

Práctica 2: Redes



Álvaro Santos Romero

Índice

1. Preparación del entorno (parte 1)	2
2. Configuraciones iptables	3
3. Preparación del entorno (parte 2)	6
4. Servidor DHCP	10
4.1. Instalación	10
4.2. Configuración	10
4.2.1. configuración de las subredes	10
4.2.2. Fichero leases	12
5. Servidor DNS	13
5.1. Vagrantfile	13
5.2. Configuración	14
5.2.1. Zona directa	14
5.2.2. Zona inversa	15
5.3. Funcionamiento	15
5.3.1. Resolución directa	16
5.3.2. Resolución inversa	16

1. Preparación del entorno (parte 1)

1. Crear tres máquinas en la misma red y prepara un fichero de configuración con **iptables** y **nmap**.

Para la creación de las máquinas virtuales, vamos a configurar un fichero **Vagrantfile** para lanzarlas simultáneamente.

Mediante la definición de red, asignamos a cada máquina una IP diferente.

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"

  config.vm.define "vm1" do |vm1|
    vm1.vm.hostname = "vm1"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm1.vm.network "private_network", ip: "192.168.100.10"
  end

  config.vm.define "vm2" do |vm2|
    vm2.vm.hostname = "vm2"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm2.vm.network "private_network", ip: "192.168.100.20"
  end

  config.vm.define "vm3" do |vm3|
    vm3.vm.hostname = "vm3"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm3.vm.network "private_network", ip: "192.168.100.30"
  end
end
```

Figura 1: Ejercicio 1

Con este script de aprovisionamiento vamos a instalar en las máquinas **nmap** e **iptables**.

```
#!/usr/bin/env bash

apt-get install -y nmap
apt-get install -y iptables
```

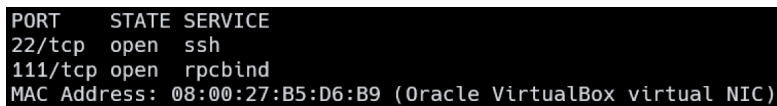
Figura 2: Ejercicio 1

Una vez lanzadas las máquinas, comprobamos que todas están configuradas correctamente.

2. Configuraciones iptables

1. Comprobar que puertos tiene abiertos VM1 desde VM2.

```
nmap -open -p- -v vm1
```



```
PORT      STATE SERVICE
22/tcp    open  ssh
111/tcp   open  rpcbind
MAC Address: 08:00:27:B5:D6:B9 (Oracle VirtualBox virtual NIC)
```

Figura 3: Ejercicio 2

Vemos que la máquina tiene abiertos los puertos **22** y **111**, es decir, están siendo usados por algún proceso.

- **-p-** — Escanea todo el abanico de puertos (1 - 65535).
- **-open** — Muestra sólo los puertos abiertos.
- **-v** — Muestra más información en el escaneo.

2. Prohibir acceso por SSH.

```
sudo iptables -A INPUT -p tcp --dport 22 -j DROP
```

- **iptables -A** — Añadir a la tabla seleccionada.
- **INPUT** — Para paquetes de entrada en nuestro sistema.
- **-p** — Protocolo de la regla, en este caso, tcp.
- **--dport 22** — Puerto de entrada afectado por la regla.
- **-j DROP** — Indicamos a la tabla qué hacer en cuanto encuentre un objetivo.

3. Responder a las siguientes preguntas.

- a) ¿Por qué se pierde el acceso por SSH al ejecutar las reglas iptables?

El acceso a la máquina vía SSH se ha perdido, ya que le hemos indicado al cortafuegos que bloquee toda entrada de conexiones **tcp** por el puerto **22**.

- b) ¿Qué pasa si recargamos la máquina?

Si recargamos la máquina, podemos volver a acceder a ella por **ssh** puesto que las tablas no guardan el estado (habría que usar iptables-persistent).

- c) ¿Si reiniciamos la máquina se pierden las reglas?

Si, puesto que la configuración de iptables se recarga por defecto.

4. Permitir conexiones locales.

```
sudo iptables -A INPUT -p ALL -s localhost -j ACCEPT
```

- **-s fuente** — Indicamos la fuente de las conexiones.

5. Permitir conexiones ya establecidas.

