

Universidad de San Carlos de Guatemala

Centro Universitario de Occidente

Ingeniería en Ciencias y Sistemas

Redes de computadoras 2

Ing. Francisco Rojas

Aux. Manuel Rojas



Proyecto final

202030028 Yefri Marconi Gonzalez Vicente

201731370 Jose Luis Baltazar Pu Sam

201931804 Marco Jose Munguia Alva

201931806 Byron Javier VAsquez Villagran

201930171 Melanni del Rosario Tzul Baquiaux

Índice

Marco teórico.....	3
Debian GNU/Linux.....	3
iptables.....	3
Interfaz de Red.....	3
Qemu.....	3
Router.....	4
Topología.....	5
Cálculo de máscaras de red.....	6
Cálculo y asignación de direcciones IP y máscaras de red.....	6
Firewall.....	6
Balanceador.....	6
Proxy.....	7
Configuraciones.....	7
Firewall.....	7
Configuración de Interfaces con nombres persistentes.....	7
Activar el reenvío de paquetes.....	8
Reglas de iptables.....	8
Configuración de interfaces.....	8
Balanceador de carga.....	10
bandwidth.....	10
LB_rules.conf.....	10
Script de balanceador.....	11
Script zabbix.....	14
Interfaces.....	14
Proxy.....	15
Configuración de interfaces.....	15
Configuración de nft.....	18
Router.....	19
Intefaces de red.....	19
Script para aplicar configuración de vlans.conf.....	20
Script de configuracion para acceso mac e ip.....	22
Virtualizador.....	27
Admin.....	27
Clientes.....	29
Zabbix.....	31
Wordpress.....	34

Marco teórico

Debian GNU/Linux

Debian es una distribución de software libre para sistemas de ordenadores con diferentes arquitecturas de hardware. Debido a que el paquete completo hace uso, por una parte, del núcleo de Linux desarrollado por Linus Torvalds y, por otra, de las herramientas básicas del sistema GNU, también es conocido como Debian GNU/Linux. Actualmente, Debian sigue siendo desarrollado como proyecto comunitario por más de 1000 desarrolladores oficiales alrededor del mundo. Como una de las colecciones más utilizadas y antiguas. Debian cuenta con más de 43000 paquetes de software listos para usar y es considerado, por lo tanto, como una solución universal de sistema operativo. Por otra parte, Debian es una de las distribuciones más influyentes y sirve de base para muchas distribuciones nuevas, como por ejemplo el popular Ubuntu. Adicionalmente, desde Debian 6.0 se ha publicado una versión basada en un núcleo FreeBSD

iptables

iptables es una herramienta de línea de comandos utilizada en sistemas operativos Linux para configurar, mantener y examinar las reglas del cortafuegos (firewall) del kernel a través del subsistema netfilter.

Permite controlar el tráfico de red que entra, sale o pasa a través del sistema, basándose en reglas definidas por el usuario.

Estas reglas pueden:

- Permitir (ACCEPT),
- Denegar (DROP),
- Rechazar (REJECT),
- Redireccionar o registrar tráfico según el protocolo, dirección IP, puertos y otras condiciones.

Interfaz de Red

Una interfaz de red es el software específico de red que se comunica con el controlador de dispositivo específico de red y la capa IP a fin de proporcionar a la capa IP una interfaz coherente con todos los adaptadores de red que puedan estar presentes.

Qemu

Es un emulador de procesadores basado en la traducción dinámica de binarios (conversión del código binario de la arquitectura fuente en código entendible por la arquitectura huésped). QEMU también tiene capacidades de virtualización dentro de un sistema operativo, ya sea GNU/Linux, Windows, o cualquiera de los sistemas operativos admitidos; de hecho es la forma más común de uso. Esta máquina virtual puede ejecutarse en cualquier tipo de Microprocesador o arquitectura Virtual Machine Manager El Hypervisor, también conocido como monitor de máquina virtual, es una herramienta para la creación, ejecución y gestión de máquinas virtuales. Estas máquinas virtuales son las encargadas de alojar servicios virtualizados como escritorios remotos o CPD Virtuales.

