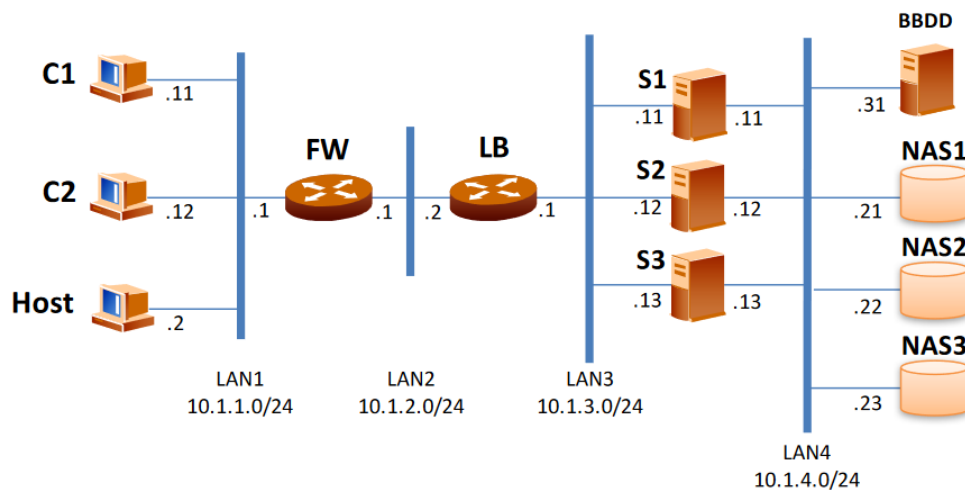


MEMORIA PRACTICA FINAL CDPS

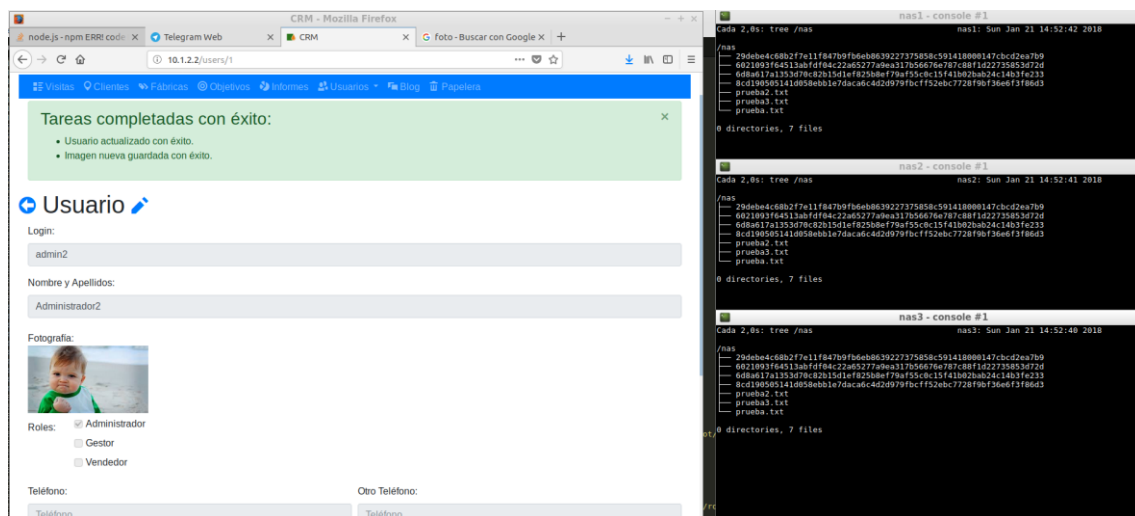
Pasos seguidos en la implementación y despliegue de la solución

Para llevar a cabo la práctica nos proporcionan la siguiente arquitectura:



El objetivo será la configuración de cada uno de los elementos de dicha arquitectura para implementar un CRM basado en el proyecto de la asignatura IWEB. Para ello habrá que configurar los siguientes elementos:

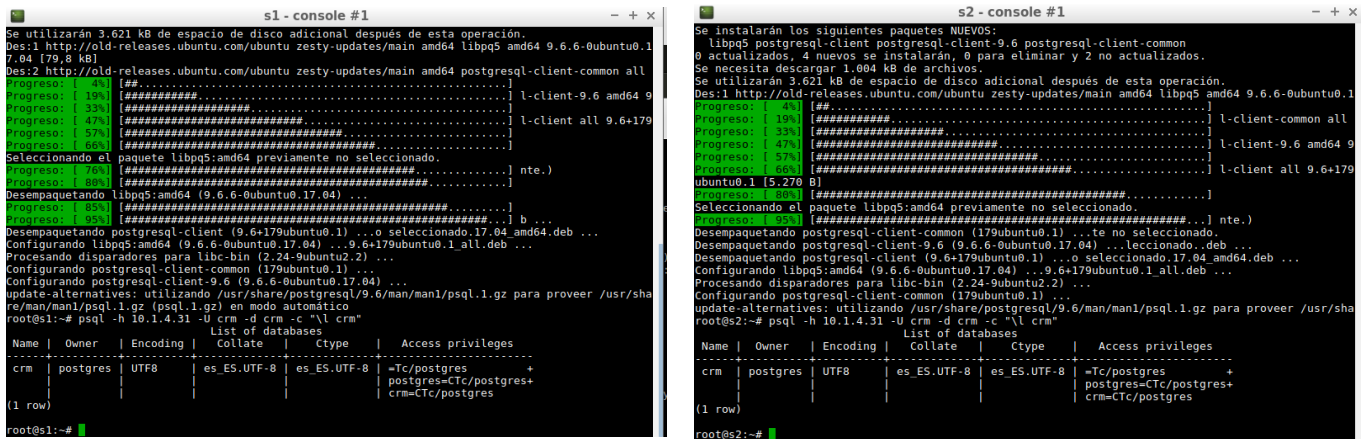
- **Cluster de almacenamiento:** Lo primero que haremos será configurar el GlusterFS, que utilizará un directorio del sistema de ficheros de la máquina virtual que se exportará y sincronizará con el resto de servidores. Su función será almacenar las imágenes del CRM y lo configuraremos para que replique la información entre los servidores nas1, nas2 y nas3.



En la imagen podemos comprobar que una vez está el sistema funcionando, cuando actualizamos la foto, se guarda dicha foto en la base de datos y se pueden ver a través de todos los nas.

- **Base de datos:**

A continuación, configuraremos la base de datos. Utilizaremos una base de datos de PostgreSQL que utilizarán los CRM.



```
s1 - console #1
Se utilizarán 3.621 kB de espacio de disco adicional después de esta operación.
Des:1 http://old-releases.ubuntu.com/ubuntu zesty-updates/main amd64 libpq5 amd64 9.6.6-0ubuntu0.17.04 [79.8 kB]
Des:2 http://old-releases.ubuntu.com/ubuntu zesty-updates/main amd64 postgresql-client-common all
Progreso: [ 4%] [#####]
Progreso: [ 19%] [#####]
Progreso: [ 33%] [#####]
Progreso: [ 47%] [#####]
Progreso: [ 57%] [#####]
Progreso: [ 66%] [#####]
Seleccionando el paquete libpq5:amd64 previamente no seleccionado.
Progreso: [ 76%] [#####]
Progreso: [ 86%] [#####]
Desempaquetando libpq5:amd64 (9.6.6-0ubuntu0.17.04) ...
Progreso: [ 85%] [#####]
Progreso: [ 92%] [#####]
Desempaquetando postgresql-client (9.6+179ubuntu0.1) ... o seleccionado.17.04 amd64.deb ...
Configurando libpq5:amd64 (9.6.6-0ubuntu0.17.04) ...9.6+179ubuntu0.1_all.deb ...
Procesando disparadores para libc-bin (2.24-9ubuntu2.2) ...
Configurando postgresql-client-common (179ubuntu0.1) ...
Configurando postgresql-client-9.6 (9.6.6-0ubuntu0.17.04) ...
update-alternatives: utilizando /usr/share/postgresql/9.6/man/man1/psql.1.gz para proveer /usr/sha
re/man/man1/psql.1.gz (psql.1.gz) en modo automático
root@s1:~# psql -h 10.1.4.31 -U crm -d crm -c "\l crm"
          List of databases
Name | Owner  | Encoding | Collate | Ctype  | Access privileges
-----+-----+-----+-----+-----+-----
crm  | postgres | UTF8      | es_ES.UTF-8 | es_ES.UTF-8 | =Tc/postgres,+
      |          |          |             |             | postgres=Ctc/postgres+
      |          |          |             |             | crm=Ctc/postgres
(1 row)

root@s1:~#

s2 - console #1
Se instalarán los siguientes paquetes NUEVOS:
libpq5 postgresql-client postgresql-client-9.6 postgresql-client-common
0 actualizados, 4 nuevos se instalarán, 0 para eliminar y 2 no actualizados.
Se necesita descargar 1.004 kB de archivos.
Se utilizarán 3.621 kB de espacio de disco adicional después de esta operación.
Des:1 http://old-releases.ubuntu.com/ubuntu zesty-updates/main amd64 libpq5 amd64 9.6.6-0ubuntu0.17.04 [79.8 kB]
Des:2 http://old-releases.ubuntu.com/ubuntu zesty-updates/main amd64 postgresql-client-common all
Progreso: [ 4%] [#####]
Progreso: [ 19%] [#####]
Progreso: [ 33%] [#####]
Progreso: [ 47%] [#####]
Progreso: [ 57%] [#####]
Progreso: [ 66%] [#####]
Seleccionando el paquete libpq5:amd64 previamente no seleccionado.
Progreso: [ 76%] [#####]
Progreso: [ 86%] [#####]
Desempaquetando libpq5:amd64 (9.6.6-0ubuntu0.17.04) ...
Progreso: [ 85%] [#####]
Progreso: [ 92%] [#####]
Desempaquetando postgresql-client (9.6+179ubuntu0.1) ... o seleccionado.17.04 amd64.deb ...
Configurando libpq5:amd64 (9.6.6-0ubuntu0.17.04) ...9.6+179ubuntu0.1_all.deb ...
Procesando disparadores para libc-bin (2.24-9ubuntu2.2) ...
Configurando postgresql-client-common (179ubuntu0.1) ...
Configurando postgresql-client-9.6 (9.6.6-0ubuntu0.17.04) ...
update-alternatives: utilizando /usr/share/postgresql/9.6/man/man1/psql.1.gz para proveer /usr/sha
re/man/man1/psql.1.gz (psql.1.gz) en modo automático
root@s2:~# psql -h 10.1.4.31 -U crm -d crm -c "\l crm"
          List of databases
Name | Owner  | Encoding | Collate | Ctype  | Access privileges
-----+-----+-----+-----+-----+-----
crm  | postgres | UTF8      | es_ES.UTF-8 | es_ES.UTF-8 | =Tc/postgres,+
      |          |          |             |             | postgres=Ctc/postgres+
      |          |          |             |             | crm=Ctc/postgres
(1 row)

root@s2:~#
```

