false
      + wait                                         = false
      + wait_timeout                                 = 60

      + healthcheck (known after apply)

      + labels (known after apply)

      + networks_advanced {
          + aliases      = []
          + name         = "tcp_network"
            # (2 unchanged attributes hidden)
        }

      + ports {
          + external = (known after apply)
          + internal = 5000
          + ip       = "0.0.0.0"
          + protocol = "tcp"
        }
    }

  # docker_container.srv2 will be created
  + resource "docker_container" "srv2" {
      + attach                                       = false
      + bridge                                       = (known after apply)
      + command                                      = (known after apply)
      + container_logs                               = (known after apply)
      + container_read_refresh_timeout_milliseconds = 15000
      + entrypoint                                   = (known after apply)
      + env                                          = (known after apply)
      + exit_code                                    = (known after apply)
      + hostname                                     = (known after apply)
```

```
+ id                                      = (known after apply)
+ image                                   = "srv2"
+ init                                    = (known after apply)
+ ipc_mode                                = (known after apply)
+ log_driver                              = (known after apply)
+ logs                                    = false
+ must_run                                = true
+ name                                    = "srv2"
+ network_data                            = (known after apply)
+ network_mode                            = "bridge"
+ read_only                               = false
+ remove_volumes                          = true
+ restart                                 = "no"
+ rm                                      = false
+ runtime                                 = (known after apply)
+ security_opts                           = (known after apply)
+ shm_size                                = (known after apply)
+ start                                   = true
+ stdin_open                              = false
+ stop_signal                             = (known after apply)
+ stop_timeout                            = (known after apply)
+ tty                                     = false
+ wait                                    = false
+ wait_timeout                            = 60

+ healthcheck (known after apply)

+ labels (known after apply)

+ networks_advanced {
    + aliases      = []
    + name         = "tcp_network"
      # (2 unchanged attributes hidden)
```

```
      }


    + ports {
        + external = (known after apply)
        + internal = 5000
        + ip       = "0.0.0.0"
        + protocol = "tcp"
      }
  }

# docker_image.haproxy will be created
+ resource "docker_image" "haproxy" {
    + id          = (known after apply)
    + image_id    = (known after apply)
    + name        = "haproxy_tcp"
    + repo_digest = (known after apply)

    + build {
        + cache_from     = []
        + context        = "./haproxy"
        + dockerfile     = "Dockerfile"
        + extra_hosts    = []
        + remove         = true
        + security_opt   = []
        + tag            = []
          # (13 unchanged attributes hidden)
      }
  }

# docker_image.srv1 will be created
+ resource "docker_image" "srv1" {
    + id          = (known after apply)
    + image_id    = (known after apply)
    + name        = "srv1"
```

```
      + repo_digest = (known after apply)

      + build {
          + cache_from     = []
          + context        = "./srv1"
          + dockerfile     = "Dockerfile"
          + extra_hosts    = []
          + remove         = true
          + security_opt   = []
          + tag            = []
            # (13 unchanged attributes hidden)
        }
    }

# docker_image.srv2 will be created
+ resource "docker_image" "srv2" {
    + id          = (known after apply)
    + image_id    = (known after apply)
    + name        = "srv2"
    + repo_digest = (known after apply)

    + build {
        + cache_from     = []
        + context        = "./srv2"
        + dockerfile     = "Dockerfile"
        + extra_hosts    = []
        + remove         = true
        + security_opt   = []
        + tag            = []
          # (13 unchanged attributes hidden)
      }
  }
```

```
  # docker_network.net will be created
  + resource "docker_network" "net" {
      + driver      = (known after apply)
      + id          = (known after apply)
      + internal    = (known after apply)
      + ipam_driver = "default"
      + name        = "tcp_network"
      + options     = (known after apply)
      + scope       = (known after apply)

      + ipam_config (known after apply)
    }

Plan: 7 to add, 0 to change, 0 to destroy.

docker_network.net: Creating...

docker_image.haproxy: Creating...

docker_image.srv2: Creating...

docker_image.srv1: Creating...

docker_image.haproxy: Creation complete after 2s
[id=sha256:079f432d7cde853ffdc8d88339a73531066b3234eec2a13a53e2df607a4acb8chaproxy_tcp]

docker_network.net: Creation complete after 2s
[id=64641077b624df5d43b251956e090062a9094b243e1f2658047fb4d783f10e72]

docker_image.srv1: Creation complete after 5s
[id=sha256:fe12dfa4e4152e80710b960ffa118106d35be92cd2085e515d66fb027bb7d470srv1]

docker_container.srv1: Creating...

docker_image.srv2: Creation complete after 5s
[id=sha256:d8c190a89ae274c3fad7d883cc0a0cd54b3fd0a1bf9c1154d2944d231b7701bfsrv2]

docker_container.srv2: Creating...

docker_container.srv1: Creation complete after 2s
[id=914f1016c2e934f2fbdccfb090d7ee6092eee0252bf471ed06959e0f7374160c]

docker_container.srv2: Creation complete after 2s
[id=152c409b96e05485d143ae64cb51cabb02c3f1d8fd43d400a9ef1dc1f69a65f8]

docker_container.haproxy: Creating...

docker_container.haproxy: Creation complete after 1s
[id=5e687567617f95c3ec735c8ae5a6ecdbd10e964c128d2cdfe8d5ca60ccd3572b]
```

```
Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
```

Pruebas de balanceo HTTP con `curl`.