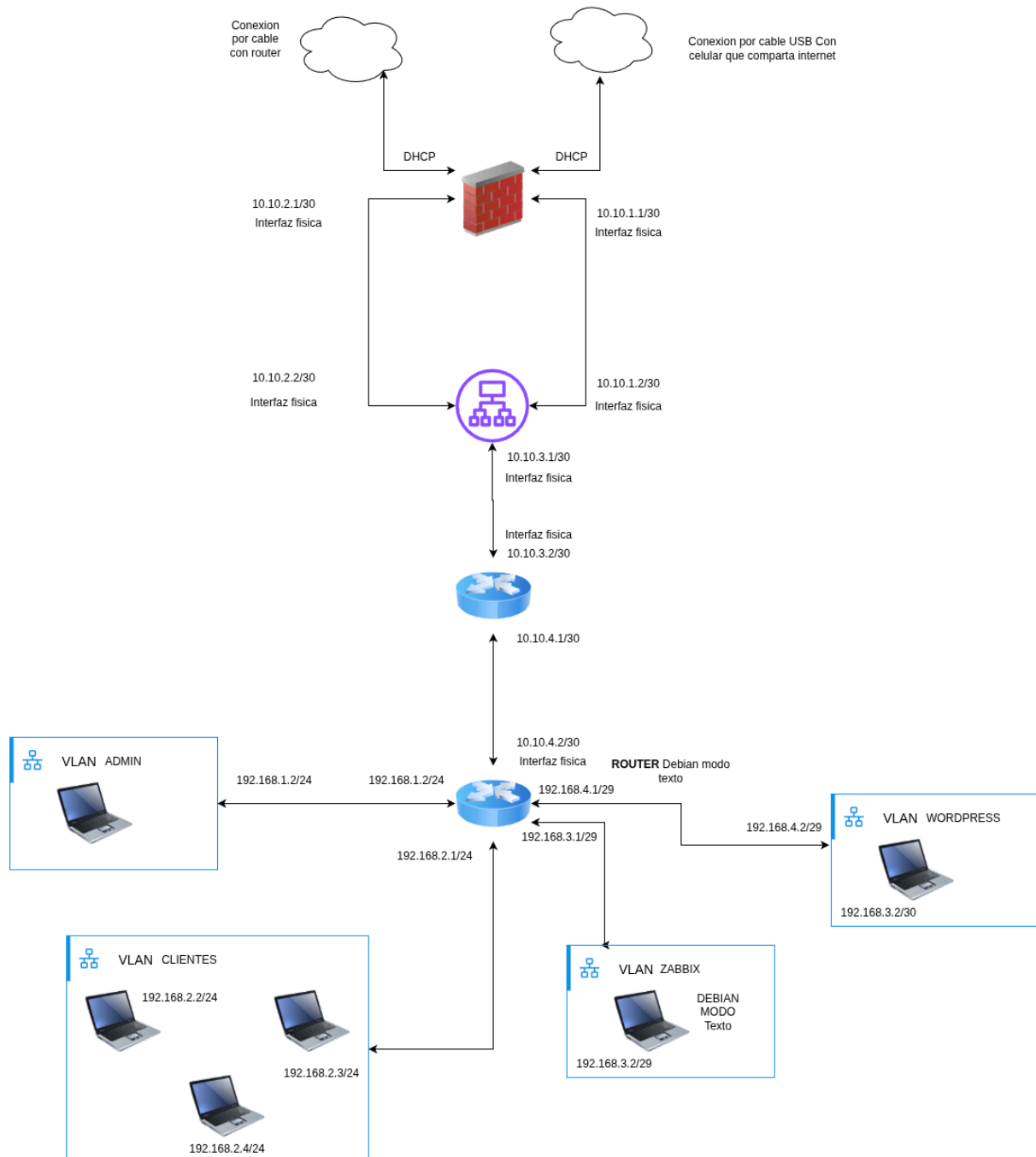
Router

Un router es un dispositivo de red que se utiliza para conectar diferentes redes y permitir que los dispositivos en cada red se comuniquen entre sí. El enrutamiento de IP es una de las funciones clave de un router. Cada dispositivo conectado a una red tiene una dirección IP única que se utiliza para identificarlo en la red. Cuando un dispositivo desea enviar un paquete de datos a otro dispositivo en una red diferente, necesita enviar el paquete a través de un router para que este pueda enrutar correctamente.

El router utiliza una tabla de enrutamiento para determinar la mejor ruta para enviar los paquetes de datos entre diferentes redes. La tabla de enrutamiento contiene información sobre las redes a las que está conectado el router y las rutas disponibles para llegar a cada red.

Cuando un paquete de datos llega al router, el router examina la dirección IP de destino del paquete y consulta su tabla de enrutamiento para determinar la mejor ruta para enviar el paquete. El router envía el paquete al siguiente dispositivo en la ruta hacia su destino, y así sucesivamente, hasta que el paquete llega a su destino final. Los routers pueden usar diferentes algoritmos de enrutamiento para determinar la mejor ruta para enviar los paquetes. Algunos de los algoritmos de enrutamiento más comunes son el enrutamiento estático, en el que las rutas se configuran manualmente en la tabla de enrutamiento, y el enrutamiento dinámico, en el que los routers intercambian información sobre las redes a las que están conectados para determinar las rutas óptimas.

Topología



https://drive.google.com/file/d/1V5O7_spaZ7ReDME67fGBrzGQ_yZfQMKJ/view?usp=sharing

Cálculo de máscaras de red

Para la máscara de red se hace uso de la notación con barra / .

Máscara: 255,255,255,0

Binario: 11111111,11111111,11111111,00000000

Partiendo de que tenemos 4 grupos de 8 bits, de los cuales los 0 representan los bits que podemos utilizar y para ver el número host disponibles tendremos el número de 0s elevado al cuadrado. En este ejemplo tenemos 8 bits 0.

Tenemos $2^8 = 256$. De los cuales en todos los casos 2 no son utilizables.

la primera .0 que es la de red

y la última, en este caso la .256 que es la broadcast.

Entonces para la máscara 255.255.255.0 tenemos 254 host disponibles.

Ya que estamos usando el formato de barra, en este se especifica el número de bits 1, que para la máscara 255.255.255.0, son 24. Por lo que tenemos una máscara de red /24.

Cálculo y asignación de direcciones IP y máscaras de red

Firewall

Interfaz	Dirección ip	máscara	cálculo
eth0	10.10.1.1	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles
eth1	10.10.2.1	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles
enp2s0	dhcp	/24	$2^{32-24} = 2^8 = 256 - 2 = 254$ host disponibles
usb0	dhcp	/24	$2^{32-24} = 2^8 = 256 - 2 = 254$ host disponibles

Balanceador

Interfaz	Dirección ip	máscara	cálculo
enx00e04c36073a	10.10.2.2	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles

enx00e04c360224	10.10.1.2	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles
enp3s0	10.10.3.1	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles

Proxy

Interfaz	Dirección ip	máscara	cálculo
enx00e04c36073a	10.10.4.1	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles
enp3s0	10.10.3.2	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles

Router

Interfaz	Dirección ip	máscara	cálculo
enx00e04c36073a	10.10.4.2	/30	$2^{32-30} = 2^2 = 4 - 2 = 2$ host disponibles
enp2s0.10	192.168.1.1	/24	$2^{32-24} = 2^8 = 256 - 2 = 254$ host disponibles
enp2s0.20	192.168.2.1	/24	$2^{32-24} = 2^8 = 256 - 2 = 254$ host disponibles
enp2s0.30	192.168.3.1	/29	$2^{32-29} = 2^3 = 8 - 2 = 6$ host disponibles
enp2s0.30	192.168.8.1	/29	$2^{32-29} = 2^3 = 8 - 2 = 6$ host disponibles

Configuraciones

Firewall

Configuración de Interfaces con nombres persistentes

- Identificar el idVendo, idProduct y serie
`udevadm info -a -p /sys/class/net/ nombreInterfaz | grep -E '{idvendo}|{idProduct}|{serial}'`
- Con la informacion que tengamos del comando anterior editamos el archivo ubicado en `etc/udev/rules.d/nombreArchivo` en caso de no tener ningun

nombre de archivo

- Luego ejecutamos el siguiente comando: `sudo udevadm control --reload-rules` y `sudo udevadm trigger`

