MICROPROYECTO 1 - Cluster LXD + Balanceador de carga usando HAProxy + JMeter en ambiente Vagrant con manejo de fallas y servidores de Backup (Mayo 2021)

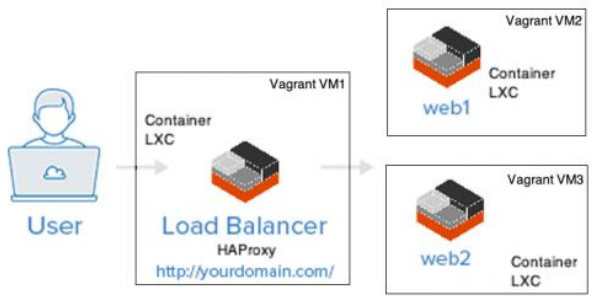
Carlos Arbey Mejía Carlos_arbey.mejia@uao.edu.co
Andrés Felipe Guerra andres_felipe.guerra@uao.edu.co
Repositorio Git Hub Actualizado: https://github.com/cmejia99/proyecto

Abstract - The purpose of this document is to present the solution to micro project 1 of the cloud computing class of the specialization in Artificial Intelligence at the Autonoma de Occidente University from Cali.

I. DESCRIPCIÓN MICRO PROYECTO

Se requiere implementar balanceo de carga con la ayuda de HAProxy (www.haproxy.org/). Los clientes enviarán peticiones al balanceador de carga HAProxy y obtendrán respuestas desde dos servidores web corriendo en contenedores LXD. La configuración requerida se muestra en la siguiente figura # 1.

Figura # 1: Distribución requerida



II. REQUERIMIENTOS

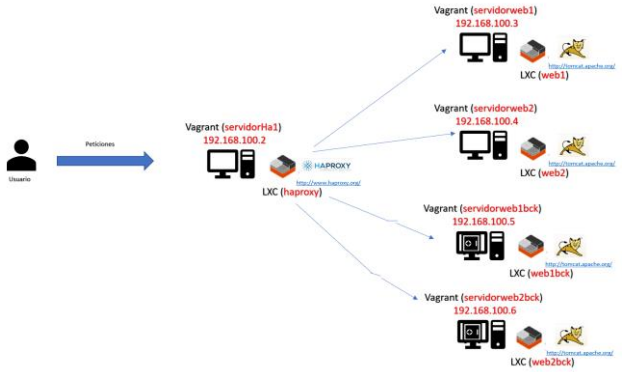
1. Las peticiones no se realizan directamente a los servidores web, sino que el balanceador de carga decidirá que servidor será el encargado de procesar la petición.
2. Los dos servidores tendrán solo apache corriendo y en el balanceador de carga se ejecutará HAProxy.
3. La GUI del balanceador de carga será accesible desde la máquina anfitriona para visualizar el estado y estadísticas detalladas de los servidores web.
4. Cada máquina virtual Vagrant debe correr al menos un contenedor LXD.
5. Al menos dos contenedores LXD deben correr servidores web
6. HAProxy debe correr en un contenedor separado.

III. SOLUCIÓN

A continuación, se describe la arquitectura planteada para la solución a los criterios del micro proyecto:

A. Arquitectura que manejamos para el proyecto:

Figura # 2: Arquitectura proyecto



Como se plantea en la figura # 2, manejaremos lo siguiente:

- 5 máquinas Vagrant con Ubuntu ubuntu-20.04 de la siguiente manera:

  Servidor Haproxy se nombra como servidorHa1 el cual presentará la IP: 192.168.100.2 y estará ejecutando un contenedor LXD con el servicio de Haproxy

  Maquinas Backend:

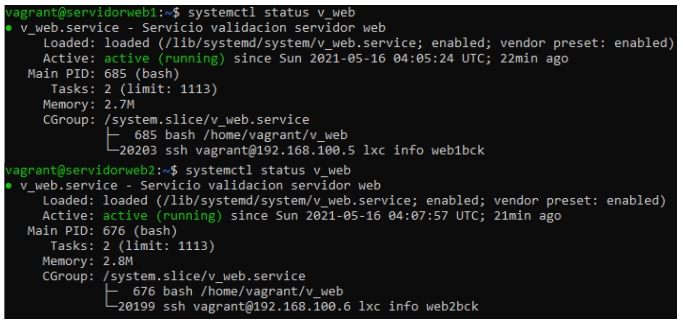
  - **Web1**: Se nombran como servidorweb1 y se asigna la IP: 192.168.100.3, tendrá corriendo un contenedor LXD con una maquina ejecutando un servicio apache2 para la presentación de la página del servidor 1.

