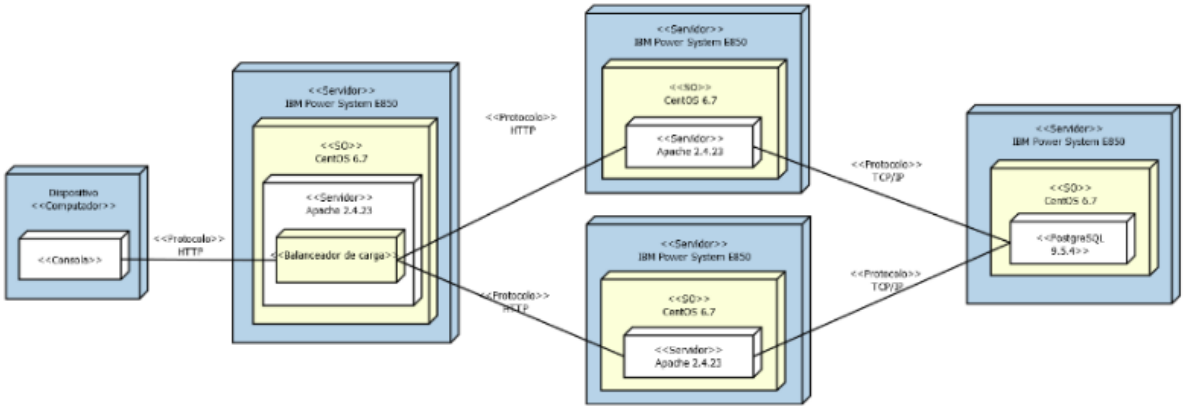
Sistemas distribuidos

Primer examen

Rodrigo Rivera A00268536

1. Comandos de Linux necesarios para aprovisionar las máquinas virtuales.
   Para lograr aprovisionar las máquinas necesarias para brindar el servicio presentado por la arquitectura propuesta:



   Se debe hacer la configuración de cada uno de los siguientes servicios: Balanceador de cargas, web y base de datos para ello hay que tener en cuenta que se configurará en máquinas en centos.
   Comencemos con el balanceador de cargas:
   Para esta parte usaremos Nginx el cual es un servidor web más liviano y flexible que apache y a su vez permite hacer load balancing.
   Los comandos a usar serán los siguientes:

   Primero nos trasladamos a los repositorios de yum:
   `# cd /etc/yum.repos.d`

   Luego creamos uno para Nginx:
   `# vi nginx.repo`

   En el archivo pegamos las siguientes especificaciones para la descarga del servidor:

```
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/centos/$releasever/$basearch/
gpgcheck=0
enabled=1
```

   Por último instalamos nginx:

```
#yum install nginx
```

Para poder ejecutar la función de balanceador de carga debemos configurar el archivo:

```
# cd /etc/nginx/nginx.conf
```

Con la siguiente información:

```
worker_processes  1;
events {
  worker_connections 1024;
}
http {
   upstream servers {
        server 192.168.131.83;
        server 192.168.131.84;
   }
   server {
       listen 8080;
       location / {
            proxy_pass http://servers;
       }
   }
}
```

Es aquí donde se le define al balanceador de cargas http que reenvíe la solicitud recibida por el puerto 8080 a los servidores 192.168.131.83-192.168.131.84.

Después del paso anterior pasamos a habilitar el puerto 8080, usado en este caso ya que el 80 se usará para servicio web en iptables con el siguiente comando:

```
# iptables -I INPUT -p tcp --dport 8080 --syn -j ACCEPT
# service iptables restart
```

Finalmente, iniciamos el servicio nginx

```
# service nginx start
```

Para el aprovisionamiento de la parte web seguimos los siguientes comandos:

```
#yum install httpd -y

#yum install php -y

#yum install php-mysql -y

#yum install mysql -y
```

Luego.

#iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 80 -j ACCEPT

#service iptables sabe

Como se mencionó anteriormente para el aprovisionamiento web se habilita el puerto 80 para las peticiones http.

Por último, para la sección de la base de datos:

#yum install mysql-server -y

#yum install expect -y

iptables -I INPUT 5 -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT

service iptables sabe

Como se puede apreciar, se abre el puerto 3306 para atender las peticiones.