```
sudo iptables -A INPUT -m conntrack --ctstate ESTABLISHED -p ALL -j ACCEPT
```

- **-m conntrack --ctstate ESTABLISHED** — Para indicar que vamos a utilizar el comando ctstate para filtrar por conexiones ya establecidas.

6. Configurar políticas por defecto de rechazo de paquetes.

```
sudo iptables -P INPUT DROP
```

- **-P** — Añade una política por defecto a una cadena.
- **INPUT** — Para paquetes de entrada a nuestro sistema.
- **-DROP** — Le indicamos a iptables que la política por defecto va a ser rechazar los paquetes.

Para guardar el entorno configurado, vamos a utilizar el comando **iptables-save > archivo**, que permite exportar las reglas de un archivo, en nuestro caso, a nuestra carpeta compartida de vagrant. Para restaurarlo, **iptables-restore < archivo**.

7. Permitir conexión a los puertos 80 y 443.

```
sudo iptables -A INPUT -p tcp -dport 80 -j ACCEPT
```

```
sudo iptables -A INPUT -p tcp -dport 443 -j ACCEPT
```

- **-p tcp** — Protocolo de la regla.
- **-dport** — Puerto destino.
- **-j ACCEPT** — Le indicamos a iptables que queremos aceptar las conexiones.

8. Conexión únicamente de VM2 mediante FTP.

```
sudo iptables -A INPUT -p tcp -dport 21 -s vm2 -j ACCEPT
```

- **-p tcp** — Protocolo de la regla.
- **-dport** — Puerto destino de la regla.
- **-s vm2** — Origen de la petición
- **-j ACCEPT** — Le indicamos a iptables que sólo queremos que se conecte **vm2**.

9. Conexión desde VM1 a MySQL únicamente desde localhost.

```
sudo iptables -A INPUT -p tcp -dport 3306 -s localhost -j ACCEPT
```

- **-p tcp** — Protocolo de MySQL es tcp.
- **-dport** — Puerto destino, en este caso, 3306 (puerto por defecto de MySQL).
- **-s localhost** — Origen de la conexión.
- **-j ACCEPT** — Aceptar las conexiones.

Para comprobar el funcionamiento, vamos a utilizar **telnet** y **netcat**.

Para este ejemplo, desde la máquina **VM1** nos ponemos en escucha por el puerto 3306.

```
vagrant@vm2:~$ telnet vm1 3306
Trying 192.168.100.10...
telnet: Unable to connect to remote host: Connection timed out
```

Figura 4: Prueba de las reglas

Como se observa, no podemos conectarnos desde otra máquina, pero desde la propia máquina sí.

```
root@vm1:/vagrant# telnet localhost 3306
Trying ::1...
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
hola
que tal
elpepe
```

Figura 5: Prueba de las reglas

10. Rechazar las conexiones de VM2 por ftp y aceptar las conexiones de los puertos 1:1000.

```
sudo iptables -A INPUT -p tcp -s vm2 -dport 21 -j DROP
```

```
sudo iptables -A INPUT -p tcp -s vm2 -dport 1:1000 -j ACCEPT
```

- **-p tcp** — Protocolo de la regla tcp ya que sólo podemos utilizar rango de puertos con tcp.
- **-dport 1:1000** — Puertos destino de la regla.
- **-s vm2** — Le indicamos a **iptables** que sólo queremos que se conecte desde **VM2**.
- **-j ACCEPT** — Aceptamos las conexiones.

Para **VM3** hacemos lo mismo.

Finalmente, así es como quedaría la tabla filter de nuestro "servidor http/s".

```
Chain INPUT (policy DROP)
target     prot opt source                destination            ctstate ESTABLISHED
ACCEPT     all  --  localhost              anywhere
ACCEPT     all  --  anywhere               anywhere
ACCEPT     tcp  --  anywhere               anywhere               tcp dpt:http
ACCEPT     tcp  --  anywhere               anywhere               tcp dpt:https
ACCEPT     tcp  --  vm2                    anywhere               tcp dpt:ftp
ACCEPT     tcp  --  localhost              anywhere               tcp dpt:mysql
DROP       tcp  --  vm2                    anywhere               tcp dpt:ftp
ACCEPT     tcp  --  vm2                    anywhere               tcp dpts:tcpmux:1000
ACCEPT     tcp  --  vm3                    anywhere               tcp dpts:tcpmux:1000
```

Figura 6: tabla filter iptables

3. Preparación del entorno (parte 2)

. Creación de un entorno de dos subredes y un router.

Router IPs — 192.168.2.1/192.168.3.1 (necesitamos tener acceso desde el router a ambas subredes).

