Documentación proyecto Doraemon

Francisco Ucles Ayllón
DNI: 49170940E
Paula Marín Turpín
DNI: 48841926T
Titulación: PCEO Grado en Matemáticas e Ingeniería Informática
Curso: 5º PCEO
Asignaturas: Administración de Sistemas Operativos y Redes
Administración Avanzada de Redes

Universidad de Murcia

Facultad de Informática

Índice

Índice de figuras

1. Introducción

Esta memoria corresponde al Proyecto DORAEMON que tiene como objetivo construir la infraestructura básica de red sobre la que se desplegarán los servicios y mecanismos de seguridad solicitados en el enunciado. Cada organización se ha desplegado en un equipo físico distinto, utilizando VirtualBox como entorno de virtualización y empleando máquinas Linux (Fedora y Ubuntu Server) para la configuración de routers y hosts.

2. Descripción general de la topología

Cada organización del esquema se ha desplegado en un equipo físico distinto (PC anfitrión), de modo que cada entorno virtual queda aislado físicamente, emulando redes de distintas sedes. Dentro de cada equipo físico:

- Existe un router principal con dos interfaces:
  - Una interfaz conectada a la red interna de su organización, donde residen los hosts y servidores locales.
  - Una interfaz en modo NAT, que proporciona salida a Internet.
- Cada organización cuenta con uno o varios hosts que solo poseen conexión de tipo "red interna", es decir, no pueden salir directamente a Internet, sino únicamente comunicarse con su router local.

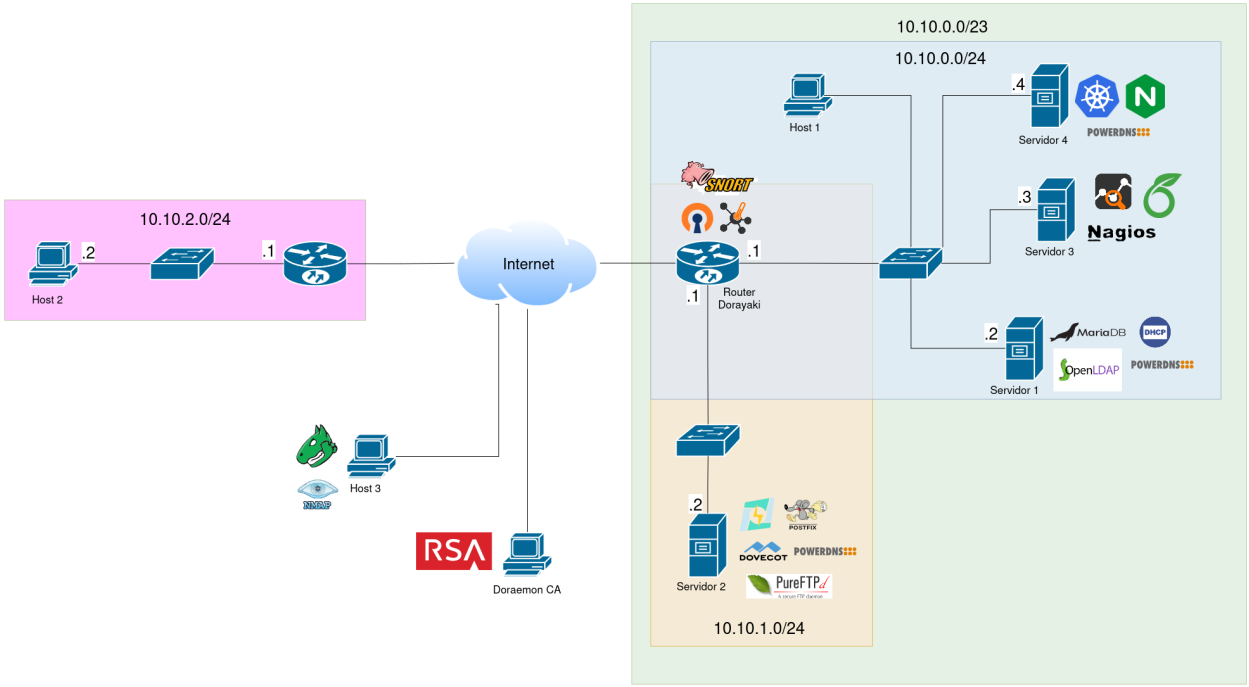


Figura 1: Esquema de la topología para el proyecto DORAEMON

3. Configuración de la red

La verificación de conectividad se encontrará en el Anexo A.