En las imágenes vemos como en s1 y desde s2 (también ocurre en s3) detectan el correcto funcionamiento de la base de datos.

- **Los servidores CRM:**

Continuaremos configurando s1, s2 y s3. Debemos configurarlos para que den soporte a la aplicación CRM. Para ello instalaremos node, npm, git y arrancaremos el servidor. Node.js es un entorno de ejecución para JavaScript y npm es el gestor de paquetes de Javascript.

Aquí también configuraremos la variable de entorno DATABASE_URL para que se pueda acceder a la base de datos.



- **Balanceador de tráfico:**

El siguiente paso será configurar el balanceador de tráfico. Su objetivo será distribuir la carga entre los tres servidores que soportan el CRM, utilizando un algoritmo de round-robin.

The image displays two screenshots of the 'XR Status Overview' web interface, accessed via Mozilla Firefox. The browser's address bar shows the URL '10.1.2.2:8001'.

Top Screenshot: Detailed Status and Quick Overview

Detailed Status

Server 0:80

Status: Accepting connections, 0 concurrent client(s), 3 defined back ends

Dispatch mode: round-robin

Type: tcp

Checks:

- Wakeup interval: 5 sec
- Checkup interval: off

Timeouts:

- Client read: 30 sec
- Client write: 5 sec
- Back end read: 30 sec
- Back end write: 3 sec
- DNS cache validity: 3600 sec

Fast sockets closing: eliminates TIME_WAIT state

Debugging:

- Verbose logging: no
- Debug logging: no
- Traffic log directory:

Activity scripts:

- Onstart command:
- Onend command:
- Onfail command:

Network buffer size: 2048 bytes

DOS Protection:

- Max. connections: 0 (maximum value (0 for unlimited))
- Sample duration: 1 sec
- Hard max connection rate: 0 sessions per sample (0 for unlimited)
- Soft max connection rate: 0 sessions per sample (0 for unlimited)

Quick Overview

Back end 10.1.3.11:3000 up, alive, available, 0 connections

Back end 10.1.3.12:3000 up, alive, available, 0 connections

Back end 10.1.3.13:3000 up, alive, available, 0 connections

Activity

Thread	Description	Back end	Duration
140420949501696	Checkup thread		63.5306
140420957894400	Wakeup thread		63.5307
140420966287104	Web interface		63.531

Bottom Screenshot: Backend Configuration

Back end 10.1.3.11:3000

State:

- Health: alive, available
- Connections: 0
- Connect failures: 0
- Served: 101541 bytes, 1 clients

Options:

- Weight: 1
- Max. connections: 0 (unlimited)
- Load average: 0.1
- Backend check (. to reset): connect..
- Up state: yes
- Stop all connections: Stop now

Back end 10.1.3.12:3000

State:

- Health: alive, available
- Connections: 0
- Connect failures: 0
- Served: 58159 bytes, 1 clients

Options:

- Weight: 1
- Max. connections: 0 (unlimited)
- Load average: 0.1
- Backend check (. to reset): connect..
- Up state: yes
- Stop all connections: Stop now

Back end 10.1.3.13:3000

State:

- Health: alive, available
- Connections: 0
- Connect failures: 1
- Served: 0 bytes, 1 clients

Options:

- Weight: 1
- Max. connections: 0 (unlimited)
- Load average: 0.1
- Backend check (. to reset): connect..
- Up state: yes
- Stop all connections: Stop now

Aquí podemos ver su interfaz para configurarlo.

The screenshot shows the 'XR Status Overview' page in a web browser. The page displays various configuration options and status information for the XR service. On the right, a terminal window titled 'lb-console #1' shows the output of the 'xr --help' command, displaying various command-line options and their descriptions.