```
vagrant@router:~$ netstat -rn
```

Kernel IP routing table							
Destination	Gateway	Genmask	Flags	MSS	Window	irtt	Iface
0.0.0.0	10.0.2.2	0.0.0.0	UG	0 0		0	eth0
10.0.2.0	0.0.0.0	255.255.255.0	U	0 0		0	eth0
10.0.2.2	0.0.0.0	255.255.255.255	UH	0 0		0	eth0
192.168.2.0	0.0.0.0	255.255.255.0	U	0 0		0	eth1
192.168.3.0	0.0.0.0	255.255.255.0	U	0 0		0	eth2

Figura 7: Tabla de rutas del router

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"

  config.vm.define "router" do |router|
    router.vm.hostname = "router"
    config.vm.provision :shell, path: "bootstrap.sh"
    router.vm.network "private_network", ip: "192.168.2.1"
    router.vm.network "private_network", ip: "192.168.3.1"
  end

  config.vm.define "nodo1" do |nodo1|
    nodo1.vm.hostname = "nodo1"
    config.vm.provision :shell, path: "bootstrap.sh"
    nodo1.vm.network "private_network", ip: "192.168.2.2"
  end

  config.vm.define "nodo2" do |nodo2|
    nodo2.vm.hostname = "nodo2"
    config.vm.provision :shell, path: "bootstrap.sh"
    nodo2.vm.network "private_network", ip: "192.168.2.3"
  end

  config.vm.define "nodo3" do |nodo3|
    nodo3.vm.hostname = "nodo3"
    config.vm.provision :shell, path: "bootstrap.sh"
    nodo3.vm.network "private_network", ip: "192.168.3.2"
  end

  config.vm.define "nodo4" do |nodo4|
    nodo4.vm.hostname = "nodo4"
    config.vm.provision :shell, path: "bootstrap.sh"
    nodo4.vm.network "private_network", ip: "192.168.3.3"
  end
end
```

Figura 8: Vagrantfile del apartado

Se ha considerado asignar a los equipos una ip diferente a .1 para evitar conflictos de dispositivo.

Fichero de configuración del Vagrantfile.

```
#!/bin/bash

apt-get install iptables -y
apt-get install traceroute -y
```

Figura 9: Bootstrap

Ahora vamos a realizar la configuración del cortafuegos.

- Habilitamos el acceso a las diferentes máquinas mediante ssh cambiando la configuración del fichero `sshd_config`.

```
PasswordAuthentication yes
```

Al activar esta opción, permitimos el acceso por ssh utilizando la contraseña de la máquina (es conveniente reiniciar el servicio sshd para que se apliquen los cambios).

- A continuación, desde el router nos conectamos a cada nodo mediante ssh y tiramos las interfaces por defecto (en este caso, `eth0`).

Lo haremos desde el router porque si tiramos la interfaz desde la propia máquina nos quedaremos sin acceso ssh, ya que nosotros estamos conectados a esa interfaz desde Vagrant.

```
vagrant@nodo1:~$ sudo ifconfig eth0
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 10.0.2.15 netmask 255.255.255.0 broadcast 10.0.2.255
    inet6 fe80::a00:27ff:febb:1475 prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:bb:14:75 txqueuelen 1000 (Ethernet)
    RX packets 949 bytes 115056 (115.0 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 693 bytes 116137 (116.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 10: eth0

Cuando tiremos la ruta por defecto, si hacemos un traceroute, veremos que salta un error de red inalcanzable, puesto que ahora nuestra máquina no tiene otra ruta hacia el internet.

```
vagrant@nodo4:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
connect: Network is unreachable
```

Figura 11: Red inalcanzable

- Una vez hemos tirado las interfaces por defecto de los diferentes equipos, tendremos que añadir la ruta por defecto hacia el router.