  - **Web2**: Se nombran como servidorweb2 y se asigna la IP: 192.168.100.4, tendrá corriendo un contenedor LXD con una maquina ejecutando un servicio Apache2 para la presentación de la página del servidor 2.

  - Para ambas maquina se creo un servicio nombrado **v_web** el cual estará escuchando por un Daemon todo el tiempo si el contenedor del servidor web 1 y el servidor web 2 respectivamente esta activo o inactivo, dado el caso que este no lo este, el servicio generará la tarea de activar el servidor de Backup respectivo para cada servidor.

    Para el caso que los servidores web1 o web2 se haya recuperado el sistema liberará los servidores Backup apagándolos.

Figura # 3: Estatus v_web

- 

Maquinas Backup o respaldo:

- **Web1bck**: Se nombran como servidorweb1bck y se asigna la IP: 192.168.100.5, tendrá corriendo un contenedor LXD con una maquina ejecutando un servicio Apache2 para la presentación de la página de Backup del servidor 1.

- **Web1bck**: Se nombran como servidorweb1bck y se asigna la IP: 192.168.100.5, tendrá corriendo un contenedor LXD con una maquina ejecutando un servicio Apache2 para la presentación de la página de Backup del servidor 1.

- Ambas maquinas de respaldo tendrán la maquina del contenedor LXD apagadas siempre y cuando los servidores

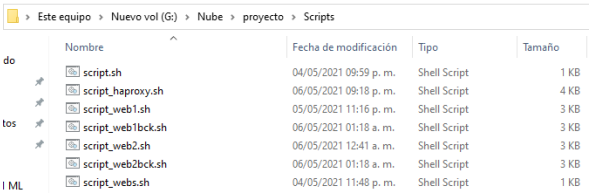
respectivos estén arriba y activos.

IV. DESCRIPCIÓN SOLUCIÓN

A continuación, se describe el paso a paso realizado:

1. Aprovisionamiento, para llevar a cabo el aprovisionamiento inicial de las máquinas, se creó el directorio scripts donde se almacenan los diferentes scripts a utilizar:

Figura # 4: Scripts micro proyecto



En estos archivos podremos encontrar:

- **script.sh**: Este tiene como finalidad actualizar los repositorios de las máquinas, instalar LXD e iniciarlo.

- **script_haproxy.sh**: Este tiene como finalidad crear la maquina Haproxy creando el contenedor con ubuntu:18.04, instalar Haproxy en la maquina LXD, configurar el archivo haproxy.cfg con las maquinas que deberá balancear y crear la página personalizada para el error 503 de no disponibilidad.

- **script_webs.sh**: Este es un script genérico para todos los servidores, el cual, actualizará los repositorios, instalará LXD y la iniciará.

- **script_web1, script_web2, script_web1bck, script_web2bck**: Son scripts personalizados para la creación de los distintos servidores, donde, se instalarán los contenedores ubuntu:18.04, se instalará el servicio Apache2, se crearán las páginas personalizadas por cada servidor y se registrará el proxy de la maquina contenedora y la Vagrant.

2. Configuración Archivo Vagrant, se realiza la configuración del archivo Vagrant de la siguiente manera:

Figura # 5: Vagrantfile micro proyecto

```
Vagrant.configure("2") do |config|
  config.vm.define :servidorHa1 do |servidorHa1|
  servidorHa1.vm.box = "bento/ubuntu-20.04"
  servidorHa1.vm.network :private_network, ip: "192.168.100.2"
  servidorHa1.vm.hostname = "servidorHa1"
  servidorHa1.vm.provision :shell, path: "Scripts/script.sh"
  servidorHa1.vm.provision :shell, path: "Scripts/script_haproxy.sh"
  end
  config.vm.define :servidorweb1 do |servidorweb1|
  servidorweb1.vm.box = "bento/ubuntu-20.04"
  servidorweb1.vm.network :private_network, ip: "192.168.100.3"
  servidorweb1.vm.hostname = "servidorweb1"
  servidorweb1.vm.provision :shell, path: "Scripts/script_webs.sh"
  servidorweb1.vm.provision :shell, path: "Scripts/script_web1.sh"
  end
  config.vm.define :servidorweb2 do |servidorweb2|
  servidorweb2.vm.box = "bento/ubuntu-20.04"
  servidorweb2.vm.network :private_network, ip: "192.168.100.4"
  servidorweb2.vm.hostname = "servidorweb2"
  servidorweb2.vm.provision :shell, path: "Scripts/script_webs.sh"
  servidorweb2.vm.provision :shell, path: "Scripts/script_web2.sh"
  end
  config.vm.define :servidorweb1bck do |servidorweb1bck|
  servidorweb1bck.vm.box = "bento/ubuntu-20.04"
  servidorweb1bck.vm.network :private_network, ip: "192.168.100.5"
  servidorweb1bck.vm.hostname = "servidorweb1bck"
  servidorweb1bck.vm.provision :shell, path: "Scripts/script_webs.sh"
  servidorweb1bck.vm.provision :shell, path: "Scripts/script_web1bck.sh"
  end
  config.vm.define :servidorweb2bck do |servidorweb2bck|
  servidorweb2bck.vm.box = "bento/ubuntu-20.04"
  servidorweb2bck.vm.network :private_network, ip: "192.168.100.6"
  servidorweb2bck.vm.hostname = "servidorweb2bck"
  servidorweb2bck.vm.provision :shell, path: "Scripts/script_webs.sh"
  servidorweb2bck.vm.provision :shell, path: "Scripts/script_web2bck.sh"
  end
end
```