Activar el reenvío de paquetes

- `sudo sysctl -p`

Reglas de iptables

```
#iptables para el firware
sudo iptables -F
sudo iptables -t nat -F
sudo iptables -X

sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT

# NAT para ambas salidas (WAN)
sudo iptables -t nat -A POSTROUTING -o enp2s0 -j MASQUERADE
sudo iptables -t nat -A POSTROUTING -o usb0 -j MASQUERADE

sudo iptables -A FORWARD -i eth0 -o enp2s0 -j ACCEPT
sudo iptables -A FORWARD -i eth1 -o enp2s0 -j ACCEPT
sudo iptables -A FORWARD -i eth0 -o usb0 -j ACCEPT
sudo iptables -A FORWARD -i eth1 -o usb0 -j ACCEPT
sudo iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j
ACCEPT

sudo iptables -A INPUT -p icmp -j ACCEPT
sudo iptables -A FORWARD -p icmp -j ACCEPT
```

Configuración de interfaces

```
# This file describes the network interfaces available on your
system
# and how to activate them. For more information, see
interfaces(5).

source /etc/network/interfaces.d/*
```



```
# The loopback network interface
auto lo
iface lo inet loopback

#Conexion por wifi
# The primary network interface
#allow-hotplug wlan0
#iface wlan0 inet dhcp
#    wpa-ssid FamiliaPuSam
#    wpa-psk 12345678

#Conexion fisica por cable
auto enp2s0
iface enp2s0 inet dhcp

#Conexion por cable USB
#auto usb0
allow-hotplug usb0
iface usb0 inet dhcp

#Conexion fisica Lan 1
#auto eth0
allow-hotplug eth0
iface eth0 inet static
    address 10.10.1.1
    netmask 255.255.255.252
    up ip route add 10.10.4.0/30 via 10.10.1.2 dev eth0 metric
100
    up ip route add 10.10.3.0/30 via 10.10.1.2 dev eth0 metric
100
    down ip route del 10.10.3.0/30 via 10.10.1.2 dev eth0 metric
100
    down ip route del 10.10.4.0/30 via 10.10.1.2 dev eth0 metric
100

#Conexion fisica Lan 2
#auto eth1
allow-hotplug eth1
iface eth1 inet static
    address 10.10.2.1
    netmask 255.255.255.252
    up ip route add 10.10.3.0/30 via 10.10.2.2 dev eth1 metric
```

```
200      up ip route add 10.10.4.0/30 via 10.10.2.2 dev eth1 metric
200      down ip route del 10.10.3.0/30 via 10.10.2.2 dev eth1 metric
200      down ip route del 10.10.4.0/30 via 10.10.2.2 dev eth1 metric
200
```

Balanceador de carga

bandwidth

```
isp1, 512, 512
isp2, 1000,1000
```

LB_rules.conf

```
# VLAN 1
192.168.1.0/24, [80,443], TCP, isp1
192.168.1.0/24, [22], TCP, isp2
# VLAN 2

192.168.2.0/24, [80, 443], TCP, isp1
192.168.2.0/24, [22], TCP, isp2
#VLAN3
192.168.3.0/30, [80,443], TCP, ispi
192.168.3.0/30, [22], TCP, isp2
#VLAN4
192.168.4.0/30, [80,443], TCP, isp1
192.168.4.0/30, [22], TCP, isp2
#IF1
10.10.1.0/30, [80,443], TCP, isp1
10.10.1.0/30, [22], TCP, isp2
#IF2
10.10.2.0/30, [80,443], TCP, isp1
10.10.2.0/30, [22], TCP, isp2

10.10.3.0/30, [80, 433], TCP, isp1
10.10.3.0/30, [22], TCP, ips2
10.10.4.0/30, [80,443], TCP, lisp1
10.10.4.0/30, [22], TCP, isp2
```

Script de balanceador

```
#!/bin/bash
BW_CONF="/etc/lb/bandwidth.conf"
LB_RULES="/etc/lb/LB_rules.conf"
# Gateways
GW1="10.10.1.1"
GW2="10.10.2.1"
# Interfaces de isp
IF1="enx00e04c36073a"
IF2="enx00e04c360224"
table2="isp2"
WEIGHT1=1
WEIGHT2=1

flush_rules(){
    "$(date): Eliminando reglas"
    iptables -t mangle -F PREROUTING 2>/dev/null
    ip rule del fwmark 1 table isp1 2>/dev/null
    ip rule del fwmark 2 table isp2 2>/dev/null
}

configure_rules() {
    iptables -t mangle -F prerouting
    while read -r ip ports proto isp; do
        [[ "$isp" || "$isp" =~ ^# && continue]]
        ipread=$(echo "$ip" | xargs | sed | "s/,/ /g;s/ //g" )
        portsread=$(echo "$ports" | xargs | "s/[] []//g; s/ /g" )
        portsread= "${portsread%}"
        protoread=$(echo "$proto" | xargs | tr "[:upper:]"
        "[:lower:]" | sed "s/,/ /g; s/ //g")
        ispread=$(echo "$isp" | xargs | tr "[:upper:]" "[:lower:]" |
        sed "s/,/ /g; s/ //g")

        mark=0

        if [[ "$ispread" == "$table1" ]]; then
            mark=1
        elif [[ "$ispread" == "$table2" ]]; then
            mark=2
        fi

        if [[ "$protoread" != "tcp" && "$protoread" != "udp" ]]; then
```

```

        continue
    fi
    echo "$mark"
    iptables -t mangle -A PREROUTING -s "$ipread" -p
"$protoread" -m multiport --dports $portsread -j MARK --set-mark
"$mark"
    echo "Regla cargada: $ipread -> puertos $portsread
($portsread) -> $ispread"

done < "$LB_RULES"
ip rule add fwmark 1 table isp1 2>/dev/null
ip rule add fwmark 2 table isp2 2>/dev/null
echo "Configuracion de reglas completada"
}

calculate_weights(){
    while IFS=, read -r isp up dw; do
        [[ -z "$isp" || "$isp" =~ ^# ]] && continue
        isp=$(echo "$isp" | xargs)
        up=$(echo "$up" | xargs)
        dw=$(echo "$dw" | xargs)

        weight=$((dw/512))
        [[ $weight -lt 1 ]] && weight=1
        if [[ "$isp" == "isp1" ]]
            WEIGHT1=$weight
        elif [[ "$isp" == "isp2" ]]
            WEIGHT2=$weight
        fi
    done < <(grep -v " ^#" $BW_CONF)
    echo "$(date): Pesos calculados: ISP1=$WEIGHT1 ISP2=$WEIGHT2"
}

update_default_route(){
    if [[ $1 == "both" ]]; then
        ip route del default 2>/dev/null
        ip route replace default \
            nexthop via $GW1 dev $IF1 weight $WEIGHT1\
            nexthop via $GW2 dev $IF2 weight $WEIGHT2
        echo "$(date): Balanceo activo: ISP1 peso $WEIGHT1 ISP2 peso
$WEIGHT2"
    elif [[ $1=="isp2" ]]; then