```
vagrant@nodo1:~$ sudo ip route add default via 192.168.2.1 dev eth1

vagrant@nodo1:~$ ip route show
default via 192.168.2.1 dev eth1
192.168.2.0/24 dev eth1 proto kernel scope link src 192.168.2.2
```

Figura 12: Nueva ruta por defecto

¡Importante! — Configurarlos para todas las máquinas de la subred.

- Posteriormente, activaremos el **ip-forwarding** en el router para retransmitir los paquetes entre redes. Para ello, editaremos el fichero `/etc/sysctl.conf` y recargamos las opciones seleccionadas con **sysctl -p**.

```
net.ipv4.ip_forward=1
```

- Por último, añadiremos una regla a la cadena **IPFORWARDING** de la tabla **nat** para enmascarar la ip origen de los remitentes de un paquete por la de la interfaz pública de una red, ocultando su ip privada.

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

```
Chain POSTROUTING (policy ACCEPT)
target     prot opt source                destination
MASQUERADE all  --  0.0.0.0/0              0.0.0.0/0
```

Figura 13: postrouting

- **-t nat** — Indicamos la tabla destino.
- **-A POSTROUTING** — Indicamos la cadena destino de la tabla seleccionada.
- **-o eth0** — Indicamos la interfaz de salida de la regla, en este caso, la interfaz por defecto del router.
- **-j MASQUERADE** — Enmascaramos (sustituimos) la ip del equipo origen de los paquetes. De esta forma, el destinatario del paquete no conocerá la ip del equipo, sino la enmascarada por el router.

El router mantendrá una tabla de IPs enmascaradas para posteriormente enviar los paquetes de vuelta al equipo original.

Comprobaciones traceroute desde el router a google (8.8.8.8). Vemos que sale directamente por la

```
vagrant@router:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (10.0.2.2)  1.254 ms  0.550 ms  1.357 ms
 2  * * *
 3  * * *
 4  * * *
 5  * * *
```

Figura 14: google desde el router

interfaz pública a internet.

Traceroute desde Nodo1 (subnet 192.168.2.0/24) a google.

```
vagrant@nodo1:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (192.168.2.1)  1.400 ms  0.888 ms  0.837 ms
 2  10.0.2.2 (10.0.2.2)  1.569 ms  3.058 ms  3.696 ms
 3  * * *
 4  * * *
 5  * * *
```

Figura 15: google desde el nodo1

Traceroute desde Nodo3 (subnet 192.168.2.0/24) a google.

```
vagrant@nodo3:~$ traceroute 8.8.8.8
traceroute to 8.8.8.8 (8.8.8.8), 30 hops max, 60 byte packets
 1  _gateway (192.168.3.1)  1.330 ms  0.715 ms  1.096 ms
 2  10.0.2.2 (10.0.2.2)  1.584 ms  2.532 ms  3.937 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
```

Figura 16: google desde el nodo3

En cambio, desde los nodos vemos que primero pasa por la **puerta de enlace** configurada y luego por la **ip pública** del router.

4. Servidor DHCP

4.1. Instalación

En la práctica utilizaremos el servidor **isc-dhcp**.

Para instalarlo en **Ubuntu 20.04**:

```
apt-get install isc-dhcp-server -y
```

4.2. Configuración

El fichero de configuración de dhcp se encuentra en **/etc/dhcp/dhcpd.conf**.

4.2.1. configuración de las subredes

```
#DHCP Ejercicio7

subnet 192.168.2.0 netmask 255.255.255.0 {
    #Tiempo de préstamo predeterminado
    default-lease-time 600;
    #Tiempo de préstamo máximo
    max-lease-time 3600;
    #Rango de ips a prestar dentro de la subred
    range 192.168.2.100 192.168.2.200;
}

subnet 192.168.3.0 netmask 255.255.255.0 {
    #Tiempo de préstamo predeterminado
    default-lease-time 600;
    #Tiempo de préstamo máximo
    max-lease-time 3600;
    #Rango de ips a prestar dentro de la subred
    range 192.168.3.100 192.168.3.200;
}
```

Figura 17: configuración dhcp

Una vez configurado, reiniciamos el servicio (**sudo systemctl restart isc-dhcpd-server**) para cargar la nueva configuración establecida.

A continuación, comprobamos que tenemos el fichero de préstamos (**dhcpd.leases**) en **/var/lib/dhcp/**, si no lo tenemos, lo creamos con nuestro editor de texto favorito.

Este fichero mantendrá una tabla con los préstamos realizados a los equipos.

Para finalizar, ejecutaremos el comando **sudo dhcpcd -f** para ejecutar el servicio en primer plano. Realmente lo usaremos para determinar si todo se ha configurado correctamente.