3.1. Configuración de red en VirtualBox

La configuración de red se ha realizado íntegramente desde VirtualBox, preparando cada máquina para reflejar la topología definida. Las organizaciones que componen la infraestructura, aunque parecidas, merecen ser estudiadas por separado:

- Sede "No confiable": en este caso solo disponemos de un router y un host. Por tanto:
  - Router: adaptador NAT para la salida a Internet y red interna para comunicarse con el host (10.10.2.0/24).
  - Host: solamente posee la red interna, del mismo nombre que la de su router asociado (10.10.2.0/24).
- Sede "Dorayaki": por el contrario, aquí encontramos 2 subredes diferenciadas, una pública y otra privada. Así:
  - Router: adaptador NAT para la salida a Internet y 2 redes internas, una para la subred privada (10.10.0.0/24) y otra la para conexión con la pública (10.10.1.0/24).
  - Servidores y host: al igual que en el caso anterior solo disponen de la red interna que los conecta con su router, teniendo en cuenta en cada caso a qué subred pertenecen.

Esta configuración garantiza que los hosts solo pueden salir a Internet a través de su router y que no hay interferencia entre organizaciones.

3.2. Configuración de IP y rutas

Una vez definida la estructura de redes en VirtualBox, se procede a la asignación de las direcciones IP y rutas correspondientes a cada máquina, asegurando la coherencia con el esquema de direccionamiento establecido para la topología.

3.2.1. Organización No confiable

El router actúa como pasarela entre la red interna y el exterior. Su interfaz conectada a la red interna ha sido configurada con una dirección del rango `10.10.2.0/24`, concretamente `10.10.2.1`, mientras que la interfaz externa (modo NAT) obtiene la dirección de manera automática mediante DHCP. El host, por su parte, ha sido configurada con la IP `10.10.2.2/24`, utilizando como puerta de enlace predeterminada la dirección del router.

Los comandos empleados fueron:

```
router-nc @ router-nc : ~  $ nmcli con mod enp0s8 ipv4.addresses 10.10.2.1/24
router-nc @ router-nc : ~  $ nmcli con mod enp0s8 ipv4.method manual
router-nc @ router-nc : ~  $ nmcli con mod enp0s3 ipv4.method auto
router-nc @ router-nc : ~  $ nmcli con up enp0s8
router-nc @ router-nc : ~  $ nmcli con up enp0s3
```

```
host @ asorhost2 : ~  $ nmcli con mod enp0s8 ipv4.addresses 10.10.2.2/24
host @ asorhost2 : ~  $ nmcli con mod enp0s3 ipv4.gateway 10.10.2.1
host @ asorhost2 : ~  $ nmcli con mod enp0s8 ipv4.method manual
host @ asorhost2 : ~  $ nmcli con up enp0s8
```

A continuación se incluyen las capturas de pantallas correspondientes a la información detallada arriba:

```
host@asorhost2:~$ sudo cat /etc/NetworkManager/system-connections/enp0s8.nmconnection
[connection]
id=enp0s8
uuid=020ef5f1-1612-4784-a654-b0ff47ccd3c9
type=ethernet
interface-name=enp0s8
timestamp=1760029805

[ethernet]

[ipv4]
address1=10.10.2.2/24
dns=8.8.8.8;
gateway=10.10.2.1
method=manual

[ipv6]
addr-gen-mode=default
method=auto

[proxy]
```

Figura 2: Información de la dirección IP del Host2.

```
router-nc@router-nc:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:95:54:81 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 86057sec preferred_lft 86057sec
    inet6 fd00::a00:27ff:fe95:5481/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 86059sec preferred_lft 14059sec
    inet6 fe80::a00:27ff:fe95:5481/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:35:9c:3b brd ff:ff:ff:ff:ff:ff
    inet 10.10.2.1/24 brd 10.10.2.255 scope global noprefixroute enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::479a:5a2d:ce85:4c40/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figura 3: Información de la dirección IP del router de la "No confiable".

3.2.2. Sede Dorayaki

En esta organización, el router presenta una configuración más compleja al disponer de 3 interfaces activas:

- Interfaz NAT, configurada automáticamente por DHCP.
- La interfaz correspondiente a la red privada (`10.10.0.0/24`) llamada *dorayaki_priv*, configurada con la IP `10.10.0.1` .
- La interfaz para la red pública (`10.10.1.0/24`) llamada *dorayaki_pub*, configurada con la IP `10.10.1.1` .

Los equipos conectados a la red privada (hosts internos) reciben direcciones dentro del rango `10.10.0.x` y utilizan `10.10.0.1` como puerta de enlace, mientras que los servidores públicos usan direcciones del rango `10.10.1.x` con `10.10.1.1` como gateway.

Los comandos utilizados fueron:

```
# Red privada
router_dorayaki @ routerdorayaki : ~  $ nmcli con mod enp0s8 ipv4.addresses 10.10.0.1/24
router_dorayaki @ routerdorayaki : ~  $ nmcli con mod enp0s8 ipv4.method manual
# Red pública
router_dorayaki @ routerdorayaki : ~  $ nmcli con mod enp0s9 ipv4.addresses 10.10.1.1/24
router_dorayaki @ routerdorayaki : ~  $ nmcli con mod enp0s9 ipv4.method manual
# Interfaz NAT
router_dorayaki @ routerdorayaki : ~  $ nmcli con mod enp0s3 ipv4.method auto
router_dorayaki @ routerdorayaki : ~  $ nmcli con up enp0s8
router_dorayaki @ routerdorayaki : ~  $ nmcli con up enp0s9
router_dorayaki @ routerdorayaki : ~  $ nmcli con up enp0s3
```

El servidor de la red pública y el Servidor1 se configuran de forma similar al host de la organización "No confiable", adecuando las direcciones IP.

A continuación se incluyen las capturas de pantallas correspondientes a la información IP de los dispositivos configurados de forma estática:

```
router_dorayaki@routerdorayaki:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:76:05:cd brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
       valid_lft 86169sec preferred_lft 86169sec
    inet6 fd00::a00:27ff:fe76:5cd/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 86172sec preferred_lft 14172sec
    inet6 fe80::a00:27ff:fe76:5cd/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:0b:79:69 brd ff:ff:ff:ff:ff:ff
    inet 10.10.0.1/24 brd 10.10.0.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe0b:7969/64 scope link
       valid_lft forever preferred_lft forever
4: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:0f:41:c8 brd ff:ff:ff:ff:ff:ff
    inet 10.10.1.1/24 brd 10.10.1.255 scope global enp0s9
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe0f:41c8/64 scope link
       valid_lft forever preferred_lft forever
```

Figura 4: Información de la dirección IP del router de Dorayaki.

```
server1_dorayaki@server1dorayaki:~$ sudo cat /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 10.10.0.3/24
      routes:
        - to: default
          via: 10.10.0.1
      nameservers:
        addresses: [8.8.8.8]
```

Figura 5: Información de la dirección IP del Servidor1 de la subred privada de Dorayaki.

```
server2_dorayaki@server2dorayaki:~$ sudo cat /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: no
      addresses:
        - 10.10.1.2/24
      routes:
        - to: default
          via: 10.10.1.1
      nameservers:
        addresses: [8.8.8.8]
```

Figura 6: Información de la dirección IP del Servidor2 de la subred pública de Dorayaki.

En el caso del host de la sede Dorayaki y el otro servidor de red privada vamos a usar DHCP. Hemos configurado el Servidor1 para que actúe como servidor DHCP como veremos más adelante. Por lo tanto, la configuración el host de la sede Doryaki se resume en:

```
host1_dorayaki @ asorhost1 : ~  $ nmcli con mod enp0s3 ipv4.method auto
host1_dorayaki @ asorhost1 : ~  $ nmcli con up enp0s3
```

Con esto la configuración ya estaría completada y sería la resultante:

```
host1_dorayaki@asorhost1:~$ sudo cat /etc/NetworkManager/system-connections/Conexión\ cableada\ 1.nmconnection
[connection]
id=Conexión cableada 1
uuid=a56407b8-43f4-3f88-8905-1c459abf5e95
type=ethernet
autoconnect-priority=-999
interface-name=enp0s3
timestamp=1760008717

[ethernet]

[ipv4]
method=auto

[ipv6]
addr-gen-mode=default
method=auto

[proxy]
```

Figura 7: Información de la dirección IP del Host1.

En el caso del Servidor3 es tan sencillo como configurar netplan de la siguiente manera:

```
server3_dorayaki@server3dorayaki:~$ sudo cat /etc/netplan/50-cloud-init.yaml
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
```

Figura 8: Información de la dirección IP del Servidor3 de la subred privada de Dorayaki.

Más adelante veremos cómo configurar el DHCP para que siempre le dé la misma IP al Servidor3.

3.3. Activación del envío de IP en routers

Para permitir el tráfico entre las redes internas y el exterior, se habilitó el reenvío de paquetes en todas las máquinas:

```
user @ ejemplo : ~  $ sudo sysctl -w net.ipv4.ip_forward=1
```

Esto garantiza que los routers puedan funcionar como gateways.

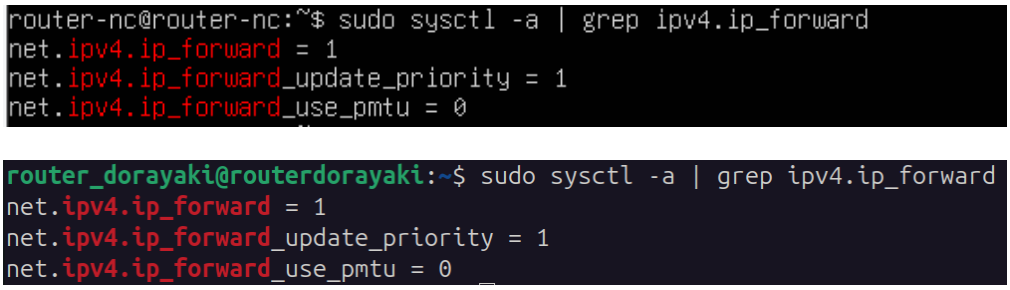


Figura 9: Comprobación de la activación de forward en los routers

3.4. Configuración de NAT

Para permitir que los equipos internos sin acceso directo a Internet pudieran comunicarse con el exterior, fue necesario habilitar la traducción de direcciones de red (NAT) en los routers. El NAT cumple una doble función: por un lado, oculta las direcciones privadas de la red interna, sustituyéndolas temporalmente por la dirección pública del router; y por otro, permite el retorno correcto de los paquetes de respuesta. De este modo, tanto los hosts como los servidores internos pueden acceder a Internet aunque utilicen direcciones privadas (no enrutable).

En el caso de la organización "No confiable", el NAT se configura siguiendo las siguientes reglas de iptables:

```
router-nc @ router-nc : ~  $ sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE
```

Estas reglas realizan la máscara de origen (MASQUERADE) en la interfaz NAT, permiten el tráfico de salida desde la LAN hacia Internet, y autorizan el tráfico de respuesta de vuelta hacia la red interna.

En el caso de la sede Dorayaki, el NAT debe aplicarse de igual manera para ambas redes internas, por lo que se añadieron las reglas siguientes:

```
router_dorayaki @ routerdorayaki : ~  $ sudo iptables -t nat -A POSTROUTING -s 10.10.0.0/23 -o
enp0s3 -j MASQUERADE
```

Con esta configuración, tanto los hosts internos como los servidores pueden acceder a Internet utilizando la dirección pública asignada dinámicamente a la interfaz enp0s3, manteniendo la separación entre redes y sin exponer direcciones privadas al exterior.

Como el Servidor2 va a tener varios servicios públicos: DNS, SMTP, IMAP y POP3 es necesario hacer un DNAT hacia ese servidor con estos servicios. Para ello, se deben introducir los siguientes comandos:

```
router_dorayaki @ routerdorayaki : ~  $ sudo iptables -t nat -A PREROUTING -i enp0s3 -p TCP
-m multiport –dport smtp,imap,pop3 -j DNAT –to 10.10.1.2
router_dorayaki @ routerdorayaki : ~  $ sudo iptables -t nat -A PREROUTING -i enp0s3 -p UDP
–dport domain -j DNAT –to 10.10.1.2
```

Con esto listo, ya estarían los servicios públicos expuestos

3.5. Problemas de seguridad

Con la realización de esta configuración se detectó un problema de seguridad que se podrá subsanar más adelante (mediante uso de firewalls).

El problema en cuestión consiste en lo siguiente: un usuario puede realizar un SSH desde la sede "No confiable.ªl servidor público (hasta ahí todo correcto), pero luego puede hacer un SSH a los dispositivos de la red privada. Eso no debería estar permitido, pues la red privada debería de ser inaccesible desde cualquier punto fuera de la sede Dorayaki.

4. Configuración de los servicios

A continuación se detalla la configuración de todos los servicios que se ofrecerán en la sede Dorayaki. Además, en el Anexo F se incluye una imagen completa de los servicios desplegados sobre contenedores.

4.1. NFS

Se va a implementar un servidor NFS para poder tener un sistema de ficheros centralizado que se pueda usar desde K8s y que las réplicas de un mismo servicio puedan compartir recursos. Nosotros vamos a montarlo en el Servidor3 dentro de la red privada Dorayaki.

Para ello, es necesario instalar NFS mediante el comando:

```
servidor3_dorayaki @ server3dorayaki : ~  $ sudo apt install nfs-kernel-server -y
```

A continuación se debe crear la carpeta que se va a compartir y darsela a un usuario y grupo sin ningún tipo de permisos como es `nobody` y `nogroup`:

```
servidor3_dorayaki @ server3dorayaki : ~  $ sudo mkdir -p /srv/nfs/k8s
servidor3_dorayaki @ server3dorayaki : ~  $ sudo chown nobody:nogroup /srv/nfs/k8s
```

Una vez creado el directorio,hay que indicarle a NFS que tiene que exportar esa misma carpeta poniendo la siguiente línea en el fichero `/etc/exports`. Además, se exportará también un directorio para el nginx que montaremos en K8s solo de lectura:

```
/srv/nfs/k8s        *(rw,sync,fsid=0,crossmnt,no_subtree_check)
/srv/nfs/k8s/nginx  *(ro,sync,no_subtree_check)
```

Figura 10: Configuración del servidor NFS

Finalmente, se exportan los sistemas de fichero con el comando:

```
servidor3_dorayaki @ server3dorayaki : ~  $ sudo exportfs -rav
servidor3_dorayaki @ server3dorayaki : ~  $ sudo systemctl restart nfs-kernel-server
```

Con esto quedaría listo el servidor NFS. Para habilitarlo por la red hay que recordar habilitar el puerto TCP 2049 en el firewall.

4.2. DHCP

Se ha configurado el Servidor1 como un servidor DHCP para la red privada de la sede Dorayaki. Para montarlo ha sido tan sencillo como instalar el servicio de DHCP mediante:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo apt install isc-dhcp-server
```

Una vez instalado solo hay que ir al fichero de configuración `/etc/dhcp/dhcpd.conf` y establecer los parámetros deseados:

```
subnet 10.10.0.0 netmask 255.255.255.0 {
   range 10.10.0.3 10.10.0.199; # Allowed IPs (removed the tunnels IPs)
   option routers 10.10.0.1;    # Default gateway
   option subnet-mask 255.255.255.0;    # Net mask

}
```

Figura 11: Configuración DHCP

Lo más relevante es la entrada `host` que nos permite darle una IP fija a un dispositivo basado en la dirección MAC de su adaptador de red. Esto es lo que nosotros usamos para garantizar que el Servidor3 y el Servidor4 siempre tengan la misma IP:

```
host server3dorayaki {
   hardware ethernet 08:00:27:be:c4:16;
   fixed-address 10.10.0.3;
}

host server4dorayaki {
   hardware ethernet 08:00:27:5b:00:34;
   fixed-address 10.10.0.4;
}
```

Figura 12: Configuración de los hosts con IPs estáticas

Otro aspecto a tener en cuenta es que se ha añadido la dirección del propio Servidor1 como servidor de DNS. Esto es porque, como se verá más adelante, todos los equipos de la red privada deben usarlo como servidor DNS:

```
# option definitions common to all supported networks...
option domain-name "dorayaki.es";
option domain-name-servers 10.10.0.4;
```

Figura 13: Configuración DNS en DHCP

4.3. Servicios de monitorización de red

Todos los servicios que se usarán para monitorizar la red y la disponibilidad del resto de servicios de la red estarán en el Servidor3 dentro de red privada de la sede Dorayaki. La elección de colocarlo en la red privada es porque entendemos que la monitorización solo debe ser accesible para los gestores de red de dentro de la empresa.

4.3.1. Nagios

En esta sección se va a detallar el proceso de instalación y configuración de Nagios. Nosotros hemos decidido instalarlo dentro del Servidor3. La idea es utilizar este servidor como punto de monitorización de todos los otros servicios que se ofrecen tanto en la red privada como en la pública. Nagios nos permite comprobar si el servicio está o no disponible mediante el uso de comandos que son personalizables. Aunque somos conscientes de que se podría crear un comando para todos y cada uno de lo servicios, eso llevaría a desarrollar scripts personalizados y llevaría mucho tiempo. Por lo tanto, solo se va a realizar la monitorización de algunos de los servicios que se encuentran en el proyecto.

Para realizar la monitorización de red propuesta es necesario instalar Nagios Core en el servidor. La instalación se llevó a cabo siguiendo las instrucciones descritas en la guía oficial del proyecto.

Una vez finalizado el proceso de instalación, verificamos que todo está funcionando accediendo desde el navegador a la dirección `http://10.10.0.3/nagios` y accediendo a la web del servicio correctamente tras introducir las credenciales del usuario `nagiosadmin`.

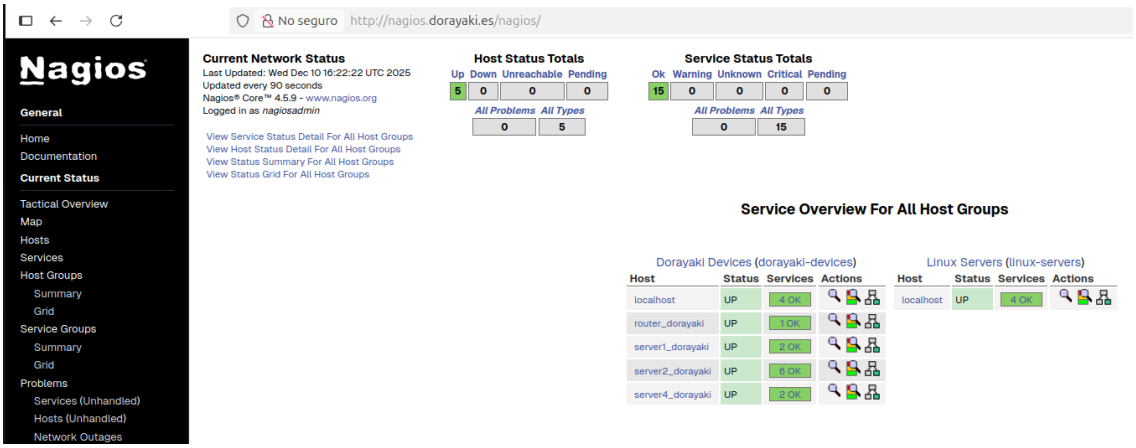


Figura 14: Verificación del estado del servicio Nagios.