```

```

    ip route del default 2>/dev/null
    ip route replace default via $GW1 dev $IF1
    echo "$(date): Todo el trafico por ISP1"
elif [[ $1=="isp1" ]]; then
    ip route del default 2>/dev/null
    ip route replace default via $GW2 dev $IF2
    echo "$(date): Todo el trafico por ISP2"
else
    echo "$(date): Ambos IPS CAIDOS, SIN CONEXION"
fi
}
echo "Iniciando balanceador"
sysctl -p
iptables -t nat -A POSTROUTING -o enx00e04c36073a -s 10.10.3.0/30
-j MASQUERADE
iptables -t nat -A POSTROUTING -p enx00e04c360224 -s 10.10.3.0/30
-j MASQUERADE
calculate_weights

while true; do
    ping -c 2 -W 1 $GW1 > /dev/null
    STATUS1=$?
    ping -c 2 -W 1 $GW2 > /dev/null
    STATUS2=$?

    if [[ $STATUS1 -eq 0 && $STATUS2 -eq 0 ]]; then
        flush_rules
        configure_rules
        update_default_route "both"
    elif [[ $STATUS1 -eq 0 ]]; then
        flush_rules
        update_default_route "isp1"
    elif [[ $STATUS2 -eq 0 ]]; then
        flush_rules
        update_default_route "isp2"
    fi
    sleep 5
done

```

Script zabbix

```
#!/bin/bash
ZBX_SERVER="192.168.3.2"
HOSTNAME= "Load Balancer"
ZBX_PORT=80
IF1="enx00e04c36073a"
IF2="enx00e04c360224"
send_bandwidth_metris() {
    while true; do
        echo "Enviando metricas de ancho de banda a zabbix"
        if ping -c 1 -W 1 "$ZBX_SERVER"> /dev/null; then
            RX1=$(vnstat -i "$IF1" --oneline cut -d';' -f10)
            TX1=$(vnstat -i "$IF1" --oneline cut -d';' -f11)
            RX2=$(vnstat -i "$IF2" --oneline cut -d';' -f10)
            TX2=$(vnstat -i "$IF2" --oneline cut -d';' -f11)

            zabbix_sender -vv -z "$ZBX_SERVER" -s "$HOSTNAME" -k
isp1.rx.bytes -o "$RX1"
            zabbix_sender -vv -z "$ZBX_SERVER" -s "$HOSTNAME" -k
isp1.tx.bytes -o "$TX1"
            zabbix_sender -vv -z "$ZBX_SERVER" -s "$HOSTNAME" -k
isp2.rx.bytes -o "$RX2"
            zabbix_sender -vv -z "$ZBX_SERVER" -s "$HOSTNAME" -k
isp2.tx.bytes -o "$TX2"

            echo "Metricas enviadas"
            echo "ISP1 DOWN=$RX1 UP=$TX1"
            echo "ISP2 DOWN=$RX2 UP=$TX2"
        else
            echo "No hay conectividad con zabbix ($ZBX_SERVER)"
        fi
        sleep 5
    done
}
echo "===== Metricas zabbix
===== "
send_bandwidth_metris
```

Interfaces

```
source /etc/network/interfaces.d/*
# The loopback network interface
```

```

auto lo
iface lo inet loopback
# red interna hacia proxy

auto enp3s0
iface enp3s0 inet static
    address 10.10.3.1
    netmask 255.255.255.252
    #/30 solo dos dispositivos
    gateway 10.10.3.2
    up ip route add 10.10.4.0/30 via 10.10.3.2 dev enp3s0
    up ip route add 192.168.2.0/24 via 10.10.3.2 dev enp3s0
    down ip route del 10.10.4.0/30 via 10.10.3.2 dev enp3s0
    down ip route del 192.168.2.0/24 via 10.10.3.2 dev enp3s0

# ISP1
auto enx00e04c36073a
iface enx00e04c36073a inet static
    address 10.10.1.2
    netmask 255.255.255.252
    #/30 solo dos dispositivos
    gateway 10.10.1.1

# ISP2
auto enx00e04c360224
    iface enx00e04c360224 inet static
    address 10.10.2.2
    netmask 255.255.255.252
    # /30 solo dos dispositivos
    gateway 10.10.2.1

```

Proxy

Configuración de interfaces

```

# This file describes the network interfaces available on your
system
# and how to activate them. For more information, see
interfaces(5).

source /etc/network/interfaces.d/*

```

```
# The loopback network interface
auto lo
iface lo inet loopback

auto enp58s0
    iface enp58s0 inet static
    address 10.10.3.2
    netmask 255.255.255.252
    gateway 10.10.3.1

auto enx00e04c3603e7
    auto enx00e04c3603e7
    iface enx00e04c3603e7 inet static
    address 10.10.4.1
    netmask 255.255.255.252
    gateway 10.10.4.2
    up ip route add 192.168.0.0/16 via 10.10.4.2 dev
enx00e04c3603e7
    down ip route del 192.168.0.0/16 via 10.10.4.2 dev
enx00e04c3603e7
    up ip route add 10.10.4.0/24 via 10.10.4.2 dev
enx00e04c3603e7
    down ip route del 10.10.4.0/24 via 10.10.4.2 dev
enx00e04c3603e7
```

