Miniproyecto 1: Cluster LXD + Balanceador de carga usando HAProxy + JMeter en ambiente Vagrant con manejo de fallas y servidores de backup

1st Jhon Alexander Quejada Urrutia
Ingeniero Mecatronico
Universidad Autonoma de Occidente
Cali, Colombia
jhon.quejada@uao.edu.co

1st Juan Camilo Valencia Solarte
Ingeniero Mecatronico
Universidad Autonoma de Occidente
Cali, Colombia
juan_c.valencia_s@uao.edu.co

***Abstract*—This document presents the implementation of linux containers or well known as LXD implementing a HAproxy load balancer with stress tests using Jmeter all based on a vagrant environment with virtualbox as a hypervisor, in addition to fault management and backup servers**

I. Introducción

Hoy en día el uso de servidores web es indispensable para el desarrollo de múltiples actividades en paginas web para diferentes campos alrededor del mundo, con el fin de resolver peticiones a miles, incluso millones de usuarios en diferentes paginas web, se podría afirmar que sin ellos, el internet no seria lo que es hoy en día, de esta manera estos servidores web buscan procesar y responder las solicitudes de los usuarios, pero por eficiencia no es posible tener activos muchos servidores al mismo tiempo, ya que por diferentes motivos el flujo de peticiones no es lineal ni constante, se debe tener una solución donde se pueda prestar el servicio a los usuarios siendo lo mas eficaz posible, es decir usando la cantidad de recursos mas reducidos o que este en la capacidad de responder a las solicitudes, y de esta misma manera utilizar herramientas tales como alternar entre los servidores web posibles en linea según sea el flujo de peticiones, es por esto que se busca desarrollar un sistema el cual pueda fluctuar la carga de los servidores para tener la mayor disponibilidad posible, además de contar con servidores de respaldo dado el caso que los principales no respondan.

II. Marco teórico

A. Entorno virtualizado

En informática un entorno virtualizado significa en resumidas palabras un hardware que no es real. El significado en nuestro contexto de trabajo, donde la maquina física o real, ejecuta una o múltiples maquinas virtualizadas en sus diferentes tipos de virtualización que se vera mas adelante, de esta manera se ejecuta una maquina virtual o también conocida hipervisor, en la siguiente figura se muestra gráficamente lo anterior mencionado.

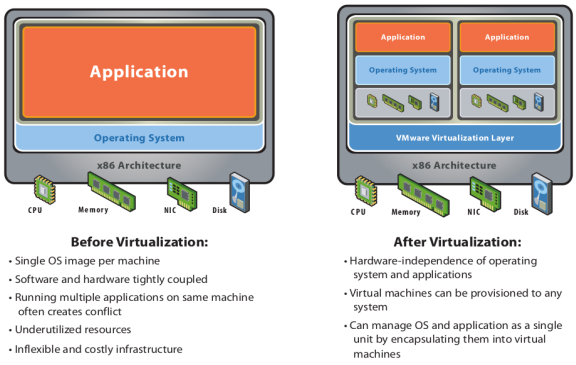


Fig. 1. Virtualización

Existen diferentes tipos de virtualización como se menciono anteriormente, entre ellas se encuentran:

1) Aplicaciones: Algunos ejemplos
   - Stream de aplicaciones
   - Escritorios remotos
2) Redes: Un ejemplo de ello puede ser Redes definidas por software o SDN donde algunos ejemplos pueden ser:
   - Nicira, o ahora conocida como VMware
   - Nuage Netwroks de Alcatel
3) Software basada en hipervisor: Es un método de virtualización sobre un mismo sistema para varias máquinas virtuales. En este tipo de virtualizaciones se suele usar un hipervisor, que es la capa intermedia entre el sistema real y las máquinas virtuales. Estos métodos sirven para virtualizar el hardware. Existen tres tipos:
   - Full virtualization
   - Para virtualization
   - Hardware assisted
4) Almacenamiento: De varios discos físicos se crea una capa virtual donde se crean volúmenes virtuales los cuales son asignados a servidores

5) Sistemas operativos/ Particionamiento: La virtualización del sistema operativo se realiza en el kernel, es decir, los administradores de tareas centrales de los sistemas operativos, este tipo de virtualización trae múltiples beneficios tales como:
   - Reduce el costo del hardware en masa, ya que las computadoras no requieren capacidades tan inmediatas.
   - Aumenta la seguridad porque todas las instancias virtuales se pueden supervisar y aislar.
   - Limita el tiempo que se destina a los servicios de TI, como las actualizaciones de software.

