

Planificación y Operación de Redes

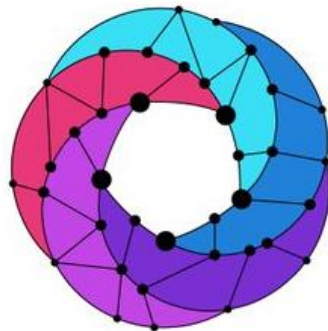
Máster en Ingeniería de Telecomunicación

Curso 2017/2018

Trabajo Final

Servicio de Red en OSM.

Open Source
MANO



Grupo 2

Ismael Martín Saldaña
Francisco José Díaz Romero
Francisco Javier Ortiz Bonilla
Alberto Fuentes Muñoz

Índice

1.	Pruebas previas	3
1.1.	Pingpong NS	3
1.2.	Prueba1 NS.....	3
1.3.	Docker con servidor DNS en Python	4
1.3.1.	Desplegar el docker del servidor DNS como NS.....	5
1.4.	Docker Ubuntu con servidor DNS en Python	6
1.5.	Docker con servidor Apache	8
2.	Servidor DNS protegido.....	10
2.1.	Forwarder + servidor DNS	10
2.2.	Snort + servidor DNS	12

La contraseña del usuario 'usuario' de nuestra máquina es 1234.

1. Pruebas previas

Para intentar familiarizarnos con la plataforma y entender cómo funcionan los dockers, funciones virtuales, servicios virtuales, ficheros yaml, etc. hemos realizado una serie de pruebas previas al servicio de red que queríamos implementar.

En este punto se van a comentar brevemente dichas pruebas.

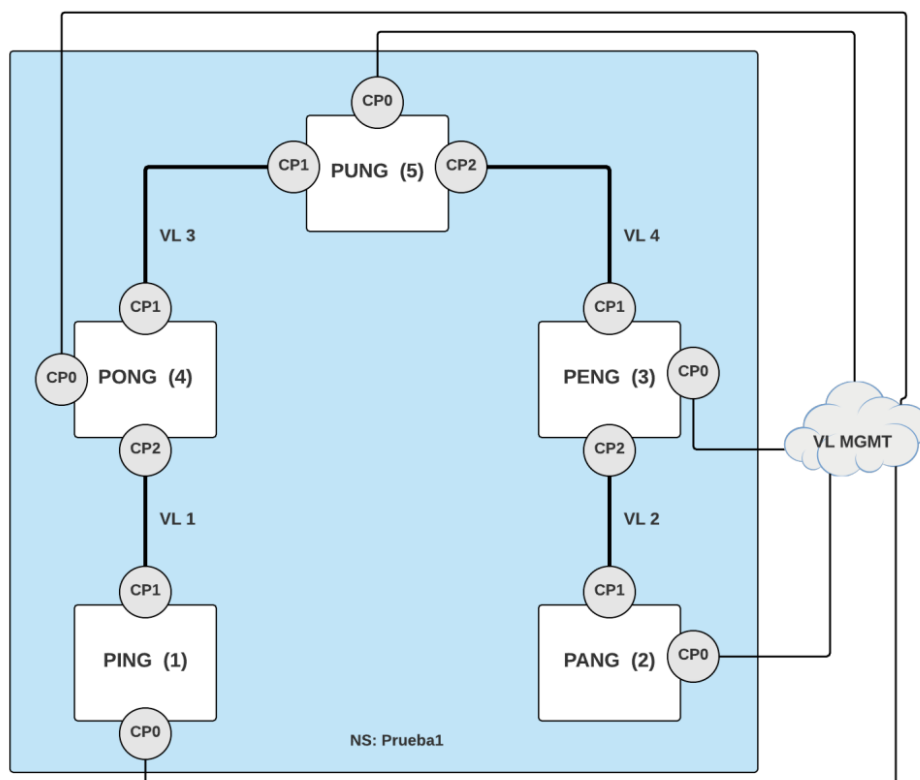
1.1. Pingpong NS

Evidentemente empezamos siguiendo el tutorial que viene en la página de OSM para implementar el servicio pingpong. Tomando como base los ficheros yaml de este ejemplo, pudimos empezar a realizar pruebas y experimentos.