Configuración Squid.conf

```
visible_hostname proxy.local
cache_log /var/log/squid/cache.log
access_log /var/log/squid/access.log squid

acl DOMINIOS_GOBIERNO dstdomain "/etc/squid/lists/gov.acl"

cache_dir ufs /var/spool/squid 100 16 256
maximum_object_size_in_memory 512 KB
cache_mem 256 MB
maximum_object_size 64 MB

http_port 3127
```



```
http_port 3128 intercept
https_port 3129 intercept ssl-bump
cert=/etc/squid/ssl_cert/myCA.pem key=/etc/squid/ssl_cert/myCA.key
generate-host-certificates=on dynamic_cert_mem_cache_size=4MB

sslcrted_program /usr/lib/squid/security_file_certgen -s
/var/lib/ssl_db -M 4MB
sslcrted_children 8 startup=1 idle=1

acl localnet src 10.10.4.2/30
acl localnet src 172.16.0.0/12
acl localnet src 192.168.0.0/16
acl localnet src fc00::/7
acl localnet src fe80::/10

acl SSL_ports port 443
acl Safe_ports port 80
acl Safe_ports port 443
acl CONNECT method CONNECT

#acl step1 at_step SslBump1
#acl step2 at_step SslBump2
#acl step3 at_step SslBump3

#acl list_allow ssl:: "/etc/squid/lists/gov.acl"
#acl list_allow sslDOMINIOS_GOBIERNO
#ssl_bump splice list_allow
# ----- SSL-Bump pasos -----
#ssl_bump peek step1
#ssl_bump peek step2
ssl_bump bump all
#ssl_bump terminate all

#https_port 3129 intercept ssl-bump
cert=/etc/squid/ssl_cert/myCA.pem key=/etc/squid/ssl_cert/myCA.key
generate

# ----- Reglas de acceso -----

#http_access allow CONNECT DOMINIOS_GOBIERNO SSL_ports
```

```
#http_access deny CONNECT !Safe_ports
#http_access allow DOMINIOS_GOBIERNO
#http_access deny all

http_access allow CONNECT DOMINIOS_GOBIERNO
http_access allow DOMINIOS_GOBIERNO
http_access deny CONNECT !Safe_ports
http_access deny all

#ssl_outgoing_options
options=NO_SSLv3,NO_TLSv1,NO_TLSv1_1,SINGLE_DH_USE,SINGLE_ECDH_USE
```

Configuración de nft

```
table ip nat {
    chain PREROUTING {
        type nat hook prerouting priority dstnat; policy
accept;
        ip saddr 10.10.4.0/24 iifname "enx00e04c3603e7" ip
daddr != 192.168.4.0/24 tcp dport 80 redirect to :3128
        ip saddr 10.10.4.0/30 iifname "enx00e04c3603e7" tcp
dport 443 redirect to :3129
        iifname "ham0" tcp dport 80 dnat to 192.168.4.2:80
        iifname "ham0" tcp dport 443 dnat to 192.168.4.2:443
    }
    chain INPUT {
        type nat hook input priority srcnat; policy accept;
    }
    chain OUTPUT {
        type nat hook output priority dstnat; policy accept;
        ip daddr 25.42.156.33 tcp dport 80 dnat to
192.168.4.2:80
        ip daddr 25.42.156.33 tcp dport 443 dnat to
192.168.4.2:443
    }

    chain POSTROUTING {
        type nat hook postrouting priority srcnat; policy
accept;
        ip saddr 192.168.0.0/16 oifname "enp58s0" masquerade
        ip saddr 10.10.4.0/30 oifname "enp58s0" masquerade
        oifname "enx00e04c3603e7" ip daddr 192.168.4.2
```

```

masquerade
    }
}
table ip filter {
    chain FORWARD {
        type filter hook forward priority filter; policy
accept;
        iifname "ham0" oifname "enx00e04c3603e7" tcp dport {
80, 443 } accept
        iifname "enx00e04c3603e7" oifname "ham0" ct state
established,related accept
        iifname "ham0" oifname "enx00e04c3603e7" tcp flags syn
tcp option maxseg size set rt mtu
    }
}

```

Router

Interfaces de red

```

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

#auto enx00e04c3603e7
#iface enx00e04c3603e7 inet dhcp

#interface admin
auto enp1s0.10
iface enp1s0.10 inet static
    address 192.168.1.1
    netmask 255.255.255.0
    vlan-raw-device enp1s0
#-----

#interface switch clientes
#
auto enp1s0.20
iface enp1s0.20 inet static
    address 192.168.2.1

```

```

        netmask 255.255.255.0
        vlan-raw-device enp1s0
#-----

#interface proxy
#
auto lan0
iface lan0 inet static
    address    10.10.4.2
    netmask    255.255.255.252
    gateway    10.10.4.1
#    up ip route add 192.168.2.0/24 via 10.10.3.1
#-----

#interface zabbix
#
auto enp1s0.30
iface enp1s0.30 inet static
    address    192.168.3.1
    netmask    255.255.255.248
    vlan-raw-device enp1s0
#-----

#interface wordpress

auto enp1s0.40
iface enp1s0.40 inet static
    address    192.168.4.1
    netmask    255.255.255.248
    vlan-raw-device enp1s0
#-----

```

Script para aplicar configuración de vlans.conf

```

#!/bin/bash

VLAN_CONF="/etc/VLANs.conf"
ADMIN_VLAN="10" #VLAN del admin
ADMIN_IFACE="enp1s0" # Interfaz VLAN del admin

echo "=== Aplicando configuración de VLANs ==="

```

```

# Verifica que existe el archivo
if [ ! -f "$VLAN_CONF" ]; then
    echo "Error: No se encuentra $VLAN_CONF"
    exit 1
fi

declare -a VLANS_CONFIG=()

while IFS= read -r line; do
    [[ "$line" =~ ^#.*$ ]] && continue
    [[ -z "$line" ]] && continue

    VLAN_ID=$(echo "$line" | awk '{print $1}' | tr -d ' ')

    if [ -n "$VLAN_ID" ]; then
        VLANS_CONFIG+=("$VLAN_ID")
    fi
done < "$VLAN_CONF"

for vlan in $(ip -br link show | grep -E '\.[0-9]+' | awk '{print $1}'); do

    VLAN_NUM=$(echo "$vlan" | awk -F. '{print $NF}' | cut -d'@' -f1)
    IFACE_NAME=$(echo "$vlan" | awk -F. '{print $1}')
    VLAN_S=${vlan%@*}

    if [ "$VLAN_NUM" == "$ADMIN_VLAN" ] && [ "$IFACE_NAME" == "$ADMIN_IFACE" ]; then
        echo "[Protegiendo VLAN admin: $VLAN_S]"
        continue
    fi

    VLAN_S=${vlan%@*}
    if [[ ! " ${VLANS_CONFIG[@]} " =~ " ${VLAN_NUM} " ]]; then
        echo "Eliminando VLAN obsoleta: $VLAN_S"
        ip link set "$VLAN_S" down 2>/dev/null
        ip link delete "$VLAN_S" 2>/dev/null
    else
        echo "Manteniendo VLAN: $VLAN_S en configuración"
    fi
fi