2. El vagrant file es el siguiente:

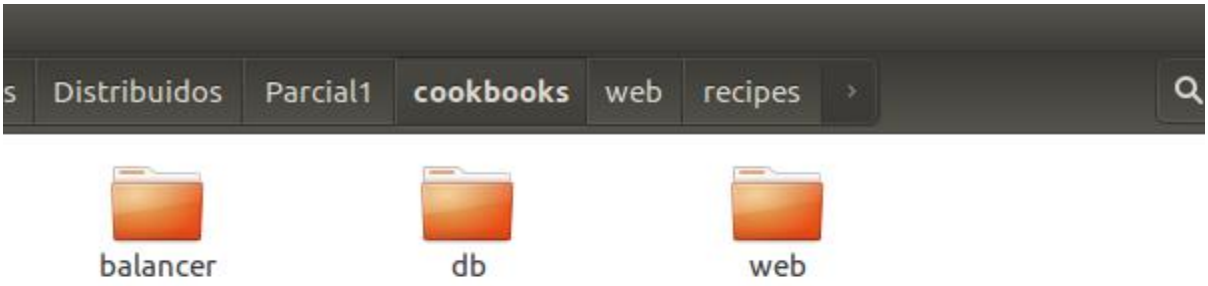
```
# -*- mode: ruby -*-
# vi: set ft=ruby :

VAGRANTFILE_API_VERSION = "2"

Vagrant.configure(VAGRANTFILE_API_VERSION) do |config|
  config.ssh.insert_key = false

  config.vm.define :centos_balancer do |balancer|
    balancer.vm.box = "centos64"
    balancer.vm.network :private_network, ip: "192.168.133.13"
    balancer.vm.network "public_network", bridge:  "eno1", ip:"192.168.131.85"
    balancer.vm.provider :virtualbox do |vb|
      vb.customize ["modifyvm", :id, "--memory", "1024","--cpus", "1", "--name",
"centos_balancer"]
    end
    config.vm.provision :chef_solo do |chef|
      chef.cookbooks_path = "cookbooks"
      chef.add_recipe "balancer"
```

```
  end
 end

 config.vm.define :centos_web1 do |web|
  web.vm.box = "Centos64Updated"
  web.vm.network :private_network, ip: "192.168.133.10"
  web.vm.network "public_network", bridge:  "eno1", ip:"192.168.131.82"
  web.vm.provider :virtualbox do |vb|
   vb.customize ["modifyvm", :id, "--memory", "1024","--cpus", "1", "--name", "centos-
web1" ]
  end
  config.vm.provision :chef_solo do |chef|
   chef.cookbooks_path = "cookbooks"
   chef.add_recipe "web"
   chef.json ={"web" => {"idserver" => "1"}}
  end
 end

 config.vm.define :centos_web2 do |web|
  web.vm.box = "Centos64Updated"
  web.vm.network :private_network, ip: "192.168.133.11"
  web.vm.network "public_network", bridge:  "eno1", ip:"192.168.131.83"
  web.vm.provider :virtualbox do |vb|
   vb.customize ["modifyvm", :id, "--memory", "1024","--cpus", "1", "--name", "centos-
web2" ]
  end
  config.vm.provision :chef_solo do |chef|
   chef.cookbooks_path = "cookbooks"
   chef.add_recipe "web"
   chef.json ={"web" => {"idserver" => "2"}}
  end
 end

 config.vm.define :centos_db do |db|
  db.vm.box = "Centos64Updated"
  db.vm.network :private_network, ip: "192.168.133.12"
  db.vm.network "public_network", bridge:  "eno1", ip:"192.168.131.84"
  db.vm.provider :virtualbox do |vb|
   vb.customize ["modifyvm", :id, "--memory", "1024","--cpus", "1", "--name", "centos-
db" ]
  end
```

```
    config.vm.provision :chef_solo do |chef|
      chef.cookbooks_path = "cookbooks"
      chef.add_recipe "db"
    end
  end

end
```

De este archivo se puede mencionar que se define el nombre de la base de datos en cada caso (para base de datos db= centos_db) en la cual se crea una máquina con el box Centos64Updated y se especifica la dirección ip de la máquina junto con la dirección e interfaz con la cual se hará puente para poder tener salida a la red del laboratorio (pool de direcciones asignada en clase) y se customiza con cada parámetro (recursos de la máquina) necesario. Luego se especifica en cada segmento la herramienta de aprovisionamiento que se usará, en este caso, chef, además de la ruta en la cual se encuentra el cookbook respectivo para dicha máquina junto con las carpetas básicas (attributes, files, recipes, templates) y receta para que se implemente el servicio deseado.