Balanceando todos los servidores.

The screenshot shows the 'Access Control Lists' page in a web browser. The page displays a table of ACLs with columns for Name, Action, and Status. On the right, a terminal window titled 'lb-console #1' shows the output of the 'xr --help' command, displaying various command-line options and their descriptions.

The screenshot shows the 'CRM' application interface in a web browser. The page displays a list of CRM records with columns for Name, Action, and Status. On the right, a terminal window titled 'lb-console #1' shows the output of the 'xr --help' command, displaying various command-line options and their descriptions.

Al bloquear s1 (10.1.3.11) en el balanceador, vemos que s1 ya no recibe tráfico del balanceador para el CRM.

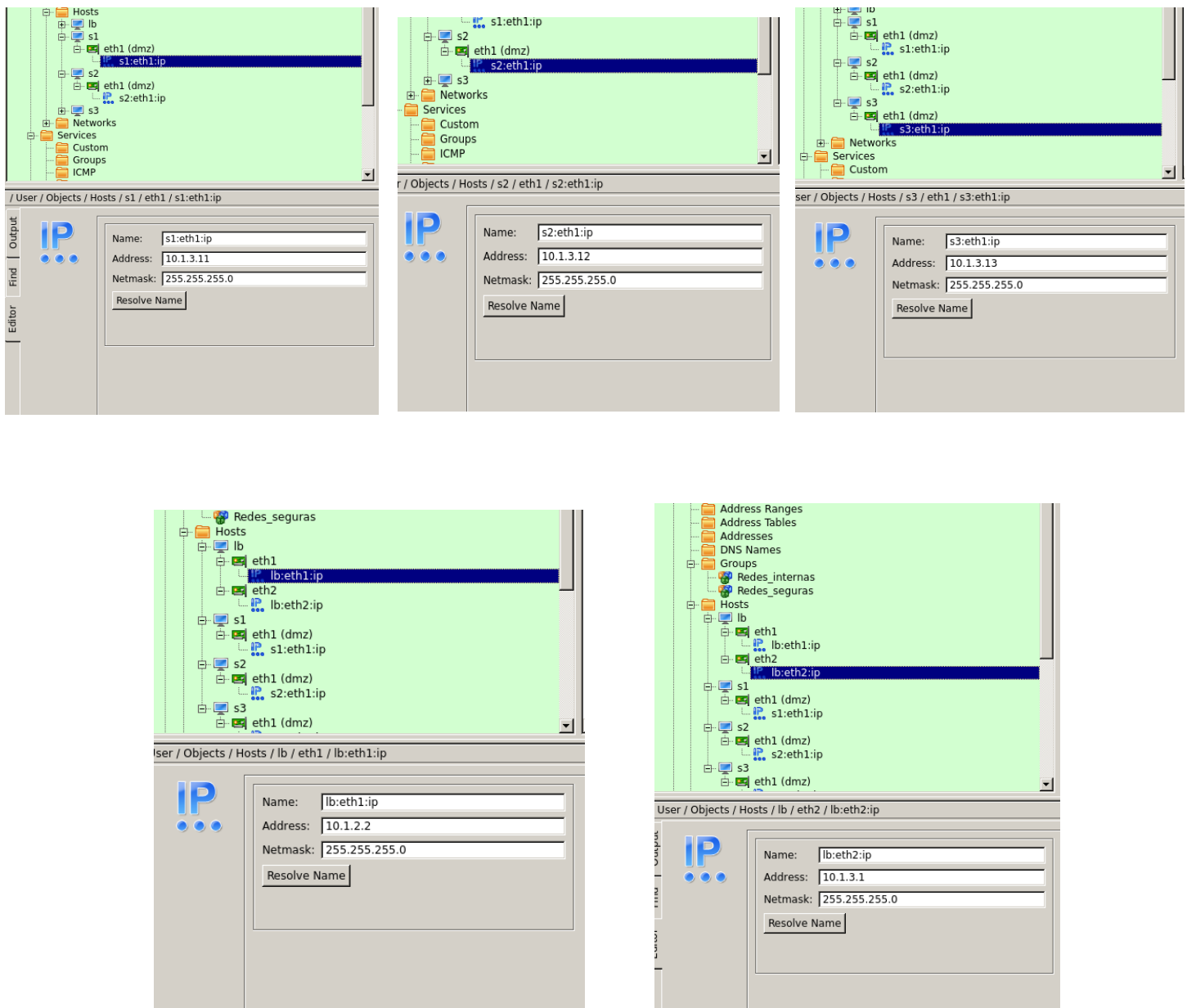
- **Firewall:**

Lo último que configuraremos será el cortafuegos. Debemos configurar el cortafuegos para filtrar el tráfico proveniente de Internet y dejar pasar el destinado a la aplicación. Por tanto, únicamente permitiremos acceso mediante ping y al puerto 80 de TCP de la dirección del balanceador de tráfico.