1.2. Prueba1 NS

Los ficheros de esta prueba se encuentran en el directorio **/home/usuario/pruebas/prueba1**

A partir del ejemplo de pingpong, creamos el siguiente escenario:



Es un escenario muy simple, realizado con el propósito de crear distintos enlaces virtuales entre VNFs en redes privadas y no en la de gestión. Todas las VNF tienen una única VDU con una imagen de Ubuntu trusty.

Para instanciar este NS:

```
$ cd /home/usuario/pruebas/prueba1
$ ./crearprueba1.sh
```

Podemos acceder a los contenedores de cada VDU y realizar pings entre las distintas VNF. Todas son alcanzables entre sí a través de la red de gestión, y aquellas que están conectadas por un link virtual también son alcanzables a través de la red privada de dicho link.

Para eliminar este NS:

```
$ cd ~/pruebas/prueba1
$ ./eliminarprueba1.sh
```

1.3. Docker con servidor DNS en Python

Los ficheros de esta prueba se encuentran en el directorio **/home/usuario/pruebas/huliodns0**

Con el objetivo de implementar un servidor DNS como función virtual, probamos un Docker con un servidor DNS escrito en Python, hecho por Samuel Colvin:

<https://github.com/samuelcolvin/dnserver>

Los ficheros Dockerfile, dnserver.py y example_zones.txt de este proyecto están copiados en el directorio **/home/usuarios/pruebas/huliodns0/docker**

Este servidor DNS escucha por defecto peticiones en la dirección **localhost**, por si hacemos una petición a otra dirección distinta (la dirección de su interfaz eth0, por ejemplo), no contesta.

Para poder enviar peticiones desde fuera, hacia la IP de la interfaz eth0 del servidor, es necesario realizar el siguiente cambio en el archivo dnserver.py.

Cambiamos las líneas 153 y 154:

```
udp_server = DNSServer(resolver, port=port)
tcp_server = DNSServer(resolver, port=port, tcp=True)
```

Por lo siguiente:

```
ip = os.popen('ifconfig eth0 | grep "addr:" | cut -d: -f2 | cut -d" " -f1')
ip = ip.read()
ip = ip[:-1]
udp_server = DNSServer(resolver, port=port, address=ip)
tcp_server = DNSServer(resolver, port=port, address=ip, tcp=True)
```

Ahora podemos crear la imagen del servidor DNS, **huliodns0**, ejecutar el docker y probar a enviar una petición de resolución de nombre.