```
vagrant@nodo1:~$ sudo dhclient -v eth1
Internet Systems Consortium DHCP Client 4.3.5
Copyright 2004-2016 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth1/08:00:27:6a:5f:ed
Sending on   LPF/eth1/08:00:27:6a:5f:ed
Sending on   Socket/fallback
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x68057ae0)
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x68057ae0)
DHCPREQUEST of 192.168.2.5 on eth1 to 255.255.255.255 port 67 (xid=0x68057ae0)
DHCPDISCOVER on eth1 to 255.255.255.255 port 67 interval 3 (xid=0xb2bf0b2d)
DHCPREQUEST of 192.168.2.100 on eth1 to 255.255.255.255 port 67 (xid=0xd0bbfb2)
DHCPOFFER of 192.168.2.100 from 192.168.2.2
DHCPACK of 192.168.2.100 from 192.168.2.2
bound to 192.168.2.100 -- renewal in 245 seconds.

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.2.100 netmask 255.255.255.0 broadcast 192.168.2.255
    inet6 fe80::a00:27ff:fe6a:5fed prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:6a:5f:ed txqueuelen 1000 (Ethernet)
    RX packets 247 bytes 43238 (43.2 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 436 bytes 128518 (128.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 18: Petición de ip

Parece que se ha configurado correctamente, puesto que está escuchando en ambas subredes.

Además, vemos que nos ha asignado la IP **192.168.2.100** la cuál está dentro del rango asignado en el fichero de configuración.

Ahora, vamos a configurar el equipo **host4** de la **subred 2** para que el servidor **dhcp** le asigne una ip estática.

Para ello, volvemos al fichero de configuración.

```
subnet 192.168.3.0 netmask 255.255.255.0 {
    #Tiempo de préstamo predeterminado
    default-lease-time 600;
    #Tiempo de préstamo máximo
    max-lease-time 3600;
    #Rango de ips a prestar dentro de la subred
    range 192.168.3.100 192.168.3.200;

    group{
        #Tiempo de préstamo predeterminado
        default-lease-time 600;
        #Tiempo de préstamo máximo
        max-lease-time 3600;

        host nodo4{
            #Dirección física tarjeta de red por la que mandaremos la petición desde nodo4 (eth1)
            hardware ethernet 08:00:27:32:1c:8b;
            #ip fija del equipo
            fixed-address 192.168.3.69;
            option host-name "nodo4";
        }
    }
}
```

Figura 19: ip estática

Debemos de crear en el apartado de la subnet 2, un grupo para albergar los diferentes hosts.

En nuestro caso, indicamos que al **nodo 4** desde la tarjeta de red (**eth1:08:00:27:32:1c:b**) nos asigne la IP (**192.168.3.69**).

¡importante! — La IP estática no debe de estar dentro del rango de IPs que proporciona el dhcp automáticamente.

```
eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.3.69 netmask 255.255.255.0 broadcast 192.168.3.255
    inet6 fe80::a00:27ff:fe32:1c8b prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:32:1c:8b txqueuelen 1000 (Ethernet)
    RX packets 225 bytes 29190 (29.1 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 212 bytes 53904 (53.9 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figura 20: asignación ip estática

Vemos que le asigna la IP correctamente.

4.2.2. Fichero leases

```
lease 192.168.2.103 {
    starts 1 2022/04/11 22:35:55;
    ends 1 2022/04/11 22:45:55;
    cltt 1 2022/04/11 22:35:55;
    binding state active;
    next binding state free;
    rewind binding state free;
    hardware ethernet 08:00:27:6a:5f:ed;
    client-hostname "nodo1";
}
```

Figura 21: fichero leases

¡importante! — La asignación de IPs estáticas no aparecen en este fichero, ya que se configuran directamente en la máquina cliente.

5. Servidor DNS

5.1. Vagrantfile

Vagrantfile para el funcionamiento de la práctica.