B. Hipervisor VMM

Un hipervisor es el encargado de hacer el monitoreo y control de la o las maquinas virtuales, por esto también es el encargado de la asignación de recursos a estas maquinas virtuales en tiempo real, estas suelen soportar diferentes tipos de sistemas operativos o o también conocidos de manera resumida por sus siglas en ingles OS, existen dos tipos:

1) Tipo 1 (Nativos): unas de las principales características de este tipo es que corre directamente sobre el hardware, y no necesita un sistema operativo host

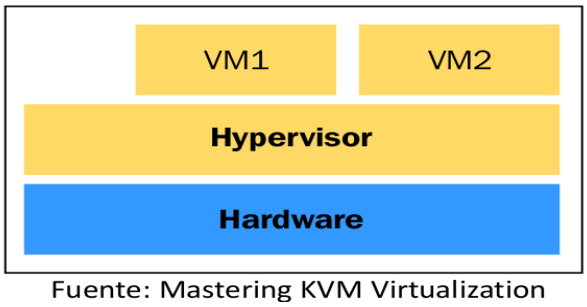


Fig. 2. Hipervisor tipo 1

2) Tipo 2: Este tipo de hipervisor reside sobre un sistema operativo (host) y depente de este para su operación.

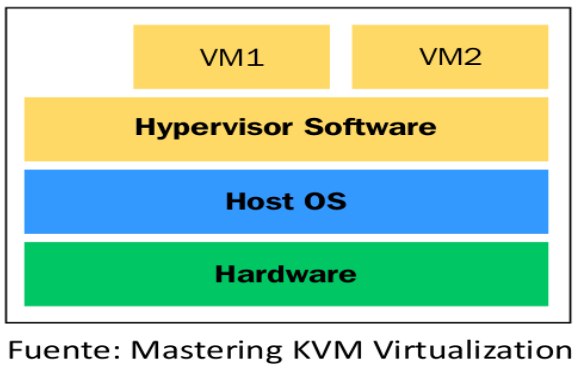


Fig. 3. Hipervisor tipo 2

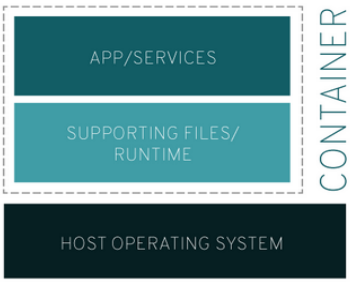
C. Vagrant

Vagrant es una herramienta gratuita de línea de comandos, disponible para Windows, MacOS X y GNU/Linux, que permite generar entornos de desarrollo reproducibles y compartibles de forma muy sencilla. Para ello, Vagrant crea y configura máquinas virtuales a partir de simples ficheros de configuración mediante un Vagrantfile el cual le caben configuraciones múltiples, tales como configurar un proveedor, limitar los recursos de cada maquina virtual que lance. Este cuenta con comandos como:

- **Vagrant init** el cual crea un Vagrantfile dentro del directorio que estemos
- **Vagrant suspend** el cual suspende una maquina virtual guardando el estado en el que quedo.
- **Vagrant up** el cual inicializa el Vagrantfile con las maquinas virtuales y configuraciones previamente configuradas.
- **Vagrant halt** el cual realiza el pagado de una maquina virtual especifica o de todas las que esten incicializadas en el directorio.
- **Vagrant destroy** el cual destruye la máquina virtual y todo su contenido.

D. Contenedores Linux

Los contenedores Linux son un conjunto de procesos separados del resto del sistema, todos los archivos que ejecutan provienen de imágenes diferentes, esto quiere decir que son portátiles y uniformes en todas las etapas, tales como desarrollo, prueba y producción



Cada contenedor tiene archivos que hacen que diferentes aplicaciones se soporten en el mismo para una tarea especifica. Los contenedores comparten el mismo kernel del sistema operativo y separan los procesos de las aplicaciones del resto del sistema.

- LXD: Es un administrador de contenedores Linux, que se basa en imágenes prefabricadas, este ofrrece gestion de redes en lo cabe, creación y configuración de puentes, túneles entre el host y el guest.
  Ademas ofrece la gestion de almacenamiento tales como bakcends, grupos de almacenamiento y volúmenes. Este es un claro ejemplo de virtualizacion de OS, por lo tanto se puede definir como un hipervisor pero no como los tradicionales (tipo 1 y 2), entonces este es definido como un tipo C o administrador de contenedores.