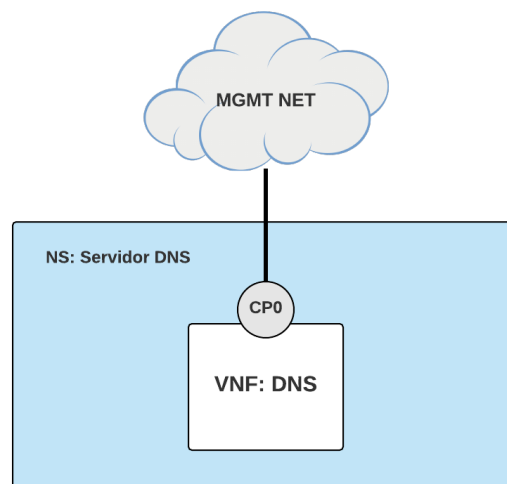
```
$ cd /home/usuario/pruebas/huliodns0/docker
$ ./crearimagen.sh
$ ./rundns.sh
$ ./consulta.sh marca.com
```

```
usuario@OSM2: ~/pruebas/dns2
usuario@OSM2:~/pruebas/dns2$ ./crearimagen.sh > /dev/null
usuario@OSM2:~/pruebas/dns2$ ./rundns.sh &
[1] 6209
usuario@OSM2:~/pruebas/dns2$ 15:08:13: loading zone file "/zones/zones.txt":
15:08:13: 1: example.com. 300 IN A 1.2.3.4
15:08:13: 2: example.com. 300 IN CNAME whatever.com.
15:08:13: 3: example.com. 300 IN MX 5 whatever.com.
15:08:13: 4: example.com. 300 IN MX 10 mx2.whatever.c
15:08:13: 5: example.com. 300 IN MX 20 mx3.whatever.c
15:08:13: 6: example.com. 86400 IN NS ns1.whatever.com.
15:08:13: 7: example.com. 86400 IN NS ns2.whatever.com.
15:08:13: 8: example.com. 300 IN TXT "hello this is so
15:08:13: 9: example.com. 86400 IN SOA ns1.example.com.
15:08:13: 10: testing.com. 300 IN TXT "one long value:
tyiQ70PVR00aNy9Iyklvu91Kuhbyi6l80RrdnuqlvjM//xjaB6DGx8+m1ENML8PEdSFbKQbh9akm2bkN
KCAGEA26JaFWZUedlqcBziAsqZ/LzTF2ASxJYuJ5sk"
15:08:13: 10 zone resource records generated from zone file
15:08:13: starting DNS server on port 53, upstream DNS server "8.8.8.8"
15:09:25: no local zone found, proxying marca.com.[A]
15:09:36: found zone for example.com.[A], 1 replies

usuario@OSM2:~/pruebas/dns2
Last login: Tue Jun 26 12:32:26 2018 from 88.15.116.79
usuario@OSM2:~/pruebas/dns2$ cd pruebas/dns2/
usuario@OSM2:~/pruebas/dns2$ ./consulta.sh marca.com
[sudo] password for usuario:
marca.com 172.17.0.4
Server: 172.17.0.4
Address: 172.17.0.4#53
Non-authoritative answer:
Name: marca.com
Address: 193.110.128.82
usuario@OSM2:~/pruebas/dns2$ ./consulta.sh example.com
example.com 172.17.0.4
Server: 172.17.0.4
Address: 172.17.0.4#53
Name: example.com
Address: 1.2.3.4
usuario@OSM2:~/pruebas/dns2$
```

1.3.1. Desplegar el docker del servidor DNS como NS

Para desplegar un servicio de red que utilice la imagen de este Docker, se ha intentado implementar el siguiente escenario:



Para instanciar este NS:

```
$ cd /home/usuario/pruebas/huliodns0/
$ ./creardns.sh
```

Sin embargo, no hay manera de echarlo a funcionar. Al instanciar el NS siempre aparece el siguiente error:

```
usuario@OSM2: ~/pruebas/huliodns0
usuario@OSM2:~/pruebas/huliodns0$ ./creardns.sh > /dev/null
usuario@OSM2:~/pruebas/huliodns0$ Error response from daemon: Container c6ded75f0e584b735eeb816fcl83018097effda496
bd27cf67df693976c6585 is not running
^C
usuario@OSM2:~/pruebas/huliodns0$ osm ns-list
```

ns instance name	id	operational status	config status	detailed status
pruebadns40	15937e00-bde9-46ce-9403-bb570f682223	failed	init	ERROR Waiting ns r

```

ready at RO. RO_id=8ef1f2b3-69b7-4308-af7c-3b2f02934597: Conflict: Conflict (HTTP 409) |
usuario@OSM2:~/pruebas/huliodns0$
```

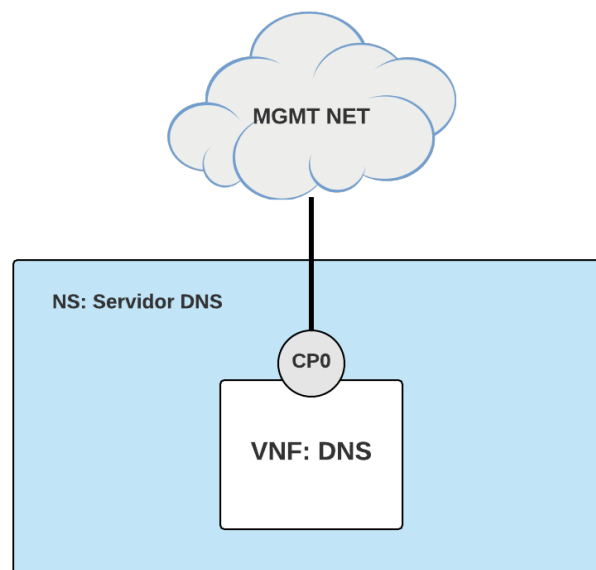
Para eliminar este NS:

```
$ cd ~/pruebas/pruebal
$ ./eliminardns.sh
```