```
Vagrant.configure("2") do |config|
  config.vm.box = "hashicorp/bionic64"

  config.vm.define "router" do |router|
    router.vm.hostname = "router"
    config.vm.provision :shell, path: "bootstrapRouter.sh"
    router.vm.network "private_network", ip: "192.168.2.1"
    router.vm.network "private_network", ip: "192.168.3.1"
  end

  config.vm.define "vm1" do |vm1|
    vm1.vm.hostname = "vm1"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm1.vm.network "private_network", ip: "192.168.2.2"
  end

  config.vm.define "vm2" do |vm2|
    vm2.vm.hostname = "vm2"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm2.vm.network "private_network", ip: "192.168.2.3"
  end

  config.vm.define "vm3" do |vm3|
    vm3.vm.hostname = "vm3"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm3.vm.network "private_network", ip: "192.168.3.2"
  end

  config.vm.define "vm4" do |vm4|
    vm4.vm.hostname = "vm4"
    config.vm.provision :shell, path: "bootstrap.sh"
    vm4.vm.network "private_network", ip: "192.168.3.3"
  end
end
```

Figura 22: vagrantfile

Fichero de aprovisionamiento:

```
#!/bin/bash
apt-get update
apt-get upgrade -y
apt-get install bind9 -y
```

Figura 23: bootstrap.sh

5.2. Configuración

Antes de nada, editaremos el fichero **named.conf.options**.

```
acl "trusted" {
    192.168.2.1;    #dns red 1
    192.168.2.2;    #host 1 red 1
    192.168.2.3;    #host 2 red 1
    192.168.3.1;    #dns red 2
    192.168.3.2;    #host 1 red 2
    192.168.3.3;    #host 2 red 2
};

options {
    directory "/var/cache/bind";

    recursion yes;
    allow-recursion {trusted; };
    listen-on { 192.168.2.1; 192.168.3.1; };
    allow-transfer {none; };

    forwarders {
        8.8.8.8;
        8.8.4.4;
    };
};
```

Figura 24: named.conf.options

En este fichero estamos indicando que las IPs de los hosts y de los servidores están en una categoría llamada **trusted**, la cuál nos va a permitir que los servidores puedan enviarse las peticiones con la opción **recursion** en caso de no encontrar ese nombre en su servidor (sección **options** del fichero).

También indicamos que sólo escuchen desde las respectivas IPs **.1**, que en este caso, son los servidores **DNS** de cada **red**.

A continuación, copiamos el fichero **named.defaults-zones** en **named.conf.local** (nos servirá de referencia).

5.2.1. Zona directa

```
zone "net1.as.uca.es" {
    type master;
    file "/etc/bind/db.net1.as.uca.es";
};
```

Figura 25: zona directa net 1

```
zone "net2.as.uca.es" {
    type master;
    file "/etc/bind/db.net2.as.uca.es";
};
```

Figura 26: zona directa net 2

```
$TTL 604800
@ IN SOA net1.as.uca.es. admin.net1.as.uca.es. (
    11 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative
; Registro de nombre del servidor (red 1)
@ IN NS ns1.net1.as.uca.es.
; Registro dirección servidor (red 1)
net1.as.uca.es. IN NS ns1.net1.as.uca.es.
; Registro de los hosts (red 1)
ns1 IN A 192.168.2.1
vm1 IN A 192.168.2.2
vm2 IN A 192.168.2.3
```

Figura 27: Base de datos de la zona directa net 1

5.2.2. Zona inversa

```
zone "2.168.192.in-addr.arpa" {
    type master;
    file "/etc/bind/db.192.186.2";
};
```

Figura 28: zona inversa

```
$TTL 604800
@ IN SOA ns1.net1.as.uca.es. admin.net1.as.uca.es. (
    9 ; Serial
    604800 ; Refresh
    86400 ; Retry
    2419200 ; Expire
    604800 ) ; Negative
; Nombre de dominio
@ IN NS ns1.
;
1 IN PTR ns1.net1.as.uca.es. ;192.168.2.1; ns1
2 IN PTR vm1.net1.as.uca.es. ;192.168.2.2; vm1
3 IN PTR vm2.net1.as.uca.es. ;192.168.2.3; vm2
```

Figura 29: Base de datos de la zona inversa

5.3. Funcionamiento

Para comprobar su funcionamiento, debemos de configurar las redes de igual manera que hicimos en la **creación del entorno (parte 2)**.

Una vez realizada la configuración, podremos empezar a configurar el fichero **/etc/resolv.conf** para añadir los DNS del router.