4.3.1.1. Configuración de los servidores a monitorizar

Una vez completada la instalación y verificación de Nagios y sus plugins, procedemos a configurar el sistema de monitorización para incluir los distintos servicios y máquinas del entorno virtual. En esta fase definiremos los archivos de configuración necesarios para que el servidor Nagios realice comprobaciones periódicas sobre los aspectos relevantes de nuestra red.

Todas las configuraciones se realizarán en los archivos incluidos dentro del directorio `dorayaki` estándar de Nagios `/usr/local/nagios/etc/objects/`

En la instalación de Nagios, únicamente se incluye por defecto el archivo *localhost.cfg*, que contiene la configuración del host local. Dado que no existe un fichero específico para otros hosts, optaremos por añadir todo el directorio entero donde añadiremos la configuración. Para ello añadiremos la directiva correspondiente al fichero principal *nagios.cfg* mediante la directiva "`cfg_dir`". De esta forma, Nagios reconoce y carga las nuevas definiciones de hosts y servicios creadas para nuestra red.

4.3.1.2. Configuración de los servicios a monitorizar

Para comenzar la monitorización de servicios, configuraremos la comprobación del servicio DNS de Google, cuyo servidor público (8.8.8.8) actúa como host remoto de referencia. Esta monitorización tiene como objetivo verificar quse se tiene acceso a internet.

Debido a que el comando "`check_dns`" no se encuentra definido por defecto en el sistema tras la instalación de Nagios, es necesario crear una nueva definición de comando que especificara cómo ejecutar el plugin correspondiente dentro del fichero `/usr/local/nagios/etc/objects/commands.cfg`:

```
define command {
    command_name    check_dns
    command_line    $USER1$/check_dns -H $ARG1$
}
```

A continuación vamos a añadir el servicio asociado a este comando dentro del fichero *localhost.cfg*:

```
define service {
    use                     generic-service
    host_name               localhost
    service_description     Comprobacion DNS externo
    check_command           check_dns!google.com
    notifications_enabled   0
}
```

Una configuración muy similar se puede realizar para otros muchos servicios como DHCP, DNS interno, SSH, SNMP y otros. Debido a la restricción de tamaño del documento, nos limitaremos a mostrar la configuración del servicio de DNS de Google. Los otros se hacen de forma muy similar y en la sección de comprobación de funcionamiento se mostrarán todos los implementados[1]

4.3.1.3. Redirección de ruta

Aunque la configuración ya estaría completada, vamos a hacer una pequeña modificación para mejorar el acceso. Hasta este momento, cualquier petición a la IP del servidor que pida el recurso `/nagios` serviría la página web de Nagios. Sin embargo, nosotros queremos que se sirva solamente cuando usamos su nombre de DNS concreto, pues de lo contrario podríamos causar conflicto con otros servicios ya existentes. Además, querríamos poder acceder al servicio usando únicamente el nombre DNS sin tener que añadir `/nagios`. Para solucionar todo estos problemas, vamos a modificar el fichero de configuración que se crea en Apache para Nagios.

Para ello, debemos de acceder al fichero de `/etc/apache2/sites-enabled/nagios.conf` y englobar todo el contenido en un `VirtualHost`, y dentro colocar tanto un `ServerName` (para restringir solo al nombre de DNS) y`RedirectMatch` (para evitar tener que poner `/nagios`). El fichero quedaría como sigue:

```
<VirtualHost *:80>
    ServerName nagios.dorayaki.es
    RedirectMatch ^/$ /nagios/
    ScriptAlias /nagios/cgi-bin "/usr/local/nagios/sbin"
    <Directory "/usr/local/nagios/sbin">
        #  SSLRequireSSL
        Options ExecCGI
        AllowOverride None
        <IfVersion >= 2.3>
            <RequireAll>
                Require all granted
                AuthName "Nagios Access"
                AuthType Basic
                AuthUserFile /usr/local/nagios/etc/htpasswd.users
                Require valid-user
            </RequireAll>
        </IfVersion>
```