Como se puede evidenciar se realiza el llamado de los scripts respectivos de aprovisionamiento de las maquinas planteadas para el proyecto.

Una vez inicializamos las maquinas vamos a tener el siguiente resultado:

Figura # 6: Estatus de las maquinas

```
PS G:\Nube\proyecto> vagrant status
Current machine states:

servidorHa1               running (virtualbox)
servidorweb1              running (virtualbox)
servidorweb2              running (virtualbox)
servidorweb1bck           running (virtualbox)
servidorweb2bck           running (virtualbox)
```

3. Implementacion del servicio **v_web**: Como se indicó anteriormente se creo un servicio v_web en las maquinas Vagrant Servidorweb1 y Servidorweb2 para que este validando constantemente si los contenedores respectivos del Apache estan respondiendo, dado el caso que no respondan, se generará una tarea de iniciar el contenedor respectivo de Backup.

   Una vez el contenedor original este arriba, el servicio genererá una tarea de inactivar el contenedor del servidor Backup que se encuentre activo.

   Los pasos realizados fueron:

   1. Se creo un archivo v_web1.sh con los permisos de ejecución

Figura # 6: Archivo v_web1.sh creado

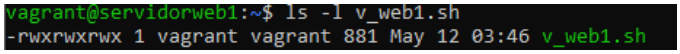
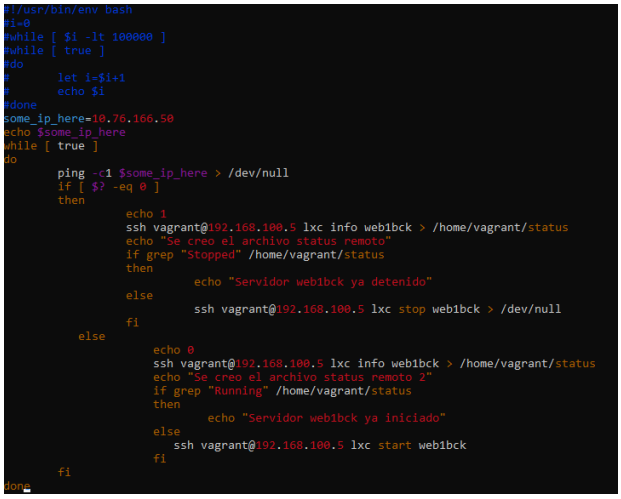


Figura # 7: Contenido archivo v_web1.sh



   Dado que el script realiza comandos ssh, se realizó la creación de certificados de confianza entre los servidores:

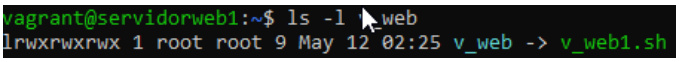
Figura # 8: Certificado llave

```
###Servidor web1
ssh-keygen
###Servidor BAckup
ssh-copy-id vagrant@192.168.100.5
```

   Esto con el motivo de no requerir clave al momento de realizar ejecución de comandos entre los servidores.