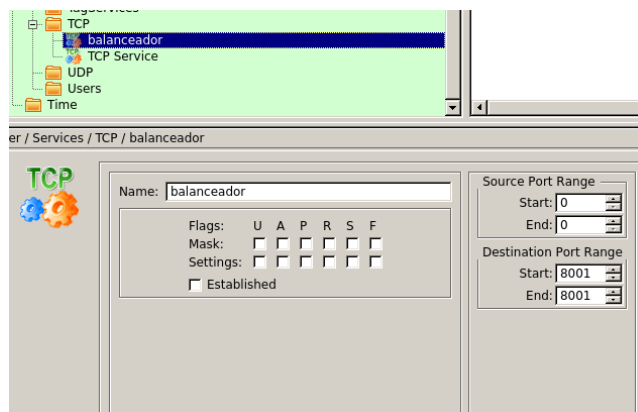
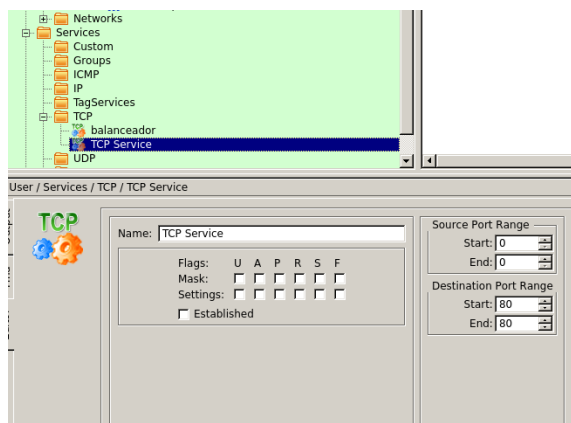
Para generar el ejecutable con la lógica del cortafuegos utilizamos la interfaz gráfica fwbuilder mediante los siguientes comandos:

```
ssh -X root@fw ; fwbuilder fw.fwb &
```

Una vez accedemos a ella estableceremos en el cortafuegos s1 (ya venía configurada), s2, s3 y lb con sus correspondientes IPs y eth en el programa del firewall.



También crearemos los servicios que vamos a utilizar:

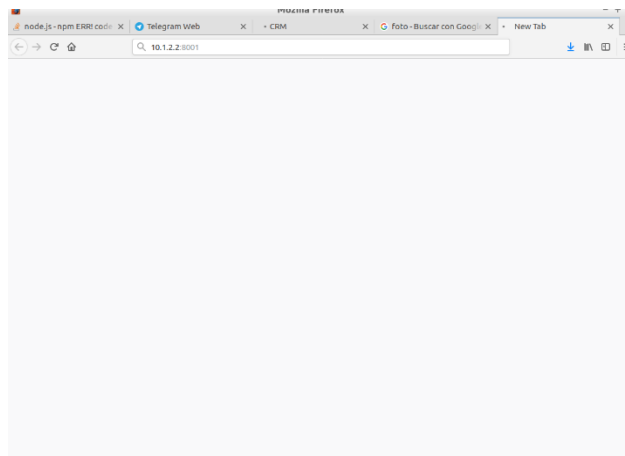


El del balanceador no es necesario incluirlo en la tabla posteriormente, pero lo hemos incluido por si en algún momento nos interesa habilitar que se pueda acceder a la configuración del balanceador.

Así quedaría la tabla del cortafuegos de paquetes:

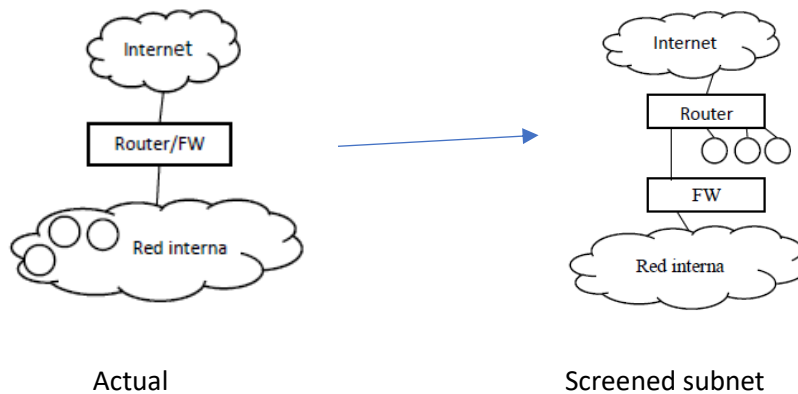
	Source	Destination	Service	Interface	Direction	Action	Time	Options	Comment
0	Any	lb	TCP Service Useful_ICMP	Any	Both	Accept	Any	log	Permitimos el acceso a icmp y tcp para el balanceador
1	Any	s1 s2 s3	http	Any	Both	Accept	Any	log	Permite acceso al servidor web de s1,s2 y s3
2	Any	Any	any ICMP	Any	Both	Accept	Any	log	Deja pasar los mensaje ICMP a traves del firewall
3	fw	fw	ssh	Any	Both	Accept	Any	log	Permite el acceso a la gestión del firewall desde las redes seguras (no borrar o deshabilitar, ya que se pierde el acceso)
4	fw	Any	X11	Any	Both	Accept	Any	log	Permite el acceso desde el fw a cualquier otro sistema
5	Red Interna	Any	ssh http	Any	Both	Accept	Any	log	Permite el acceso desde la red interna a la DMZ por ssh y http
6	Any	Any	Any	Any	Both	Deny	Any	log	Prohibe cualquier otro tráfico

Una vez activado el firewall vemos que funciona ya que nos deja acceder a la dirección del balanceador, pero no a la interfaz para su configuración.