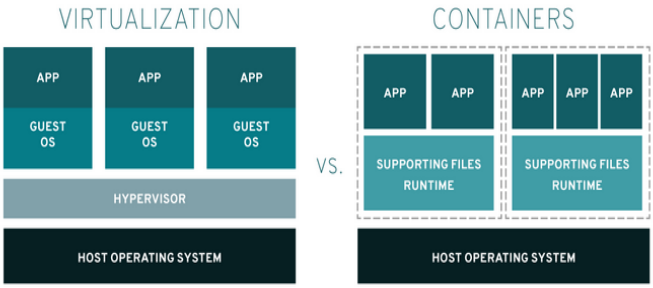


Fig. 4. Contenedor vs Virtualización

A continuación la visualización del funcionamiento de LXD

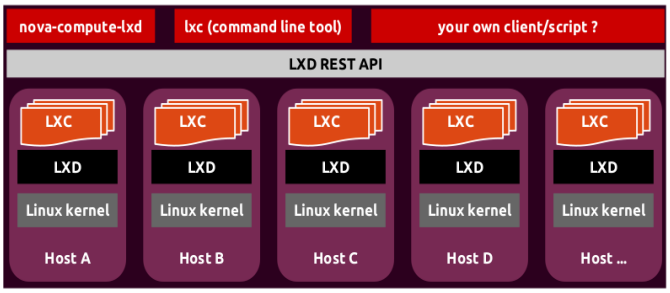


Fig. 5. Estructura LXD

E. Aprovisionamiento

El aprovisionamiento permite instalar software automáticamente, alterar configuraciones de la maquina virtual y muchos procesos mas, este proceso permite ahorrar el tiempo de instalación de paquetes necesario en la maquina virtual paso a paso lo cual no es eficiente, debido que si requerimos hacer este proceso en múltiples maquinas seria mucho el tiempo en configurar. En Vagrant este aprovisionamiento se peude hacer de diferentes maneras tales como:

- File
- Shell
- Ansible
- Puppet
- Chef
- Docker
- Podman
- Salt

El fin de cada metodo es proporcionar los comandos necesarios para instalar las dependencias y las configuraciones necesarias que se requiera en cada maquina virtual que se levante por medio de **vagrant up**, los comandos vagrant de aprovisionamiento son los siguiente:

- **vagrant up –provision** con esta instrucción se levanta el aprovisionamiento, si se hace solo con **vagrant up** este no levanta el aprovisionamiento
- **vagrant provision** de esta manera se levanta el aprovisionamiento al igual que con **vagrant reload –provision**
- **vagrant –no-provision** obliga a no levantar el aprovisionamiento configurado que se haya configurado

F. Balanceo de carga HAProxy

El balanceo de carga es la manera en que las peticiones de internet son distribuidas en los diferentes servidores web que estén disponibles. Existen diferentes niveles.

://www.overleaf.com/project/617ee138ea3e15aaaccacb40

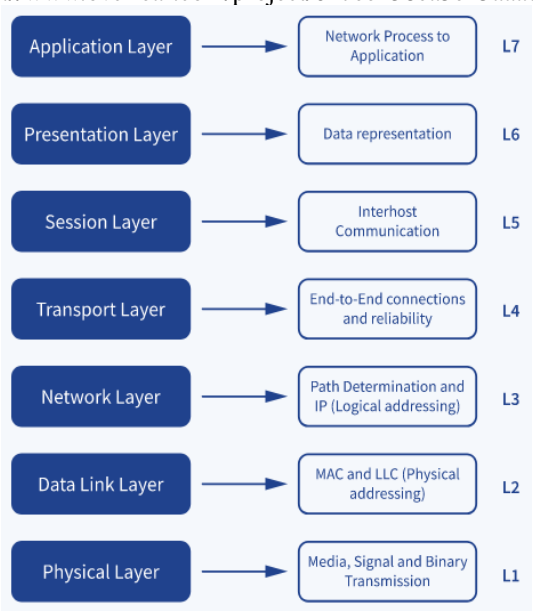


Fig. 6. Niveles de balanceo de carga

Este balanceador maneja tres tipos de terminologia:

Access Control List (ACL): Realizan acciones a base de condiciones que permiten reenvío de trafico de manera flexible basándose en el cumplimiento de patrones, o numero de conexiones hacia un backend por ejemplo

Backend: En simples palabras es el conjunto de servidores que reciben las peticiones enviadas por HAProxy

Frontend: Este se define como las peticiones que van a ser reenviadas a los backends

Existen diferentes algoritmos de balanceo de carga:

- Round Robin: Básicamente los servidores web son seleccionados por turnos.
- Least Connection: Este es un algoritmo de balanceo de carga dinámico en el que las solicitudes de los clientes se distribuyen al servidor de aplicaciones con el menor número de conexiones activas en el momento en que se recibe la solicitud del cliente.
- Resource: Este algoritmo de equilibrio de carga requiere la instalación de un agente en el servidor de aplicaciones que informa sobre su carga actual al equilibrador de carga. El agente instalado supervisa los recursos y el estado de disponibilidad de los servidores de aplicaciones,

Por otra parte existen balanceadores de carga diferentes al HAProxy tales como:

- Linux Virtual Servers (LVS): Balanceador simple de nivel 4 incluido en varias distribuciones de Linux
- Nginx: Servidor web que incluye capacidades de proxy y load-balancing
- Orquestación de contenedores: Docker Swarm, Kubernetes
- IaaS: OpenNebula, OpenStack

III. IMPLEMENTACIÓN

Para el desarrollo de esta implementación, se usó la herramienta Vagrant como el soporte para correr as maquinas virtuales en el hipervisor Virtualbox. Aquí tomamos como referencia el box que contienen la imagen de Ubuntu 18.04 del directorio de Vagrant Cloud directamente. Consecuentemente, se configuró un Vagrantfile en donde se especifica la configuración que van a tener las maquinas virtuales implementadas y posteriormente su correspondiente aprovisionamiento, veamos esto mas a detalle.

El Vagrantfile, es un archivo de configuración que se crea con el propósito de establecer nuestras maquinas virtuales y proporcionarle todas las características necesarias para que estas funcionen. En el este archivo, se pueden configurar instrucciones de aprovisionamiento según el elegido, definir direcciones IP de cada una de las maquinas para realizar comunicación entre ellas en tareas especificas, definir un nombre característico de la maquina virtual, y una de las características mas importantes es implementar la imagen que se utilizará para emular un OS deseado, en este caso escogimos Ubuntu 18.04 LTS. (ver figura 7)

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|

  if Vagrant.has_plugin? "vagrant-vbguest"
    config.vbguest.no_install  = true
    config.vbguest.auto_update = false
    config.vbguest.no_remote   = true
  end

  #config.vm.define :clienteUbuntu do |clienteUbuntu|
    #clienteUbuntu.vm.box = "bento/ubuntu-20.04"
    #clienteUbuntu.vm.network :private_network, ip: "192.168.100.2"
    #clienteUbuntu.vm.hostname = "clienteUbuntu"
  #end

  #config.vm.define :servidorUbuntu do |servidorUbuntu|
    #servidorUbuntu.vm.box = "bento/ubuntu-20.04"
    #servidorUbuntu.vm.network :private_network, ip: "192.168.100.3"
    #servidorUbuntu.vm.provision "shell", path: "script.sh"
    #servidorUbuntu.vm.hostname = "servidorUbuntu"
  #end

  # Maquinas virtuales Miniproyecto1

  #Maquina virtual 1 - Load Balancer
  config.vm.define :virtualMachine1 do |virtualMachine1|
    virtualMachine1.vm.box = "bento/ubuntu-18.04"
    virtualMachine1.vm.network :private_network, ip: "192.168.100.4"
    virtualMachine1.vm.hostname = "virtualMachine1"
    virtualMachine1.vm.provision "shell", path: "scriptlb.sh"
  end

  #Maquina virtual 2 - web1
  config.vm.define :virtualMachine2 do |virtualMachine2|
    virtualMachine2.vm.box = "bento/ubuntu-18.04"
    virtualMachine2.vm.network :private_network, ip: "192.168.100.5"
    virtualMachine2.vm.hostname = "virtualMachine2"
    virtualMachine2.vm.provision "shell", path: "scriptvm02.sh"
  end

  #Maquina virtual 3 - web2
  config.vm.define :virtualMachine3 do |virtualMachine3|
    virtualMachine3.vm.box = "bento/ubuntu-18.04"
    virtualMachine3.vm.network :private_network, ip: "192.168.100.6"
    virtualMachine3.vm.hostname = "virtualMachine3"
    virtualMachine3.vm.provision "shell", path: "scriptvm03.sh"
  end


end
```

Fig. 7. Configuración de archivo Vagrantfile

Para crear las maquinas virtuales es necesario utilizar el siguiente comando una vez definido el archivo Vagrantfile que contiene la configuración de nuestras maquinas, como se puede observar en la siguiente linea de codigo.

```
jaqu@jaqu:~/Documents/ESPECIALIZACIONIA/cloud_computing/virtualMachine$ vagrant up
```

Fig. 8. Comando Vagrant para encender o crear maquinas virtuales

Una vez encendidas nuestras maquinas virtuales podemos visualizar el estado en el que se encuentran. Cabe resaltar que en nuestro caso ya teníamos creadas nuestras maquinas virtuales, por ende solamente fue necesario encenderlas para continuar con la implementación.