1.4. Docker Ubuntu con servidor DNS en Python

Los ficheros de esta prueba se encuentran en el directorio **/home/usuario/pruebas/huliodns1**

El esquema del NS de esta prueba es el siguiente:



Ya que no conseguimos arrancar el NS utilizando el Docker anterior, decidimos realizar una imagen desde 0, que utilizase un Ubuntu y dentro se ejecutara el servidor DNS en python. El problema es que para Ubuntu la versión 3.6 de Python no está disponible, por lo que hemos tenido que utilizar otro servidor DNS distinto al anterior, que funciona con la versión 3.5 de Python.

El servidor DNS que utilizamos ahora fue escrito también por Samuel Colvin, y podemos encontrarlo aquí:

<https://gist.github.com/samuelcolvin/ca8b429504c96ee738d62a798172b046>

Al igual que antes, tenemos que realizar unos cambios en el código para que el servidor escuche en la IP de la interfaz eth0.

Cambiamos las líneas 113-116:

```
servers = [
    DNSServer(resolver, port=5053, address='localhost', tcp=True),
    DNSServer(resolver, port=5053, address='localhost', tcp=False),
]
```

Por lo siguiente:

```
ip = os.popen('ifconfig eth0 | grep "addr:" | cut -d: -f2 | cut -d" " -f1')
ip = ip.read()
ip = ip[:-1]
servers = [
    DNSServer(resolver, port=5053, address=ip, tcp=True),
    DNSServer(resolver, port=5053, address=ip, tcp=False),
]
```

```
FROM ubuntu:16.04

LABEL maintainer "Hulio Hulian German"

RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y net-tools

RUN apt-get install -y python3
RUN apt-get install -y python3-pip

RUN pip3 install dnslib

ADD ./dns_server.py /home/root/dns_server.py

EXPOSE 5053/tcp
EXPOSE 5053/udp

CMD ["python3", "/home/root/dns_server.py"]
```

Ahora, creamos la imagen del docker, instanciamos el NS y probamos a realizar una consulta:

```
$ cd /home/usuario/pruebas/huliodns1/docker
$ ./crearimagen.sh
$ cd ..
$ ./creardns.sh
$ sudo docker exec -i mn.dcl_pruebadns30.dnserver30.1.dns30 ifconfig
eth0 | grep addr:
$ nslookup -port=5053 example.com 172.17.0.2
```