3. Cookbooks



```
distribuidos@redes1:~/Documentos/Distribuidos/Parcial1/cookbooks/balancer$ tree
.
├── attributes
│   └── default.rb
├── files
│   └── default
│       └── nginx.repo
├── recipes
│   ├── default.rb
│   └── installbalancer.rb
└── templates
    └── default
        └── nginx.conf.erb
```

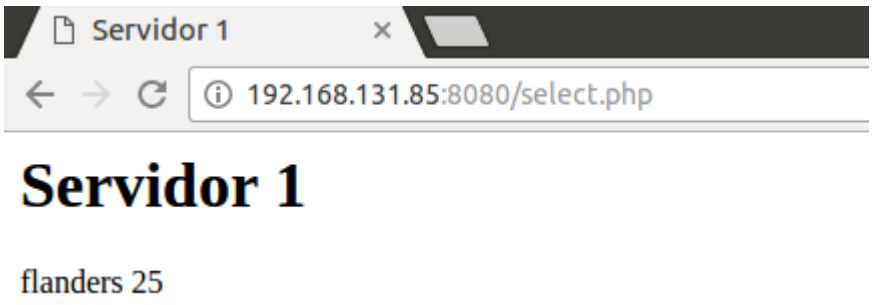
```
distribuidos@redes1:~/Documentos/Distribuidos/Parcial1/cookbooks/db$ tree
.
├── attributes
│   └── default.rb
├── files
│   └── default
├── recipes
│   ├── default.rb
│   └── installdb.rb
└── templates
    └── default
        ├── configure_mysql.sh.erb
        └── create_schema.sql.erb
```

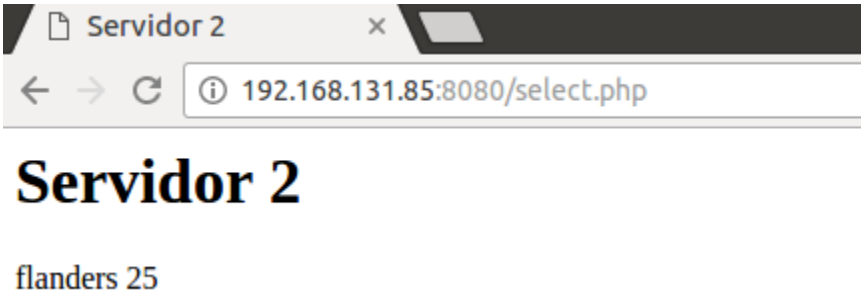
```
distribuidos@redes1:~/Documentos/Distribuidos/Parcial1/cookbooks/web$ tree
.
├── attributes
│   └── default.rb
├── files
│   └── default
│       ├── index.html
│       └── info.php
├── recipes
│   ├── default.rb
│   └── installweb.rb
└── templates
    └── default
        └── select.php.erb
```

En las anteriores imágenes se muestran los cookbooks necesarios junto con los archivos creados para poder levantar los servicios requeridos, en el repositorio de github: https://github.com/rodriesteban12/primer_parcial_distribuidos/blob/master/README.md se puede encontrar cada uno de los archivos.

4. Una vez realizado el aprovisionamiento de las máquinas, se procede a hacer vagrant up y accedemos por la ip del balanceador con el puerto 8080 que atiende las peticiones http para Nginx y el recurso select.php es decir 192.168.131.85:8080/select.php

   Al refrescar la para generar peticiones nos aparece la variación de los servidores web como se muestra en las siguientes imágenes.





5. Las dificultades encontradas en el desarrollo del proyecto fueron en primer lugar el descubrimiento de Nginx como servidor web siendo una herramienta flexible y liviano con la posibilidad de actuar como balanceador de cargas http. Una vez se encontró gracias a una referencia del profesor en clase sobre dicho servidor, se procedió a investigar e implementar, como se muestra en el proceso de aprovisionamiento anterior.
   Por otro lado, también se encontró una dificultad a la hora de mostrar el funcionamiento del balanceador de cargas, esto es, mostrar que se atiende la solicitud por el servidor 1 y luego por el 2. Lo anterior se mitigó por medio de una forma de definir un atributo desde el vagrant file en formato json para así asignar al centos_web1 el idserver 1 y al centos_web2 el idserver 2. De esta forma en el select.php.erb se define un header en html antes de hacer la consulta a la base de datos con los demás atributos definidos en “attributes”, que simplemente imprime el idserver de la máquina a la cual sea redirigida la solicitud por medio del balanceador de cargas.