```
PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello

StatusCode        : 200

StatusDescription : OK

Content           : Hola desde app1

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15
Content-Type: text/plain;charset=UTF-8
Date: Sun, 09 Nov 2025 12:40:28 GMT

                    Hola desde app1

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:28 GMT]}

Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass

RawContentLength  : 15




PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello



StatusCode        : 200

StatusDescription : OK

Content           : Hola desde app2

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15
Content-Type: text/plain;charset=UTF-8
Date: Sun, 09 Nov 2025 12:40:31 GMT

                    Hola desde app2

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:31 GMT]}
```

```
Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass

RawContentLength  : 15




PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello



StatusCode        : 200

StatusDescription : OK

Content           : Hola desde app1

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15
Content-Type: text/plain;charset=UTF-8
Date: Sun, 09 Nov 2025 12:40:32 GMT

                    Hola desde app1

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:32 GMT]}

Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass

RawContentLength  : 15




PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello



StatusCode        : 200

StatusDescription : OK

Content           : Hola desde app2

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15
Content-Type: text/plain;charset=UTF-8
```

```
Date: Sun, 09 Nov 2025 12:40:33 GMT

                    Hola desde app2

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:33 GMT]}

Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass

RawContentLength  : 15




PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello



StatusCode        : 200

StatusDescription : OK

Content           : Hola desde app1

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15
Content-Type: text/plain;charset=UTF-8
Date: Sun, 09 Nov 2025 12:40:33 GMT

                    Hola desde app1

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:33 GMT]}

Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass

RawContentLength  : 15




PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello



StatusCode        : 200
```

```
StatusDescription : OK

Content           : Hola desde app2

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15
Content-Type: text/plain;charset=UTF-8
Date: Sun, 09 Nov 2025 12:40:34 GMT

                    Hola desde app2

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:34 GMT]}

Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass

RawContentLength  : 15




PS C:\Users\anton\Desktop\plantilla_practica\escenario1> curl http://localhost:80/hello



StatusCode        : 200

StatusDescription : OK

Content           : Hola desde app1

RawContent        : HTTP/1.1 200 OK

                    Content-Length: 15

                    Content-Type: text/plain;charset=UTF-8

                    Date: Sun, 09 Nov 2025 12:40:34 GMT


                    Hola desde app1

Forms             : {}

Headers           : {[Content-Length, 15], [Content-Type, text/plain;charset=UTF-8], [Date, Sun, 09
Nov 2025 12:40:34 GMT]}

Images            : {}

InputFields       : {}

Links             : {}

ParsedHtml        : mshtml.HTMLDocumentClass
```

```
RawContentLength  : 15
```

Pruebas de balanceo TCP (con Python para mandar el ping).

```
PS C:\Users\anton\Desktop\plantilla_practica\escenario2> python -c "import socket; s=socket.socket();
s.connect(('localhost',4000)); s.sendall(b'HELLO\n'); print(s.recv(1024).decode()); s.close()"

>>

OLLEH SRV2



PS C:\Users\anton\Desktop\plantilla_practica\escenario2> python -c "import socket; s=socket.socket();
s.connect(('localhost',4000)); s.sendall(b'HELLO\n'); print(s.recv(1024).decode()); s.close()"

>>

OLLEH SRV1



PS C:\Users\anton\Desktop\plantilla_practica\escenario2> python -c "import socket; s=socket.socket();
s.connect(('localhost',4000)); s.sendall(b'HELLO\n'); print(s.recv(1024).decode()); s.close()"

>>

OLLEH SRV2



PS C:\Users\anton\Desktop\plantilla_practica\escenario2> python -c "import socket; s=socket.socket();
s.connect(('localhost',4000)); s.sendall(b'HELLO\n'); print(s.recv(1024).decode()); s.close()"

>>

OLLEH SRV1



PS C:\Users\anton\Desktop\plantilla_practica\escenario2> python -c "import socket; s=socket.socket();
s.connect(('localhost',4000)); s.sendall(b'HELLO\n'); print(s.recv(1024).decode()); s.close()"

>>

OLLEH SRV2
```