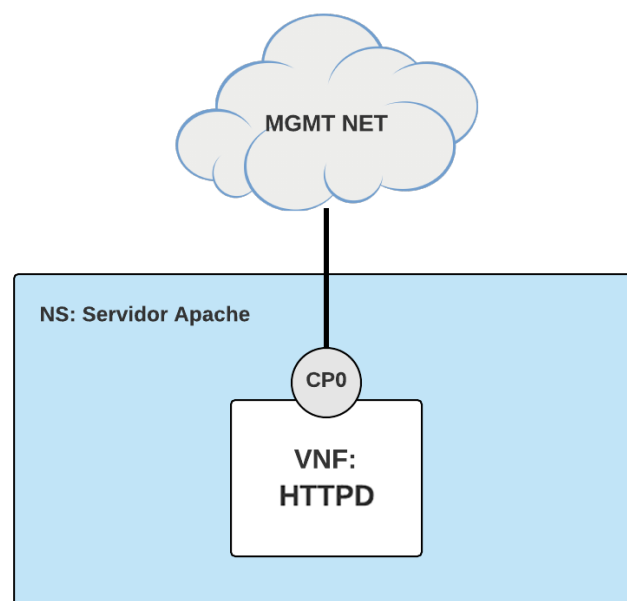
```
usuario@OSM2: ~/pruebas/huliodns1
Untagged: huliodns1:latest
usuario@OSM2:~/pruebas/huliodns1$ cd docker/
usuario@OSM2:~/pruebas/huliodns1/docker$ ./crearimagen.sh > /dev/null
usuario@OSM2:~/pruebas/huliodns1/docker$ cd ..
usuario@OSM2:~/pruebas/huliodns1$ ./creardns.sh > /dev/null
usuario@OSM2:~/pruebas/huliodns1$ osm ns-list
+-----+-----+-----+-----+-----+
| ns instance name | id | operational status | config status | detailed status |
+-----+-----+-----+-----+-----+
| pruebadns30      | 3484711d-d6e8-48fc-9128-8326e05f14c2 | running | configured | done |
+-----+-----+-----+-----+-----+
usuario@OSM2:~/pruebas/huliodns1$ sudo docker exec -i mn.dcl_pruebadns30.dnserver30.1.dns30 ifconfig eth0 | grep ad
dr:
    inet addr:172.17.0.2 Bcast:172.17.255.255 Mask:255.255.0.0
usuario@OSM2:~/pruebas/huliodns1$ nslookup -port=5053 example.com 172.17.0.2
Server:
    172.17.0.2
Address:
    172.17.0.2#5053

Name:   example.com
Address: 1.2.3.4
usuario@OSM2:~/pruebas/huliodns1$
```

Observamos que el NS funciona correctamente.

1.5. Docker con servidor Apache

Los ficheros de esta prueba se encuentran en el directorio **/home/usuario/pruebas/httpd**
El esquema del NS de esta prueba es el siguiente:



Los ficheros yaml para la VNFD y el NSD son iguales que para la prueba anterior con un servidor DNS, pero cambiando la imagen a usar (en este caso un docker con un servidor Apache) y todos los nombres e ids de las VNFD, interfaces, etc.

La imagen del docker httpd la hemos bajado directamente del repositorio:

```
$ docker pull httpd
```

Y podemos consultar el contenido del Dockerfile y los demás ficheros aquí:

<https://github.com/docker-library/httpd/>

Lo cual nos ha sido útil para saber que hay que ejecutar el comando **/usr/local/bin/httpd-foreground** en el docker tras instanciar el NS.

Instanciamos el NS y probamos a realizar una consulta HTTP:

```
$ cd /home/usuario/pruebas/httpd
$ ./crearhttpd.sh

$ sudo docker exec -i mn.dc1_pruebapache10.httpd10.1.httpd10 ip addr |
grep 172
$ curl 172.17.0.2
```

The screenshot shows two terminal windows. The top window is titled 'usuario@OSM2: ~/pruebas/httpd' and shows the execution of a script to create a Docker container. The output includes messages from the Apache httpd service, indicating it is configured and resuming normal operations. The bottom window is titled 'usuario@OSM2: ~/pruebas/huliodns1' and shows the execution of 'oam ns-list' and 'sudo docker exec' to run a curl command against the container. The output of the curl command shows an HTML response: '<html><body><h1>It works!</h1></body></html>'. The terminal output for the first window is as follows:

```
usuario@OSM2:~/pruebas/httpd$ ./crearhttpd.sh > /dev/null
usuario@OSM2:~/pruebas/httpd$ [Tue Jun 26 16:41:42.390633 2018] [mpm_event:notice] [pid 25:tid 140559322371968] AH0
Apache/2.4.33 (Unix) configured -- resuming normal operations
[Tue Jun 26 16:41:42.390764 2018] [core:notice] [pid 25:tid 140559322371968] AH00094: Command line: 'httpd -D FOREG
```