[1]Todos los implementados tampoco son todos los posibles pues eso llevaría demasiado tiempo de configuración para un escenario de pruebas.

```
        <IfVersion < 2.3>
            Order allow,deny
            Allow from all
            AuthName "Nagios Access"
            AuthType Basic
            AuthUserFile /usr/local/nagios/etc/htpasswd.users
            Require valid-user
        </IfVersion>
    </Directory>
    Alias /nagios "/usr/local/nagios/share"
    <Directory "/usr/local/nagios/share">
    #  SSLRequireSSL
        Options None
        AllowOverride None
        <IfVersion >= 2.3>
            <RequireAll>
                Require all granted
                AuthName "Nagios Access"
                AuthType Basic
                AuthUserFile /usr/local/nagios/etc/htpasswd.users
                Require valid-user
            </RequireAll>
        </IfVersion>
        <IfVersion < 2.3>
            Order allow,deny
            Allow from all
            AuthName "Nagios Access"
            AuthType Basic
            AuthUserFile /usr/local/nagios/etc/htpasswd.users
            Require valid-user
        </IfVersion>
    </Directory>
</VirtualHost>
```

Ahora solo tenemos que recargar el servicio de Apache y lo tendríamos:

```
server3_dorayaki @ server3dorayaki : ~  $ sudo systemctl reload apache2
```

4.3.1.4. Comprobación de funcionamiento

Para comprobar el correcto funcionamiento es necesario acceder al panel de Nagios para observar si se encuentran todos los hosts que hemos definido:

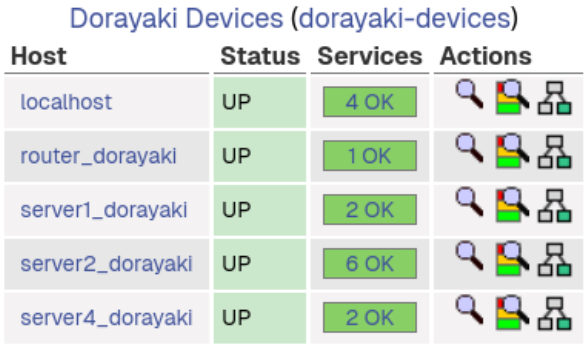


Figura 15: Grupo de servidores en Nagios

Además, si accedemos a cualquiera de los servidores veremos que todo está funcionando correctamente:

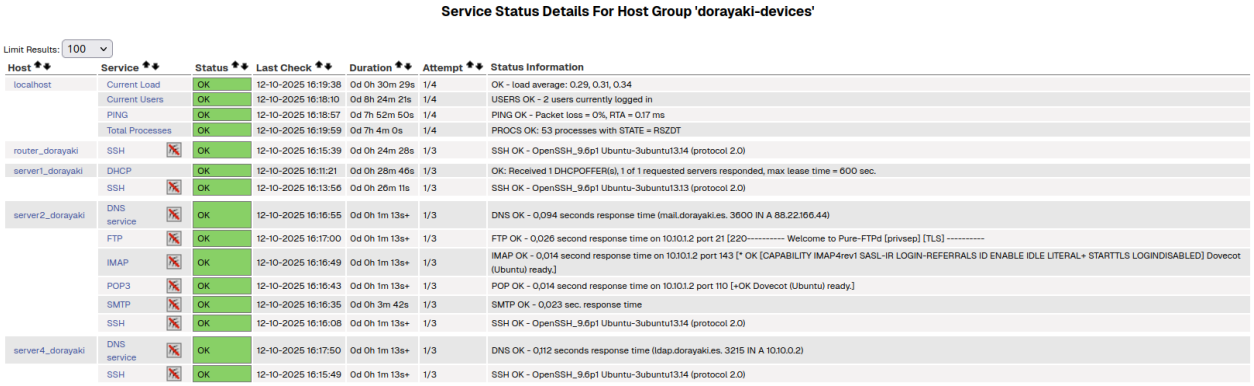


Figura 16: Grupo de servicios monitorizados en Nagios

4.3.2. Ntop

Ntop (y su versión moderna ntopng) es una herramienta de análisis de tráfico en tiempo real que permite observar, clasificar y visualizar el tráfico de la red de manera dinámica. Está especializado en la detección de patrones y posibles anomalías al proporcionar estadísticas detalladas sobre el uso del ancho de banda, protocolos y aplicaciones.

Al igual que Nagios, Ntop se instaló en el Servidor3, situado en la red privada de Dorayaki. Esto trae una problemática: la monitorización completa de la red. Ntop analiza el tráfico de red entrante por sus interfaces y como está en un dispositivo por el que no pasa todo el tráfico, solo estaríamos monitorizando los eventos de red en los que intervenga el Servidor3. Para solucionarlo, hay 2 alternativas: instalar Ntop en el router (punto en el que sí confluyen todas las comunicaciones) o hacer uso de NetFlow/sFlow. Nosotros nos hemos decantado por la segunda opción.