```

```

done

echo ""
while IFS= read -r line; do
    [[ "$line" =~ ^#.*$ ]] && continue
    [[ -z "$line" ]] && continue

    VLAN_ID=$(echo "$line" | awk '{print $1}')
    INTERFACES=$(echo "$line" | awk '{for(i=2;i<=NF;i++) print $i}')

    for iface in $INTERFACES; do
        VLAN_NAME="${iface}.${VLAN_ID}"

        if ip link show "$VLAN_NAME" &> /dev/null; then
            ip link set "$VLAN_NAME" up 2>/dev/null
        else
            ip link add link "$iface" name "$VLAN_NAME" type vlan
id "$VLAN_ID"
            ip link set "$VLAN_NAME" up
            echo "  - $VLAN_NAME creada y activada"
        fi
    done

done < "$VLAN_CONF"

```

Script de configuracion para acceso mac e ip

```

#!/bin/bash

RULES_FILE="/etc/nftables.conf"
ACCESS_IP_FILE="/etc/router/access.ip"
ACCESS_MAC_FILE="/etc/router/access.mac"

echo "Leyendo IPs permitidas..."
ALLOWED_IPS=""
while IFS= read -r ip; do

    [[ -z "$ip" || "$ip" =~ ^# ]] && continue

```

```

    ALLOWED_IPS="$ALLOWED_IPS $ip,"
done < "$ACCESS_IP_FILE"
ALLOWED_IPS="${ALLOWED_IPS%,"}"

echo "Leyendo MACs permitidas..."
ALLOWED_MACS=""
while IFS= read -r mac; do
    [[ -z "$mac" || "$mac" =~ ^# ]] && continue
    ALLOWED_MACS="$ALLOWED_MACS $mac,"
done < "$ACCESS_MAC_FILE"
ALLOWED_MACS="${ALLOWED_MACS%,"}"

cat > "$RULES_FILE" << 'EOF'
#!/usr/sbin/nft -f
flush ruleset

define PROXY_IP = 10.10.4.1
define PROXY_HTTP_PORT = 3128
define PROXY_HTTPS_PORT = 3129
define INTERFACE_TO_PROXY = enx7cc2c646bf94

define ALLOWED_IPS = {EOF
echo "$ALLOWED_IPS }" >> "$RULES_FILE"

cat >> "$RULES_FILE" << 'EOF'

define ALLOWED_MACS = {EOF
echo "$ALLOWED_MACS }" >> "$RULES_FILE"

cat >> "$RULES_FILE" << 'EOF'

table inet nat {
    chain prerouting {
        type nat hook prerouting priority -100; policy
accept;

        iifname "enp1s0.*" ip saddr $ALLOWED_IPS tcp dport
80 dnat ip to $PROXY_IP:$PROXY_HTTP_PORT
        iifname "enp1s0.*" ip saddr $ALLOWED_IPS tcp dport
443 dnat ip to $PROXY_IP:$PROXY_HTTPS_PORT

```

```

        iifname "enp1s0.*" ether saddr $ALLOWED_MACS tcp
dport 80 dnat ip to $PROXY_IP:$PROXY_HTTP_PORT
        iifname "enp1s0.*" ether saddr $ALLOWED_MACS tcp
dport 443 dnat ip to $PROXY_IP:$PROXY_HTTPS_PORT
    }

    chain postrouting {
        type nat hook postrouting priority 100; policy
accept;
        oifname $INTERFACE_TO_PROXY ip saddr $ALLOWED_IPS
counter masquerade
        oifname $INTERFACE_TO_PROXY ether saddr
$ALLOWED_MACS counter masquerade
    }
}

table inet filter {
    chain input {
        type filter hook input priority 0; policy drop;

        iif lo accept
        ct state established,related accept

        iifname "enp1s0.*" ip saddr $ALLOWED_IPS udp dport
53 accept
        iifname "enp1s0.*" ip saddr $ALLOWED_IPS tcp dport
53 accept

        iifname "enp1s0.*" ether saddr $ALLOWED_MACS udp
dport 53 accept
        iifname "enp1s0.*" ether saddr $ALLOWED_MACS tcp
dport 53 accept

        iifname "enp1s0.*" ip saddr $ALLOWED_IPS ip
protocol icmp accept

        iifname "enp1s0.*" ether saddr $ALLOWED_MACS ip
protocol icmp accept

```



```

        iifname "enp1s0.*" ip saddr $ALLOWED_IPS tcp dport
22 accept
        iifname "enp1s0.*" ether saddr $ALLOWED_MACS tcp
dport 22 accept

        iifname "enp1s0.*" ip saddr $ALLOWED_IPS accept

        iifname "enp1s0.*" ether saddr $ALLOWED_MACS
accept

        iifname $INTERFACE_TO_PROXY ct state
established,related accept
    }

    chain forward {
        type filter hook forward priority 0; policy drop;

        ct state established,related accept

        iifname "enp1s0.*" ip saddr $ALLOWED_IPS oifname
$INTERFACE_TO_PROXY accept

        iifname "enp1s0.*" ether saddr $ALLOWED_MACS
oifname $INTERFACE_TO_PROXY accept

        iifname $INTERFACE_TO_PROXY oifname "enp1s0.*" ip
daddr $ALLOWED_IPS ct state established,related accept
        iifname $INTERFACE_TO_PROXY oifname "enp1s0.*" ct
state established,related accept

        iifname "enp1s0.*" oifname "enp1s0.*" ip saddr
$ALLOWED_IPS ip daddr $ALLOWED_IPS accept

        iifname "enp1s0.*" oifname "enp1s0.*" ether saddr
$ALLOWED_MACS accept
    }

    chain output {
        type filter hook output priority 0; policy accept;
    }
}
EOF