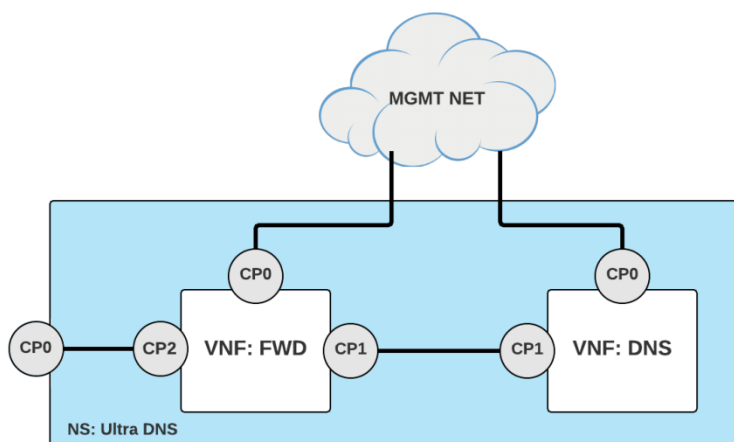
The terminal output for the second window is as follows:

```
usuario@OSM2:~/pruebas/huliodns1$ oam ns-list
+-----+-----+-----+-----+-----+
| ns instance name | id | operational status | config status | detailed status |
+-----+-----+-----+-----+-----+
| pruebapache10 | afa257d9-4070-4bb5-afe9-29ab1d9430da | running | configured | done |
+-----+-----+-----+-----+-----+
usuario@OSM2:~/pruebas/huliodns1$ sudo docker exec -i mn.dc1_pruebapache10.httpd10.1.httpd10 ip addr | grep 172
inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
usuario@OSM2:~/pruebas/huliodns1$ curl 172.17.0.2
<html><body><h1>It works!</h1></body></html>
usuario@OSM2:~/pruebas/huliodns1$
```

Observamos que el NS funciona correctamente.

2. Servidor DNS protegido

Como escenario final para el trabajo se había pensado implementar un NS compuesto por dos VNFD, una con un servidor DNS, y otra con un firewall o un IDS que estuviese delante del DNS y filtrase el tráfico que llega a él. El escenario sería el siguiente:

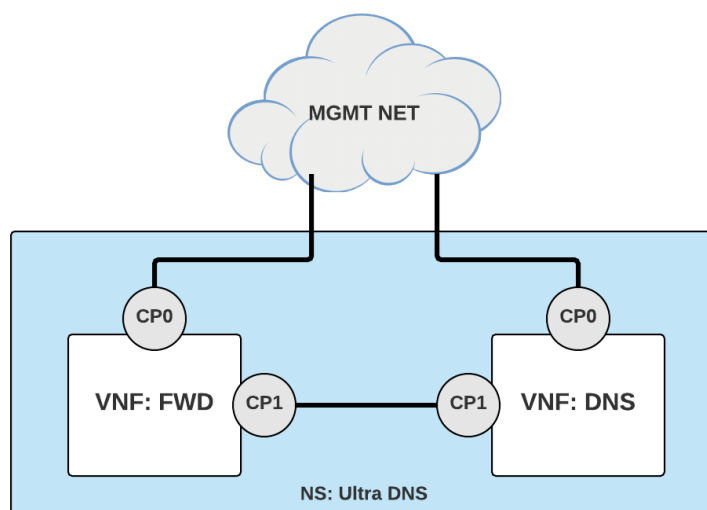


De forma que las peticiones DNS llegarían al CP0 del NS, atravesarían el firewall o IDS y acabarían en el DNS.

Sin embargo, no hemos sido capaces de conseguir asociar un connection-point del nsd con un connection point de un VNF, así que lo que se ha implementado al final es lo siguiente.

2.1. Forwarder + servidor DNS

Se ha implementado el siguiente NS:



Los ficheros de este NS se encuentran en el directorio
/home/usuario/pruebas/ultradns/forwarder

El NS consiste en dos VNFD, un servidor DNS y un forwarder. La idea es que las peticiones DNS se envíen al forwarder (a través de la red de gestión, CP0) y este las reenvíe al DNS (mediante el link virtual que une ambos CP1). El forwarder idealmente sería un firewall, aunque para el propósito de este trabajo se ha quedado en un simple reenviador de tráfico.

La imagen del DNS es **huliodns1**, expuesta en el punto 1.4.