Puntos débiles de la arquitectura

Uno de los puntos débiles que podemos observar en la arquitectura utilizada respecto a fiabilidad, es que al utilizar un cortafuegos de filtro de paquetes como es nuestro caso situado en esa zona de la arquitectura, si consiguieran acceder a los servidores, tendrían acceso a la red interna y podrían por ejemplo borrar los usuarios de la base de datos. Para solucionarlo podríamos utilizar una arquitectura screened subnet, en la que se establecen dos zonas en la arquitectura. Por un lado, colocaremos los servidores, y por otro la red interna, interconectadas por el firewall.



Respecto a la escalabilidad, la utilización de únicamente tres servidores puede suponer un problema para el firewall y para los servidores si el CRM maneja cantidades crecientes de trabajo. La solución a este problema consistiría en aumentar en número de servidores, así como de nas.

Descripción del script de configuración

Para llevar a cabo la configuración de la arquitectura necesitamos un script en Python (pfinalp2.py) y el ejecutable del cortafuegos (fw.fw). El script de configuración se compone de 5 métodos principales:

- **Cluster de almacenamiento:** glusterconf()

Lo primero que tenemos que hacer es añadir los servidores al cluster, que se realizará mediante las ordenes:

```
"gluster peer probe 10.1.4.22"
```

```
"gluster peer probe 10.1.4.23"
```

ya que esas direcciones son las de nas2 y nas3, y las ejecutamos desde nas1. (Por el problema al crear el volumen gluster que surgió damos esos mismos comandos en nas2 y nas3, con las direcciones de los otros).

Más adelante crearemos el volumen con los 3 servidores, ejecutando en nas1 el comando:

“gluster volume create nas replica 3 transport tcp 10.1.4.21:/nas 10.1.4.22:/nas 10.1.4.23:/nas force”

y posteriormente lo arrancaremos con: *“gluster volume start nas”*

Para finalizar, cambiaremos el valor del timeout en todos los servidores nas1, nas2, nas3 para agilizar la recuperación del volumen ante caídas de algún servidor con:

“gluster volumen set nas network.ping-timeout 5”

```
def glusterconf():
    logger.debug("Entramos al metodo glusterconf")
    #ANADIMOS LOS SERVIDORES GLUSTER
    line="sudo lxc-attach --clear-env -n nas1 -- gluster peer probe 10.1.4.22"
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n nas1 -- gluster peer probe 10.1.4.23"
    call(line, shell=True)

    #SOLUCION PROPIA AL PROBLEMA GLUSTER. CONSULTADO CON DAVID
    line="sudo lxc-attach --clear-env -n nas2 -- gluster peer probe 10.1.4.21"
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n nas2 -- gluster peer probe 10.1.4.23"
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n nas3 -- gluster peer probe 10.1.4.21"
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n nas3 -- gluster peer probe 10.1.4.22"
    call(line, shell=True)

    #####
    #CREAMOS EL VOLUMEN CON LOS TRES SERVIDORES NAS
    line="sudo lxc-attach --clear-env -n nas1 -- gluster volume create nas replica 3 transport tcp 10.1.4.21:/nas 10.1.4.22:/nas 10.1.4.23:/nas force "
    call(line, shell=True)
    #ARRANCAMOS EL VOLUMEN
    line="sudo lxc-attach --clear-env -n nas1 -- gluster volume start nas"
    call(line, shell=True)
    #PARA AGILIZAR LA RECUPERACION DEL VOLUMEN ANTE POSIBLES CAIDAS DE UNO DE LOS SERVIDORES
    line="sudo lxc-attach --clear-env -n nas1 -- gluster volume set nas network.ping-timeout 5"
    call(line, shell=True)
    #####
```