```

```
echo "Aplicando reglas nftables"
nft -f "$RULES_FILE"

echo "IPs permitidas: $ALLOWED_IPS"
echo "MACs permitidas: $ALLOWED_MACS"

nft list ruleset
```

Virtualizador

Admin

Script de bridge10 para vlan administrador:

```
#!/bin/bash
modprobe 8021q

ip link add link enp12s0 name enp12s0.10 type vlan id 10

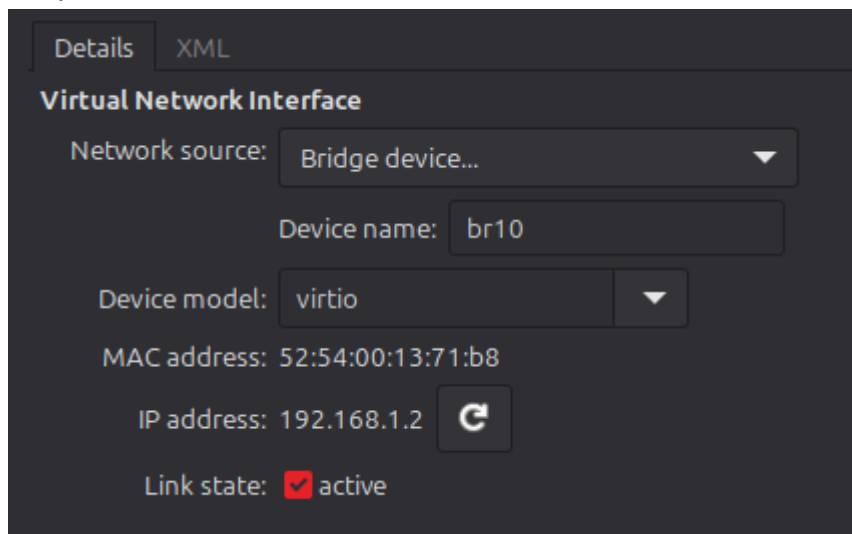
ip link add name br10 type bridge

ip link set dev enp12s0.10 master br10

ip link set dev enp12s0.10 up

ip link set dev br10 up
```

Grupo en la vlan



Details XML


Virtual Network Interface

Network source: Bridge device... ▼

Device name: br10

Device model: virtio ▼

MAC address: 52:54:00:13:71:b8

IP address: 192.168.1.2 

Link state: ☒ active

Interface para el admin

#Admin

#Interface

auto enp1s0

iface enp1s0 inet static

address 192.168.1.2/24

gateway 192.168.1.1

up ip route add 192.168.2.0/24 via 192.168.1.1

up ip route add 192.168.3.0/29 via 192.168.1.1

```
up ip route add 192.168.4.0/29 via 192.168.1.1
```

Clientes

Script de br20 para vlan clientes:

```
#!/bin/bash
modprobe 8021q

ip link add link enp12s0 name enp12s0.20 type vlan id 20

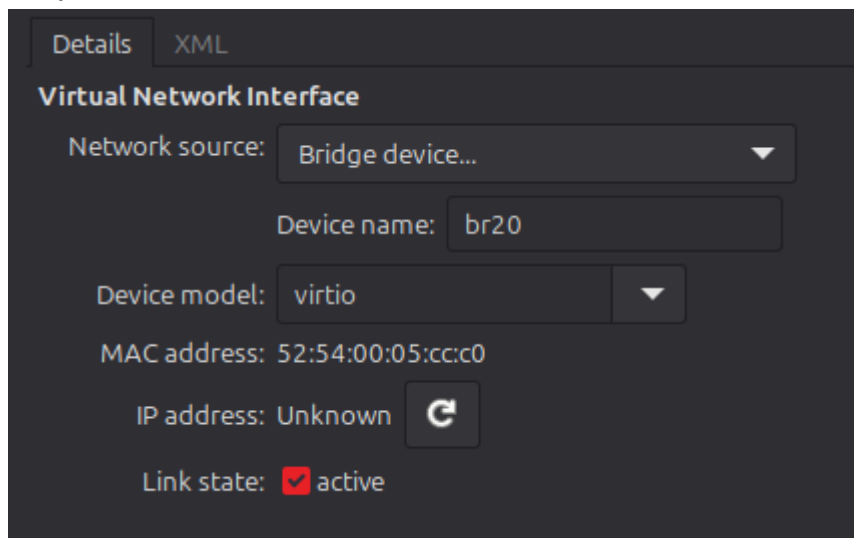
ip link add name br20 type bridge

ip link set dev enp12s0.20 master br20

ip link set dev enp12s0.20 up

ip link set dev br20 up
```

Grupo en vlan



Details XML

Virtual Network Interface

Network source: Bridge device... ▼

Device name: br20

Device model: virtio ▼

MAC address: 52:54:00:05:cc:c0

IP address: Unknown ↻

Link state: ☒ active

Interface para cada cliente:

```
#Clientes
```

```
#Interface
```

```
auto enp1s0
```

```
iface enp1s0 inet static
```

```
    address 192.168.2.2/24
```

```
    gateway 192.168.2.1
```

```
    up ip route add 192.168.1.0/24 via 192.168.2.1
```

```
    up ip route add 192.168.3.0/29 via 192.168.2.1
```

```
    up ip route add 192.168.4.0/29 via 192.168.2.1
```

```
auto enp1s0
```

```
iface enp1s0 inet static
    address 192.168.2.3/24
    gateway 192.168.2.1
    up ip route add 192.168.1.0/24 via 192.168.2.1
    up ip route add 192.168.3.0/29 via 192.168.2.1
    up ip route add 192.168.4.0/29 via 192.168.2.1
```

```
auto enp1s0
```

```
iface enp1s0 inet static
    address 192.168.2.4/24
    gateway 192.168.2.1
    up ip route add 192.168.1.0/24 via 192.168.2.1
    up ip route add 192.168.3.0/29 via 192.168.2.1
    up ip route add 192.168.4.0/29 via 192.168.2.1
```