La imagen del FWD se ha hecho a partir del siguiente Dockerfile.

```
FROM ubuntu:16.04

LABEL maintainer "Hulio Hulian German"

RUN apt-get update && apt-get -y upgrade
RUN apt-get install -y net-tools

RUN apt-get install -y iptables

ADD ./config.sh /home/root/config.sh
RUN chmod +x /home/root/config.sh

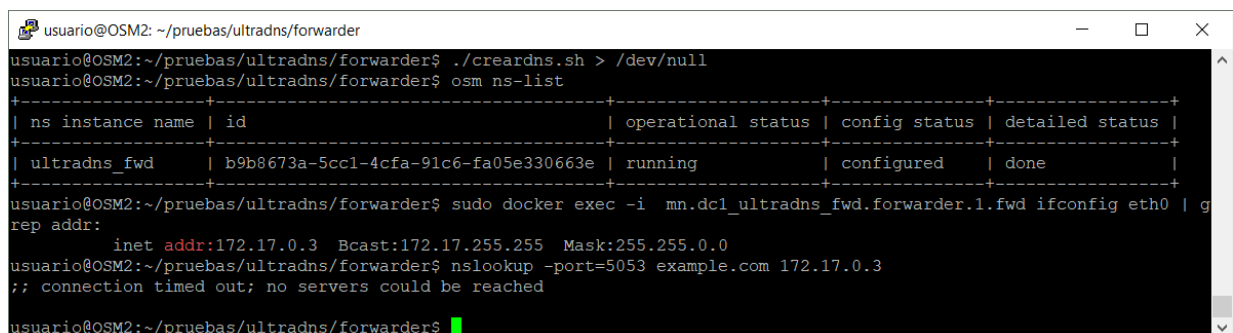
EXPOSE 5053/tcp
EXPOSE 5053/udp

CMD ["/bin/bash", "/home/root/config.sh"]
```

```
$ cd /home/usuario/pruebas/ultradns/forwarder/docker
$ ./crearimagen.sh
```

Instanciamos el NS y probamos a realizar una consulta DNS.

```
$ cd /home/usuario/pruebas/ultradns/forwarder
$ ./creardns.sh
$ sudo docker exec -i mn.dc1_ultradns_fwd.forwarder.1.fwd ifconfig eth0
| grep addr:
$ nslookup -port=5053 example.com 172.17.0.3
```



```
usuario@OSM2: ~/pruebas/ultradns/forwarder
usuario@OSM2:~/pruebas/ultradns/forwarder$ ./creardns.sh > /dev/null
usuario@OSM2:~/pruebas/ultradns/forwarder$ osm ns-list
+-----+-----+-----+-----+-----+
| ns instance name | id | operational status | config status | detailed status |
+-----+-----+-----+-----+-----+
| ultrasns_fwd | b9b8673a-5cc1-4cfa-91c6-fa05e330663e | running | configured | done |
+-----+-----+-----+-----+-----+
usuario@OSM2:~/pruebas/ultradns/forwarder$ sudo docker exec -i mn.dc1_ultradns_fwd.forwarder.1.fwd ifconfig eth0 | g
rep addr:
    inet addr:172.17.0.3 Bcast:172.17.255.255 Mask:255.255.0.0
usuario@OSM2:~/pruebas/ultradns/forwarder$ nslookup -port=5053 example.com 172.17.0.3
;; connection timed out; no servers could be reached
usuario@OSM2:~/pruebas/ultradns/forwarder$
```