```
jaqu@jaqu:~/Documents/ESPECIALIZACIONIA/cloud_computing/virtualMachine$ vagrant status
Current machine states:

virtualMachine1           running (virtualbox)
virtualMachine2           running (virtualbox)
virtualMachine3           running (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Fig. 9. Visualización del estado de las maquinas virtuales

A. *Aprovisionamiento*

El aprovisionamiento de servicios es un recurso que nos permite modificar y configurar a gran detalle nuestras maquinas virtuales, en donde podemos instalar software, realizar actualizaciones, instalar dependencias y/o librerías entre otras.
En nuestro caso, aprovisionamos nuestras maquinas virtuales, instalando el servidor Web Apache2 y el balanceador de carga HAProxy, cada uno en su respectivo contenedor. En las siguientes figuras se observa como está el archivo de configuración para realizar el aprovisionamiento de nuestras maquinas virtuales. También es importante tener en cuenta que previamente se realizó el clustering de nuestras VM, lo que nos proporciona una versatilidad mayor al instalar todos los recursos necesarios para nuestros contenedores, ya que teniendo el clustering no es necesario ingresar a las maquinas virtuales en donde se encuentran los contenedores web1 y web2, si no que desde el contenedor principal haproxy, podemos realizar todo el proceso de instalación. (Observe a mayor detalle las siguientes figuras)

```
#APROVISIONAMIENTO DEL BALANCEADOR DE CARGA

echo "Creación de contenedores web"
lxc launch ubuntu:18.04 HAProxy --target virutalMachine1
lxc launch ubuntu:18.04 Web01 --target vitualMachine2
lxc launch ubuntu:18.04 Web02 --target virtualMachine3

echo "Ingresamos al modo superusuario del contenedor Web01 para instalar el apache2"
lxc exec Web01 /bin/bash
apt update && apt upgrade
apt install apache2
systemclt enable apache2
vim /var/www/html/index.html

echo "Modificando el archivo index.html"
sed -i '
<!DOCTYPE html>
<html>
<body>
<h1>CONTENEDOR WEB1 VIRTUALMACHINE2</h1>
<p>Bienvenidos al contenedor LXD WEB1</p>
</body>
</html>
' /var/www/html/index.html
lxc file push index.html Web01/var/www/html/index.html
systemctl start apache2
exit
```

Fig. 10. Creación de contenedores ”HAProxy, Web01, Web02”, instalación de Apache2 en contenedor Web01

Una vez instalado el servidor web1 en el contenedor web1 desde la maquina virtual 1, procedemos a ingresar al modo superusuario del contenedor web2, para realizar la configuración del Apache2.

```
echo "Ingresamos al modo superusuario del contenedor Web01 para instalar el apache2"
lxc exec Web02 /bin/bash
apt update && apt upgrade
apt install apache2
systemctl enable apache2
vim /var/www/html/index.html

echo "Modificando el archivo index.html"
sed -i '
<!DOCTYPE html>
<html>
<body>
<h1>CONTENEDOR WEB2 VIRTUALMACHINE</h1>
<p>Bienvenidos al contenedor LXD WEB2</p>
</body>
</html>
' /var/www/html/index.html
lxc file push index.html Web02/var/www/html/index.html
systemctl start apache2
exit
```

Fig. 11. Instalación de Apache2 en contenedor Web02

Finalmente, procedemos a realizar la instalación del haproxy en el contenedor HAProxy de nuestra maquina virtual 1, con este, vamos a ejecutar todas las peticiones emitidas por diferentes usuarios, las cuales van a ser solucionadas mediante nuestros contenedores web que integran el servidor Web Apache2. Además de esto, modificamos el archivo haproxy.cfg para generar los servidores de Backend que se encargarán de resolver estas peticiones y el servidor de frontend que se encargará de tomar la decisión acerca de a cual servidor web direccionar las peticiones hechas por los usuarios.

```
echo "Ingresamos al modo superusuario del contenedor HAProxy para instalar el haproxy"
lxc exec HAProxy /bin/bash
apt update && apt upgrade
apt install haproxy
systemctl enable haproxy

echo "Modificamos el archivos haproxy.cfg"
vim /etc/haproxy/haproxy.cfg
sed -i '
global
        log /dev/log    local0
        log /dev/log    local1 notice
        chroot /var/lib/haproxy
        stats socket /run/haproxy/admin.sock mode 660 level admin expose-fd listeners
        stats timeout 30s
        user haproxy
        group haproxy
        daemon

        # Default SSL material locations
        ca-base /etc/ssl/certs
        crt-base /etc/ssl/private

        # Default ciphers to use on SSL-enabled listening sockets.
        # For more information, see ciphers(1SSL). This list is from:
        #  https://hynek.me/articles/hardening-your-web-servers-ssl-ciphers/
        # An alternative list with additional directives can be obtained from
        #  https://mozilla.github.io/server-side-tls/ssl-config-generator/?server=haproxy
        ssl-default-bind-ciphers ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:RSA+A
ESGCM:RSA+AES:!aNULL:!MD5:!DSS
        ssl-default-bind-options no-sslv3

defaults
        log     global
        mode    http
        option  httplog
        option  dontlognull
        timeout connect 5000
        timeout client  50000
        timeout server  50000
        errorfile 400 /etc/haproxy/errors/400.http
        errorfile 403 /etc/haproxy/errors/403.http
        errorfile 408 /etc/haproxy/errors/408.http
        errorfile 500 /etc/haproxy/errors/500.http
        errorfile 502 /etc/haproxy/errors/502.http
        errorfile 503 /etc/haproxy/errors/503.http
        errorfile 504 /etc/haproxy/errors/504.http


backend web-backend
   balance roundrobin
   stats enable
   stats auth admin:admin
   stats uri /haproxy?stats

   server web1 240.101.0.172:80 check
   server web2 240.102.0.58:80 check

frontend http
  bind *:80
  default_backend web-backend
' /etc/haproxy/haproxy.cfg
```