   2. Se creo un enlace simbólico al archivo sh:

Figura # 9: Enlace simbolico



   3. Se creo un archivo **v_web.service** el cual se registro en la siguiente ruta:

Figura # 10: Creación archivo **v_web.service**

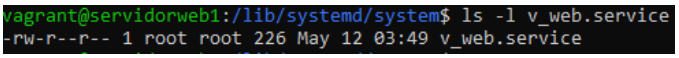
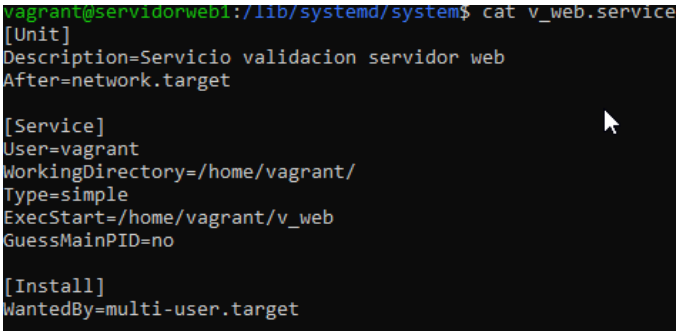
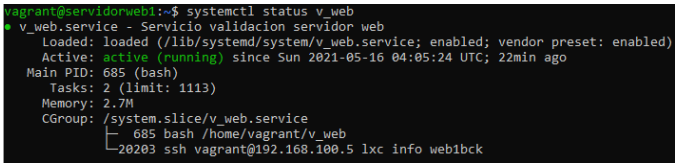


Figura # 11: Contenido **v_web.service**



   4. Como último paso se resgistro el servicio.

Figura # 12: Registro **v_web.service**



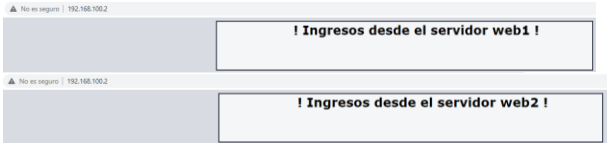
Es de aclarar que los pasos anteriores se realizarán por los servidores web1 y web2.

V. PRUEBAS ARQUITECTURA

Se realizaron las siguientes pruebas:

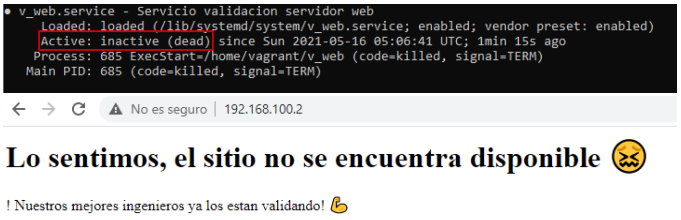
- Validación del sitio inicial con el aprovisionamiento: Se realizó el llamado del sitio con la Ip del servidor Haproxy 192.168.100.2 dos veces, evidenciando que manejaba la carga por turnos:

Figura # 13: Manejo de cargas Haproxy



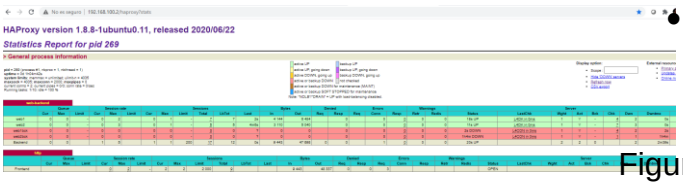
- Se inactiva temporalmente el servicio v_web para validar la página de no disponibilidad:

Figura # 14: Suspención servidor v_web



- Se valida las estadísticas de Haproxy con el aprovisionamiento inicial de la arquitectura:

Figura # 15: Estadistico de cargas

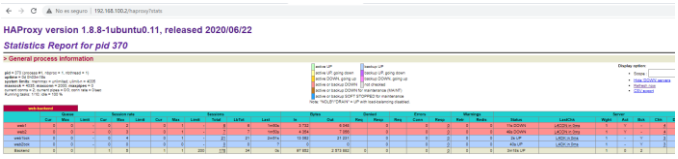


- Se realiza prueba inactivando los dos contenedores para web1 y web2 para validar la activación automática del servicio de Backup:

Figura # 16: Suspension contenedores

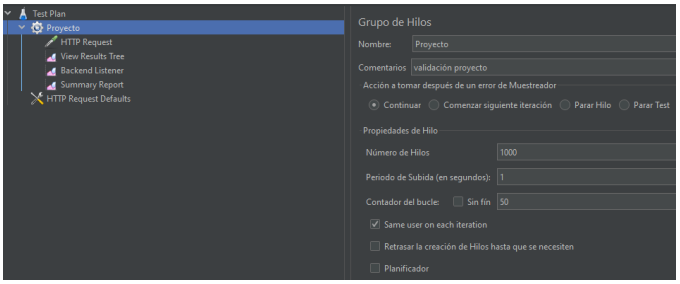
Inicio automatico de los servidores de respaldo:

Figura # 17: Activacion contenedores backups



- Pruebas con Jmeter: La configuración que se llevo a cabo fue la siguiente:

Figura # 18: Jmeter



Se configuro 1000 hilos por 50 ciclos a la Ip: 192.168.100.2

Los escenarios que se validaron fueron:

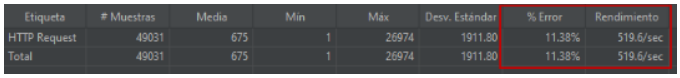
- Sin disponibilidad del servicio: Se evidencio que el porcentaje de error fue del 100%

Figura # 19: Porcentaje error, sin servicios

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Est...	% Error	Rendimie...
HTTP Re...	50000	556	3	1611	143.93	100.00%	1703.2/sec
Total	50000	556	3	1611	143.93	100.00%	1703.2/sec

- Con un servidor activo: Se evidencio que el porcentaje de error fue del 11.38%

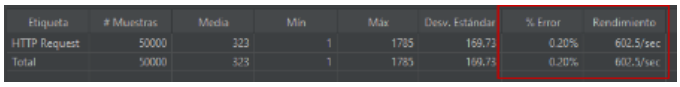
Figura # 20: Porcentaje error, un solo servicio



Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento
HTTP Request	49031	675	1	26974	1911.80	11.38%	519.6/sec
Total	49031	675	1	26974	1911.80	11.38%	519.6/sec

- Con dos servidores activos: Se evidencio que el porcentaje de error fue del 0.20%

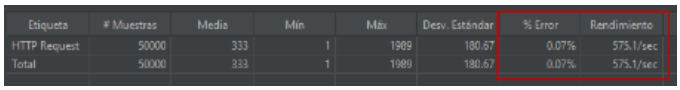
Figura # 21: Porcentaje error, dos servicios



Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento
HTTP Request	50000	323	1	1785	169.73	0.20%	602.5/sec
Total	50000	323	1	1785	169.73	0.20%	602.5/sec

- Con dos servidores activos y uno de Backup activo respaldando la carga: Se evidencio que el porcentaje de error fue del 0.07%

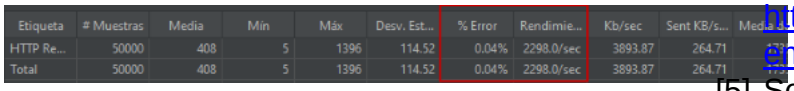
Figura # 22: Porcentaje error, dos servicios y un servicio backup



Etiqueta	# Muestras	Media	Min	Máx	Desv. Estándar	% Error	Rendimiento
HTTP Request	50000	333	1	1989	180.67	0.07%	575.1/sec
Total	50000	333	1	1989	180.67	0.07%	575.1/sec

- Con dos servidores activos y dos de Backup activos respaldando la carga: Se evidencio que el porcentaje de error fue del 0.04%

Figura # 23: Porcentaje error, dos servicios y dos servicio backup



Etiqueta	# Muestras	Media	Min	Máx	Desv. Est...	% Error	Rendimie...	Kb/sec	Sent KB/s...	Med...
HTTP Re...	50000	408	5	1396	114.52	0.04%	2298.0/sec	3893.87	264.71	
Total	50000	408	5	1396	114.52	0.04%	2298.0/sec	3893.87	264.71	

VI. CONCLUSIONES

- El aprovisionamiento de nuestro proyecto es eficiente, ya que su tamaño de datos genera poco consumo de recursos en el servicio, produciendo alta disponibilidad de los servidores que la componen y la escalabilidad se realice en poco tiempo. Esto depende también de nuestra maquina anfitriona.

- Al virtualizar nuestras maquinas y generar un buen aprovisionamiento, podemos distribuir la infraestructura con los elementos que la van a componer de la mejor forma, para que nuestros servicios sean agiles y flexibles en la estabilidad y respuesta que demande los servicios, conforme a los recursos de nuestro distribuidor de cargas a nuestros contenedores web.

- Al realizar una automatización en nuestro distribuidor de carga y las maquinas backend, podemos tener de manera segura, que nuestros servicios van a estar trabajando de acuerdo a lo solicitado por el usuario y de manera automática, según la configuración introducida en nuestro distribuidor de carga con los servicios.

REFERENCIAS

[1] Práctica Linux Containers, UAO Link: https://campus.uaovirtual.edu.co/mod/assign/view.php?id=103799

[2] Practica Aprovisionamiento, UAO Link: https://campus.uaovirtual.edu.co/mod/assign/view.php?id=103807

[3] Práctica Balanceo de Carga con HAProxy UAO Link: https://campus.uaovirtual.edu.co/mod/assign/view.php?id=103813

[4] Enlaces simbólicos Link: https://www.hostinger.co/tutoriales/crear-enlace-simbolico-linux

[5] Servicios Ubuntu Link: https://eltallerdelbit.com/crear-servicio-systemd-ubuntu/