Zabbix

Script de br30 para vlan zabbix:

```
#!/bin/bash
modprobe 8021q

ip link set dev br30 up

ip link add link enp12s0 name enp12s0.30 type vlan id 30

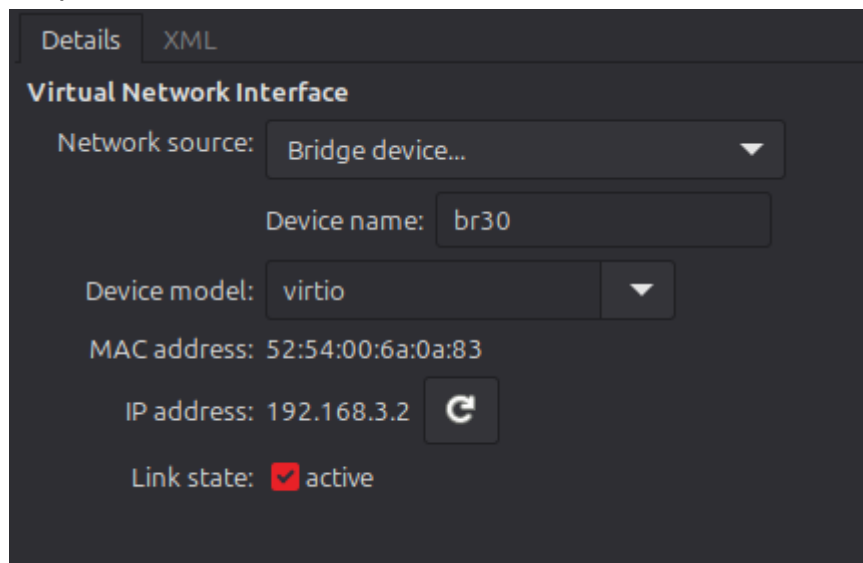
ip link add name br30 type bridge

ip link set dev enp12s0.30 master br30

ip link set dev enp12s0.30 up

ip link set dev br30 up
```

Grupo en vlan



The screenshot shows a web interface for configuring a 'Virtual Network Interface'. At the top, there are two tabs: 'Details' (selected) and 'XML'. Below the tabs, the title 'Virtual Network Interface' is displayed. The configuration fields are as follows: 'Network source' is a dropdown menu set to 'Bridge device...'; 'Device name' is a text input field containing 'br30'; 'Device model' is a dropdown menu set to 'virtio'; 'MAC address' is a text input field containing '52:54:00:6a:0a:83'; 'IP address' is a text input field containing '192.168.3.2' with a refresh icon to its right; and 'Link state' is a checkbox labeled 'active' which is checked.

Interface para zabbix:

```
#Zabbix
```

```
#Interface
```

```
auto enp1s0
```

```
iface enp1s0 inet static
```

```
    address 192.168.3.2/29
```

```
    gateway 192.168.3.1
```

```
    up ip route add 192.168.1.0/24 via 192.168.3.1
```

```
    up ip route add 192.168.2.0/24 via 192.168.3.1
```

```
    up ip route add 192.168.4.0/29 via 192.168.3.1
```


Configuración Zabbix Host:

Host

Host

IPMI

Tags

Macros

Inventory

Encryption

Value mapping

* Host name

Load Balancer

Visible name

Load Balancer

Templates

type here to search

Select

* Host groups

Zabbix servers

type here to search

Select

Interfaces

Type

IP address

DNS name

Connect to

Port

Default

Agent

10.10.3.1

IP

DNS

10050

Remove

Add

Description

Monitored by

Server

Proxy

Proxy group

Enabled

Update

Clone

Delete

Cancel

Configuración de métricas (items) para graficas:

<input type="checkbox"/>	Name ▲	Triggers	Key	Interval	History	Trends	Type	Status	Tags	Info
<input type="checkbox"/>	*** Active Interface - Fail Over		custom.active.interface		31d	365d	Zabbix trapper	Enabled		
<input type="checkbox"/>	*** Test Value		test.value		31d	365d	Zabbix trapper	Enabled		
<input type="checkbox"/>	*** Trafico Entrante isp1		isp1.rx.bytes		31d	0	Zabbix trapper	Enabled		
<input type="checkbox"/>	*** Trafico Entrante isp2		isp2.rx.bytes		31d	0	Zabbix trapper	Enabled		
<input type="checkbox"/>	*** Trafico Saliente isp1		isp1.tx.bytes		31d	0	Zabbix trapper	Enabled		
<input type="checkbox"/>	*** Trafico Saliente isp2		isp2.tx.bytes		31d	0	Zabbix trapper	Enabled		

Displaying 6 of 6 found

Configuración para la gráfica:

* Name

* Width

* Height

Graph type

Show legend ☒

Show working time ☒

Show triggers ☒





Percentile line (left) ☐

Percentile line (right) ☐

Y axis MIN value

Y axis MAX value

* Items

	Name	Function	Draw style	Y axis side	Color	Action
1:	Load Balancer: Trafico Entrante isp1	all	Line	Left		Remove
2:	Load Balancer: Trafico Entrante isp2	all	Line	Left		Remove
3:	Load Balancer: Trafico Saliente isp1	all	Line	Left		Remove
4:	Load Balancer: Trafico Saliente isp2	all	Line	Left		Remove

[Add](#)

[Update](#) [Clone](#) [Delete](#) [Cancel](#)

Wordpress

Script de br40 para vlan Wordpress:

```
#!/bin/bash
modprobe 8021q

ip link add link enp12s0 name enp12s0.40 type vlan id 40

ip link add name br40 type bridge

ip link set dev enp12s0.40 master br40

ip link set dev enp12s0.40 up

ip link set dev br40 up
```

Grupo en vlan

Details

XML

Virtual Network Interface

Network source:

Bridge device...

Device name:

br40

Device model:

virtio

MAC address:

52:54:00:bc:dc:c2

IP address:

192.168.4.2

Link state:

☒ active

Interface para wordpress:

```
#Wordpress
```

```
#Interface
```

```
auto enp1s0
```

```
iface enp1s0 inet static
```

```
    address 192.168.4.2/29
```

```
    gateway 192.168.4.1
```

```
    up ip route add 192.168.1.0/24 via 192.168.4.1
```

```
    up ip route add 192.168.2.0/24 via 192.168.4.1
```

```
    up ip route add 192.168.3.0/29 via 192.168.4.1
```