Fig. 12. Instalación de Haproxy y configuración del archivo haproxy.cfg

```
systemctl start haproxy
exit

echo "Configuración de reenvio de puertos para probar el balanceador de carga"
lxc config device add haproxy http proxy listen=tcp:0.0.0.0:2080 connect=tcp:127.0.0.1:80
```

Fig. 13. Cierre de superusuario contenedor HAProxy y creación de puerto para comunicación local

B. *Implementación del Clustering*

Primero que todo cabe aclarar que es el 'clustering' entre los servidores. La definición de cluster o clustering de servidores se refiere a lo conexión de varios servidores y que estos funcionen como uno solo, el objetivo de esta implementación es facilitar que los recursos estén siempre disponibles entre ellos, para este caso se establecen maestros y esclavos los cuales reciben instrucciones del maestro, para nuestro caso de estudio, el nodo del contenedor donde se ubica el balanceador de carga HAProxy sera el maestro, y los servidores web serán los esclavos.

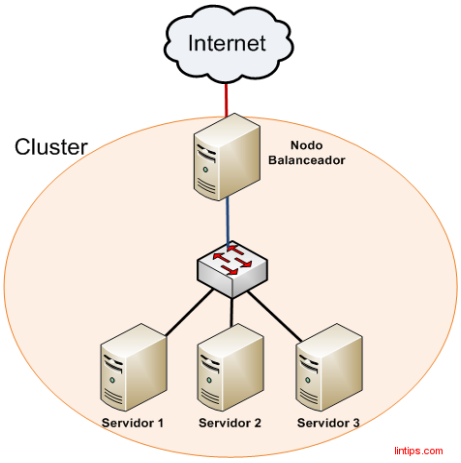


Fig. 14. Diagrama esquemático del Clustering

Consecuentemente encendamos nuestras maquinas virtuales, podemos establecer na cocnexión hacia ellas usando el comando ssh, con el proposito de iniciar sesion en nuestro sistema virtual. (Observe la siguiente figura)

```
jaqu@jaqu:~/Documents/ESPECIALIZACIONIA/cloud_computing/virtualMachine$ vagrant ssh virtualMachine1
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 4.15.0-161-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Tue Nov  2 18:26:51 UTC 2021

  System load:  0.0               Users logged in:       0
  Usage of /:   6.1% of 61.80GB   IP address for eth0:   10.0.2.15
  Memory usage: 17%               IP address for eth1:   192.168.100.4
  Swap usage:   0%                IP address for lxdfan0: 240.4.0.1
  Processes:    102


This system is built by the Bento project by Chef Software
More information can be found at https://github.com/chef/bento
Last login: Mon Nov  1 22:11:31 2021 from 10.0.2.2
```

Fig. 15. Comando ssh para establecer conexión con las maquinas vituales

Acto seguido, podemos observar como se encuentra configurado el clustering de nuestras maquinas virtuales; se puede observar que las tres maquinas virtuales están conectadas por el puerto 8443, su estado actual en "Online", y están "Fully operational", con lo que podemos verificar que el clustering entre ellas está activo. Con este, tenemos acceso al modo superusuario de cualquier otra maquina virtual, y esto nos permite poder crear archivo entre maquinas virtuales sin acceder a ellas.

```
vagrant@virtualMachine1:~$ lxc cluster list
+---------------+---------------------------+----------+--------+-----------------+
|     NAME      |            URL            | DATABASE | STATE  |     MESSAGE     |
+---------------+---------------------------+----------+--------+-----------------+
| virtualMachine1 | https://192.168.100.4:8443 | YES      | ONLINE | fully operational |
+---------------+---------------------------+----------+--------+-----------------+
| virtualMachine2 | https://192.168.100.5:8443 | YES      | ONLINE | fully operational |
+---------------+---------------------------+----------+--------+-----------------+
| virtualMachine3 | https://192.168.100.6:8443 | YES      | ONLINE | fully operational |
+---------------+---------------------------+----------+--------+-----------------+
```