- **Base de datos: bddconf()**

Para instalar y configurar (configuración de usuario, contraseña,... como podemos leer en los comentarios del código) la base de datos ejecutaremos los siguientes pasos:

```
#El metodo bddconf() se encarga de la configuracion de la Base de DATOS
def bddconf():
    logger.debug("Entramos al metodo bddconf")
    #INSTALAMOS POSTGRESOL
    line="sudo lxc-attach --clear-env -n bbdd -- apt update"
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n bbdd -- apt -y install postgresql"
    call(line, shell=True)
    #FIJA LA ESCUCHA DE LA BASE DE DATOS PARA ATENDER PETICIONS EN LA DIRECION 10.1.4.31
    line="sudo lxc-attach -n bbdd -- bash -c \"echo 'listen_addresses='\"'10.1.4.31'\"'\"' >> /etc/postgresql/9.6/main/postgresql.conf \""
    call(line, shell=True)
    #ANADIMOS A LA LISTA DE CONFIANZA TODAS LOS EQUIPOS DE LA LAN 10.1.4.0/24
    line="sudo lxc-attach --clear-env -n bbdd -- bash -c \"echo 'host all all 10.1.4.0/24 trust' >> /etc/postgresql/9.6/main/pg_hba.conf\""
    call(line, shell=True)
    #CONFIGURAMOS LA BASE DE DATOS DANDOLE UN NOMBRE UN USUARIO CON TODOS SUS PRIVILEGIOS Y UNA CONTRASEÑA PARA DICHO USUARIO
    line="sudo lxc-attach --clear-env -n bbdd -- bash -c \"echo 'CREATE USER crm with PASSWORD '\"'xxxx'\"';' | sudo -u postgres psql \""
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n bbdd -- bash -c \"echo 'CREATE DATABASE crm;' | sudo -u postgres psql\""
    call(line, shell=True)
    line="sudo lxc-attach --clear-env -n bbdd -- bash -c \"echo 'GRANT ALL PRIVILEGES ON DATABASE crm to crm;' | sudo -u postgres psql\""
    call(line, shell=True)
    #REINICIAMOS LA BASE DE DATOS CON LAS CONFIGURACIONES YA FIJADAS.
    line="sudo lxc-attach --clear-env -n bbdd -- systemctl restart postgresql"
    call(line, shell=True)
    #####
```


- **CRM: crmconf()**

En este apartado debemos instalar todo lo necesario para que funcione la aplicación de los CRM en los servidores. Para ello instalamos git, curl y nodejs. A continuación, haremos el clone del CRM.

Continuaremos instalando el npm install, y el npm install forever, que utilizaremos para arrancar el servidor en background. Para ello, mediante la opción de lxc attach:

–set-var=postgres://crm:xxxx@10.1.4.31:5432/crm, configuramos la variable de entorno de acuerdo a la configuración previa de la base de datos.

Para finalizar, aplicaremos las migraciones y ejecutaremos el seeder (solo en s1), y arrancaremos el CRM con start.

```
def crmconf():
    logger.debug("Entramos al metodo crmconf")
    #PARA TODOS LOS SERVIDORES
    for i in [1,2,3]:
        #INSTALAMOS LOS PAQUETES: GIT, CURL, NODE
        print(str(i)+"\n\n")
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- apt-get update"
        call(line, shell=True)
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- apt-get install git -y"
        call(line, shell=True)
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- sudo apt-get install curl -y"
        call(line, shell=True)
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- bash -c \" curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -; sudo apt-get install nodejs\""
        call(line, shell=True)
        #DESCARGAMOS EL CRM
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- git clone https://github.com/CORE-UPM/CRM_2017"
        call(line, shell=True)
        #HACIENDO EL NPM INSTALL Y CON LA UTILIDAD FOREVER LO DEJAMOS EN BACKGROUND
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- set-var 'DATABASE_URL=postgres://crm:xxxx@10.1.4.31:5432/crm' -- bash -c \" cd CRM_2017; npm install; npm install forever\""
        call(line, shell=True)
        #CONFIGURAMOS LA CARPETA DE SUBIDAS DE FOTOS
        #CREAMOS LA CARPETA
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- mkdir CRM_2017/public/uploads"
        call(line, shell=True)
        #LA ASIGNAMOS A LA NAS
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- mount -t glusterfs 10.1.4.21:/nas CRM_2017/public/uploads"
        call(line, shell=True)
        #SOLO EN EL SERVIDOR 1 HACEMOS EL MIGRATE Y EL SEED LOCAL
        if i == 1:
            line="sudo lxc-attach --clear-env -n s1 -- set-var 'DATABASE_URL=postgres://crm:xxxx@10.1.4.31:5432/crm' -- bash -c \" cd CRM_2017; npm run-script migrate_local; npm run-script seed_local\""
            call(line, shell=True)
        #ARRANCAMOS LA APLICACION EN BACKGROUND
        line="sudo lxc-attach --clear-env -n s"+str(i)+" -- set-var 'DATABASE_URL=postgres://crm:xxxx@10.1.4.31:5432/crm' -- bash -c \" cd CRM_2017; ./node_modules/forever/bin/forever start ./bin/www\""
        call(line, shell=True)
```