```
nameserver 127.0.0.53
nameserver 192.168.2.1
nameserver 192.168.3.1
options edns0
```

Figura 30: Resolv.conf

De esta manera, al realizar peticiones DNS, utilizará los servidores de nombre del router en caso de no encontrar la información en el de por defecto.

Desde los hosts de la red 1.

¡Nota! — Sólo se adjuntan capturas del primer host, puesto que en todos se realiza lo mismo y sería información redundante.

5.3.1. Resolución directa

```
vagrant@vm1:~$ nslookup vm1.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:      192.168.2.1
Address:     192.168.2.1#53

Name:   vm1.net1.as.uca.es
Address: 192.168.2.2

vagrant@vm1:~$ nslookup vm2.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:      192.168.2.1
Address:     192.168.2.1#53

Name:   vm2.net1.as.uca.es
Address: 192.168.2.3
```

Figura 31: resolución directa

5.3.2. Resolución inversa

¡Nota! — Hay que indicar que utilice el servidor para que muestre correctamente el nombre.

```
vagrant@vm1:~$ host 192.168.2.1 192.168.2.1
Using domain server:
Name: 192.168.2.1
Address: 192.168.2.1#53
Aliases:

1.2.168.192.in-addr.arpa domain name pointer ns1.net1.as.uca.es.

vagrant@vm1:~$ host 192.168.2.2 192.168.2.1
Using domain server:
Name: 192.168.2.1
Address: 192.168.2.1#53
Aliases:

2.2.168.192.in-addr.arpa domain name pointer vm1.net1.as.uca.es.

vagrant@vm1:~$ host 192.168.2.3 192.168.2.1
Using domain server:
Name: 192.168.2.1
Address: 192.168.2.1#53
Aliases:

3.2.168.192.in-addr.arpa domain name pointer vm2.net1.as.uca.es.
```

Figura 32: resolución inversa

Usando el comando dig

```

vagrant@vm3:~$ dig vm4.net2.as.uca.es @192.168.3.1
; <<> DiG 9.11.3-1ubuntu1.17-Ubuntu <<> vm4.net2.as.uca.es @192.168.3.1
; global options: +cmd
; Got answer:
; -->HEADER<-- opcode: QUERY, status: NOERROR, id: 37044
; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 2
;
; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 688de975ecc5465cff2cecb0625893019c7f3e2bc469b2d8 (good)
; QUESTION SECTION:
; vm4.net2.as.uca.es.                IN      A
;
; ANSWER SECTION:
; vm4.net2.as.uca.es.                604800  IN      A      192.168.3.3
;
; AUTHORITY SECTION:
; net2.as.uca.es.                    604800  IN      NS      ns2.net2.as.uca.es.
;
; ADDITIONAL SECTION:
; ns2.net2.as.uca.es.                604800  IN      A      192.168.3.1
;
; Query time: 2 msec
; SERVER: 192.168.3.1#53(192.168.3.1)
; WHEN: Thu Apr 14 21:32:49 UTC 2022
; MSG SIZE rcvd: 125

```

Figura 33: dig

Además, también funciona la resolución DNS hacia hosts de distinta red, por ejemplo, desde **VM3** puedo preguntar por **x.net1.as.uca.es**:

```

vagrant@vm3:~$ nslookup vm1.net1.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:          192.168.3.1
Address:         192.168.3.1#53

Name:   vm1.net1.as.uca.es
Address: 192.168.2.2

vagrant@vm1:~$ nslookup vm4.net2.as.uca.es
;; Got SERVFAIL reply from 127.0.0.53, trying next server
Server:          192.168.2.1
Address:         192.168.2.1#53

Name:   vm4.net2.as.uca.es
Address: 192.168.3.3

```

Figura 34: nslookup desde distinta red

Esto es porque ambas zonas (**net1** y **net2**) se encuentran conectadas al equipo router, por lo que, desde cualquier host (bajo la restricción **trusted** de la configuración) se puede consultar cualquier nombre de dominio.