Fig. 16. Visualización de la configuración del clustering

C. *Balanceo de carga con HAProxy*

Para la implementación del balanceo de carga, Los clientes enviarán peticiones al contenedor HAProxy y obtendrán respuestas desde nuestros servidores web1 y web2. En la siguiente figura podemos observar como se comporta el manejo de cargas hacia nuestros servidores web, usando el comando **" curl (ip del contenedor haproxy) "**, para hacer pruebas desde nuestro terminal. Para ello, se hicieron dos peticiones a manera de ejemplo para ver como respondían nuestros servidores web, y se observa como se repartió el manejo de cargas entre ellos.

Es importante tener en cuenta que estamos usando el método roundrobin, en donde se realizan peticiones de manejo de carga por turno alternando los servidores web que se hayan configurado.

```
vagrant@virtualMachine1:~$ curl 240.4.0.251
<!DOCTYPE html>
<html>
<body>
<h1>PAGINA DE PRUEBA CONTENEDOR WEB2 DE LA VM03</h1>
<p>Bienvenidos a mi contenedor LXD web2</p>
</body>
</html>
vagrant@virtualMachine1:~$ curl 240.4.0.251
<!DOCTYPE html>
<html>
<body>
<h1>PAGINA DE PRUEBA CONTENEDOR WEB1 DE LA VM02</h1>
<p>Bienvenidos a mi contenedor LXD web1</p>
</body>
</html>
```

Fig. 17. Prueba de manejo de cargas a servidores web1 y web2

Prueba de manejo de cargas desde red local

Para poder realizar pruebas desde nuestro navegador web, se realizó un reenvío de puertos, para conectarnos por el puerto 80 y poder realizar el mismo envío de peticiones para pruebas de carga de nuestros servidores web1 y web2.

Para el reenvío de puertos usamos la siguiente instrucción:

```
vagrant@virtualMachine1:~$ lxc config device add haproxy http proxy
 listen=tcp:0.0.0.0:1080 connect=tcp:127.0.0.1:80
```

Fig. 18. Configuración del reenvío de puertos

Por otra parte podemos comprobar la respuesta a las peticiones por medio de nuestro navegador web, simplemente ingresando la dirección IP de nuestro nodo maestro en nuestro caso es 192.168.100.4 por el puerto 1080 previamente configurado para el reenvío, se hace la petición y el balanceador alternara los servidores de acuerdo a las peticiones, que simplemente podemos refrescar la pagina para ello, y como se puede observar en las siguientes figuras:

Fig. 19. Prueba de manejo de cargas en red local usando el navegador firefox servidor web1

Fig. 20. Prueba de manejo de cargas en red local usando el navegador firefox servidor web2

Pruebas de carga usando Jmeter

Primero que todo para iniciar la prueba de carga creamos una prueba la cual configuraremos la ip a la que le haremos las peticiones al igual que el numero de ellas en un tiempo estimado.
Creamos la configuración de IP y puertos con los cuales nos queremos comunicar desde Jmeter.

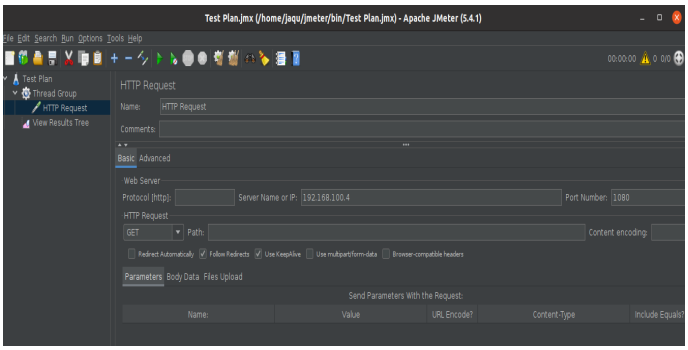


Fig. 21. Configuración de IP y puerto

Paso siguiente configuramos las peticiones, cuantas y su frecuencia, para nuestro caso configuramos 2000 usuarios haciendo peticiones cada 0.5 segundos

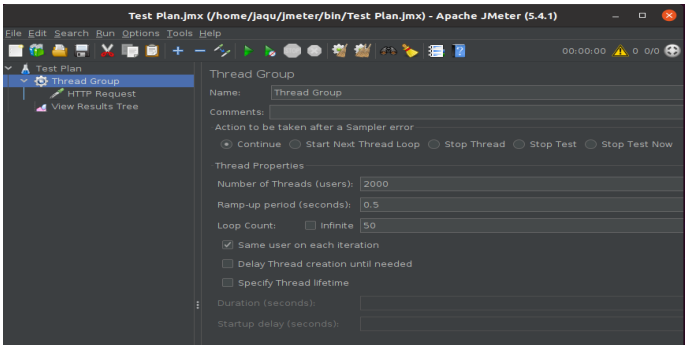


Fig. 22. Configuración de cantidad de peticiones y su frecuencia

Después de las peticiones, se corre el programa y se puede observar como se cargan las peticiones al balanceador y sus respuestas.