- **Balanceador: lbconf()**

Configuramos el balanceador indicándole el puerto (80) por donde veremos la aplicación del CRM una vez balanceada, le indicaremos los servidores donde puede escuchar (10.1.3.11:3000, 10.1.3.12:3000 y 10.1.3.13:3000), y el puerto donde estará la interfaz web para manejar el balanceador (8001). Esto se realizará en background.

```
def lbconf():
    logger.debug("Entramos al metodo lbconf")
    #ASIGNA AL PUERTO 80 DEL BALANCEADOR LO QUE SE VERIA EN EL 3000 DE LOS SERVIDORES. LA INTERFAZ PARA CONFIGURAR EL BALANCEADOR SE DEJA EN EL PUERTO 8001.
    line="sudo lxc-attach --clear-env -n lb -- bash -c \"xr -S -dr --server tcp:0:80 --backend 10.1.3.11:3000 --backend 10.1.3.12:3000 --backend 10.1.3.13:3000 --web-interface 0:8001\" &"
    call(line, shell=True)
    #####
```

- **Firewall: fwconf()**

Lo que haremos en este método será copiar el archivo fw.fw previamente realizado a la máquina del cortafuegos, y ejecutarlo.

```
def fwconf():
    logger.debug("Entramos al metodo fwconf")
    #COPIA EL ARCHIVO .FW DENTRO DEL FIREWALL DEL ESCENARIO
    line="sudo cp fw.fw /var/lib/lxc/fw/rootfs/root"
    call(line, shell=True)
    #EJECUTA EL ARCHIVO .FW HACIENDO VIGENTES LAS NORMAS
    line="sudo lxc-attach --clear-env -n fw -- /root/fw.fw"
    call(line, shell=True)
```

- Esqueleto, arrancar y destruir

Para arrancar el escenario y poder probar los métodos individualmente.

```
def arrancar():  
    logger.debug("Arrancamos el escenario")  
    os.system("sudo vnx -f /home/cdps/pfinal/pfinal.xml --create")
```

```
def destruir():  
    logger.debug("Destruimos el escenario")  
    os.system("sudo vnx -f /home/cdps/pfinal/pfinal.xml --destroy")
```

```
f = sys.argv  
tamanoArgumentos = len(f)  
  
if tamanoArgumentos > 1:  
    metodo = f[1]  
    logger.debug("El argurmento 1 es :"+ str(metodo))  
  
    if metodo == "fw-conf":  
        fwconf()  
    elif metodo == "bdd-conf":  
        bddconf()  
    elif metodo == "gluster-conf":  
        glusterconf()  
    elif metodo == "crm-conf":  
        crmconf()  
    elif metodo == "lb-conf":  
        lbconf()  
  
    elif metodo == "arrancar":  
        arrancar()  
  
    elif metodo == "total":  
        total()  
  
    elif metodo == "destruir":  
        destruir()  
  
    elif metodo == "ayuda":  
        print("\n\n##### AYUDA #####\n")  
  
        print("* 'python pfinalp2.py' ejecuta diferentes funciones segun la opcion que se añade detras: \n ")  
        print("* 'python pfinalp2.py fw-conf' configura el FIREWALL\n")  
        print("* 'python pfinalp2.py total' arranca todo el escenario \n ")  
        print("* 'python pfinalp2.py bdd-conf' configura la base de datos\n")  
        print("* 'python pfinalp2.py gluster-conf' configura el GLUSTER\n")  
        print("* 'python pfinalp2.py lb-conf' configura el BALANCEADOR\n")  
        print("* 'python pfinalp2.py crm-conf' configura el CRM\n")  
        print("\n\n##### FIN DE LA AYUDA #####\n\n")  
  
    else:  
        print "Las opciones son erroneas. Introduzca 'python pfinalp2.py ayuda' para mas informacion"
```

```
def total():  
    logger.debug("Todas las configuraciones de golpe")  
    arrancar()  
    glusterconf()  
    bddconf()  
    crmconf()  
    lbconf()  
    fwconf()
```

Para arrancar el escenario y realizar toda la configuración necesaria bastará con ejecutar la función total() del script pfinalp2.py.

Despliegue en Openstack, Amazon AWS o Google Cloud

Para el despliegue en OpenStack, debemos configurar sus componentes en concordancia con la arquitectura. Utilizaremos las herramientas que nos proporcione OpenStack para crear dicho escenario.

Una forma de realizarlo sería introduciendo mediante el elemento Nova una máquina virtual por cada elemento de la arquitectura, lo cual aportaría mayor capacidad de computación. Esta opción sería la más adecuada si quisiéramos que esta aplicación fuera usada por un gran número de usuarios. Al ser la práctica de carácter didáctico podríamos utilizar simplemente una máquina virtual que contenga el resto de servicios virtualizados. A través del dashboard (Horizon), podríamos monitorizar el funcionamiento de cada una de las máquinas virtuales que componen el escenario. Si quisiéramos monetizar el servicio (aunque no es el objetivo) podríamos utilizar Ceilometer, que dispone de las herramientas necesarias para llevar esto a cabo.

Para realizar el proceso en Amazon AWS o Google Cloud, habría que llevar a nuestro terminal de la nube el conjunto de elementos de los que disponemos en local (escenario proporcionado pfinal.tgz, script, fw.fw) y realizar el despliegue.