Pero no funciona. Podemos probar a entrar al forwarder y enviar desde ahí la petición al servidor DNS:

```

root@dc1_ultradns_fwd: /
usuario@0SM2:~/pruebas/ultradns/forwarder$ sudo docker exec -it mn.dc1.ultradns_fwd.forwarder.1.fwd /bin/bash
root@dc1_ultradns_fwd:/# nslookup -port=5053 example.com 192.168.100.103
bash: nslookup: command not found
root@dc1_ultradns_fwd:/# apt install -y dnsutils > /dev/null
root@dc1_ultradns_fwd:/# nslookup -port=5053 example.com 192.168.100.103
Server:      192.168.100.103
Address:     192.168.100.103#5053

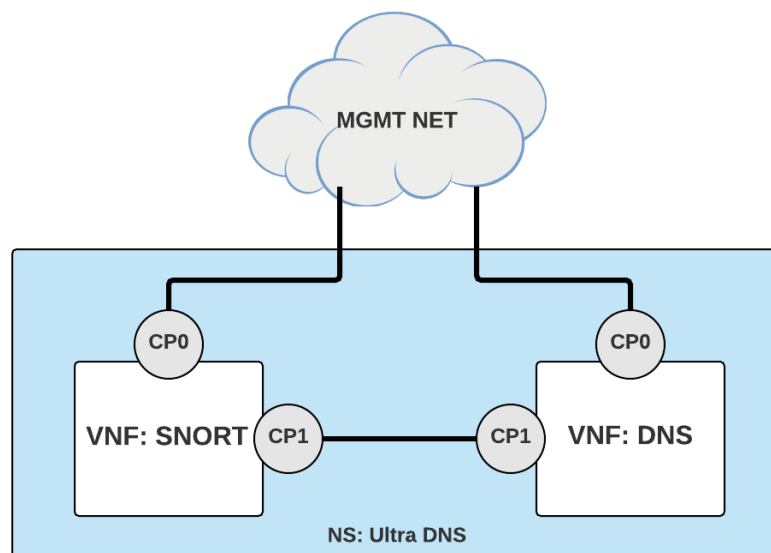
Name:   example.com
Address: 1.2.3.4
root@dc1_ultradns_fwd:/#

```

Y vemos que sí contesta el servidor DNS. El problema está en el forwarder, que no hace bien su trabajo.

2.2. Snort + servidor DNS

Adicionalmente, se ha tratado de implementar el siguiente NS, como alternativa al anterior:



Los ficheros de este NS se encuentran en el directorio
/home/usuario/pruebas/ultradns/snort

El esquema es igual, y la idea prácticamente la misma, pero cambiando el forwarder por un snort.

La imagen del docker de snort se ha encontrado aquí:

<https://github.com/sonata-nfv/son-examples/tree/master/vnfs/sonata-snort-ids-vnf-docker>

Estos ficheros se han copiado en el directorio:
/home/usuario/pruebas/ultradns/snort/docker

Creamos la imagen del snort, **huliosnort**, de la siguiente forma:

```
$ cd /home/usuario/pruebas/ultradns/snort/docker
$ ./crearimagen.sh
```

Instanciamos el NS y probamos a realizar una consulta DNS.

```
$ cd /home/usuario/pruebas/ultradns/snort
$ ./creardns.sh
$ sudo docker exec -i mn.dc1_ultradns_snort.snort.1.snort ifconfig eth0
| grep addr:
$ nslookup -port=5053 example.com 172.17.0.3
```

```

usuario@OSM2: ~/pruebas/ultradns/snort
usuario@OSM2:~/pruebas/ultradns/snort$ ./creardns.sh > /dev/null
device br0 already exists; can't create bridge with the same name
usuario@OSM2:~/pruebas/ultradns/snort$ osm ns-list
+-----+-----+-----+-----+-----+
| ns instance name | id | operational status | config status | detailed status |
+-----+-----+-----+-----+-----+
| ultradns_snort | 1990c035-2b30-4768-ac3e-af82fdec254e | running | configured | done |
+-----+-----+-----+-----+-----+
usuario@OSM2:~/pruebas/ultradns/snort$ docker exec -it mn.dc1_ultradns_snort.snort.1.snort ifconfig eth0 | grep ad
dr:
    inet addr:172.17.0.3 Bcast:172.17.255.255 Mask:255.255.0.0
usuario@OSM2:~/pruebas/ultradns/snort$ nslookup -port=5053 example.com 172.17.0.3
;; connection timed out; no servers could be reached
usuario@OSM2:~/pruebas/ultradns/snort$
  
```

Pero tampoco hemos conseguido que funcione así.