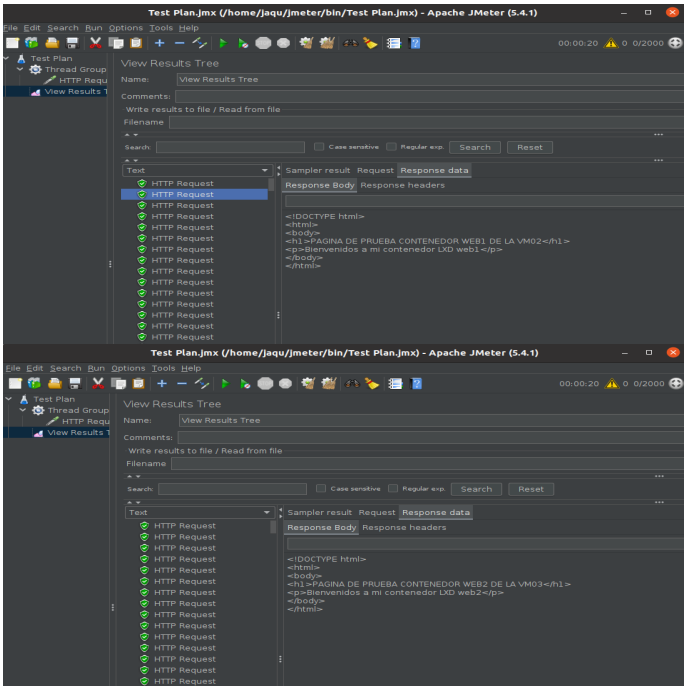


Fig. 23. Respuestas del balanceador en Jmeter

IV. Conclusiones

- El aprovisionamiento de una maquina virtual da la posibilidad de tener configuraciones previas y autogestión de la maquina al momento de iniciar, lo que es bastante beneficioso ya que si llegáramos a tratar con múltiples paquetes que se instalen manualmente, realmente tomaría mucho tiempo cada vez que lancemos una maquina, dado el caso que sean muchas maquinas en un mismo servidor seria una tarea tediosa instalar paquete o dependencias una por una, este punto es de vital importancia, dado el caso de destrucción de maquinas, y no recordemos que paquetes debemos configurar de nuevo o simplemente olvidemos uno de muchos paquetes, causaría errores mas adelante, con el aprovisionamiento corregiría esta parte
- El uso del clustering hicimos un símil sobre la comunicación reenvío de puertos que hicimos en practicas anteriores lo cual hacia comunicación directa entre maquinas virtuales en el caso anterior de estudio fue cliente y servidor, pero al momento de tener varias maquinas por comunicar seria utilizar el reenvío de puertos entre cada maquina y la principal, lo cual no seria muy practico, es por eso que por medio del clustering logramos la comunicación de todas las maquinas de manera mas eficiente, por el cual desde el nodo principal podríamos acceder a cada maquina sin necesidad de usar el vagrant ssh ”nombre de la maquina” lo cual resulto de vital importancia en la comunicación.
- El uso de herramientas como Jmeter nos ayuda y facilita a observar el comportamiento de nuestros servidores y su respectivo balanceador que nuestro caso fue un HAProxy, además que es entorno gráfico lo que facilita aun mas su uso
- Una manera mas efectiva de aprovisionamiento es tener toda la maquina lista para el trabajo solo usando vagrant up lo cual significa que debemos hacer como primer paso el aprovisionamiento de las maquinas virtuales y sus respectivas dependencias y paquetes a utilizar, como se hizo en este documento configurar los servidores web y los balanceadores de preferencia, teniendo estos podemos configurar el clustering de paso y tener las maquinas listas para el trabajo a realizar, este es un punto a tener en cuenta para proyectos futuros

References

[1] https://www.redhat.com/es/forums/latam
[2] https://stackoverflow.com/
[3] https://github.com/
[4] ttps://www.vagrantup.com/
[5] Just me and Opensource https://www.youtube.com/watch?v=LhF2HxqKPzk y https://www.youtube.com/watch?v=3UTvQ4ZNV1Yt=4s
[6] Oscar Mondragon https://www.youtube.com/watch?v=jQQS3uGyOGQt=962s
[7] Material de clase especializacion en Inteligencia Artificial Universidad Autónoma de Occidente Cali, Curso de Computación en la Nube