La idea es sencilla: mantener Ntop en el Servidor3 y que sea el router el que envíe los flujos de red al router. Como estamos haciendo uso de Ntop, la mejor manera para lograr esto es usando nProbe. Esta herramienta está desarrollada por la misma empresa que Ntop y permite usar NetFlow para enviar flujos de red a otros dispositivos. Además, como ambos han sido desarrollados por la misma empresa, su integración resulta

relativamente sencilla. El único inconveniente es que nProbe no dispone de una licencia gratuita, solo de una pequeña prueba. Esta te permite registrar un número limitado de flujos. Aunque podríamos optar por otros software completamente gratuitos, consideramos que, en un escenario real donde interesa mucho monitorizar la red, la empresa pagaría la licencia pertinente. Nosotros vamos a realizar la configuración suponiendo que tenemos la licencia, aunque al cabo de un tiempo el programa dejará de registrar flujos.

Vamos a ver los pasos a seguir para realizar correctamente esta monitorización de toda la red.

4.3.2.1. Configuración de nProbe

El primer paso será instalar nProbe dentro del router. Para ello, accedemos a la documentación oficial para añadir el repositorio de nProbe y realizar la instalación mediante el comando `ntop-installer`. Este abrirá un asistente que nos permitirá indicar que efectivamente queremos instalar nProbe.

Una vez completada la instalación se habrá creado un servicio en `systemctl` llamado `nprobe`. Como el objetivo es usar nProbe para monitorizar ciertas interfaces y luego reenviar los datos a Ntopng, no vamos a usar este servicio creado por defecto (pues vamos a crear los nuestros propios). Sin embargo, en lugar de eliminarlo o modificrlo, lo paramos y lo deshabilitamos, dejándolo como plantilla:

```
router_dorayaki @ routerdorayaki : ~  $ sudo systemctl stop nprobe
router_dorayaki @ routerdorayaki : ~  $ sudo systemctl disable nprobe
```

Es el momento de crear los dos servicios nuevos que monitorizarán, cada uno, las interfaces de las redes privadas y públicas de la sede Dorayaki:

```
router_dorayaki @ routerdorayaki : ~  $ sudo p /etc/systemd/system/nprobe.service
/etc/systemd/system/nprobe-dorayaki-private.service
router_dorayaki @ routerdorayaki : ~  $ sudo p /etc/systemd/system/nprobe.service
/etc/systemd/system/nprobe-dorayaki-public.service
```

Modificaremos ligeramente cada servicio para que utilice un fichero de configuración distinto. A continuación, se muestra uno de los servicios (el otro es completamente análogo):

```
[Unit]
Description=nprobe extensible NetFlow v5/v9/IPFIX probe/collector for private dorayaki subnet
After=network.target syslog.target redis.service pf_ring.service cluster.service
Wants=pf_ring.service cluster.service
PartOf=pf_ring.service cluster.service

[Service]
Type=simple

Environment=UNIT_NAME=%N

ExecStartPre=/bin/sh -c '/bin/echo "$(/bin/date) nprobe-dorayaki-private StartPre" >> /var/log/ntop-systemd.log'
ExecStartPre=/bin/sh -c '/bin/sed "/^-g.*$\\|^-G.*\\|^--daemon-mode.*\\|^--pid-file.*/s/^/#/" /etc/nprobe/nprobe-dorayaki-private.conf > /run/nprobe-dorayaki-private.conf'

ExecStart=/usr/bin/nprobe /run/nprobe.conf

ExecStartPost=/bin/sh -c '/bin/echo "$(/bin/date) nprobe-dorayaki-private StartPost" >> /var/log/ntop-systemd.log'
ExecStopPost=-/bin/rm -rf /run/nprobe-dorayaki-private.conf
ExecStopPost=/bin/sh -c '/bin/echo "$(/bin/date) nprobe-dorayaki-private StopPost" >> /var/log/ntop-systemd.log'

Restart=on-abnormal
RestartSec=5

[Install]
WantedBy=multi-user.target
```

Figura 17: Fichero del servicio de nprobe

Para completar esta configuración, es necesario crear sendos ficheros de configuración para indicarle qué interfaz deben escuchar y a qué dirección y puerto deben enviar los datos que recolecten:

```
-i=enp0s8
-E=0:1
-T="@NTOPNG@"
#
#       -n|--collector
#        Specifies the NetFlow collector that will be used by nProbe to send the monitored
#        flows. This option can be specified multiple times to deliver monitored flows to
#        multiple collectors in round-robin mode. To disable flow export to NetFlow collectors
#        specify -n=none
#
# -n=10.0.0.1:2055
-n=none
--ntopng="zmq://ntop.dorayaki.es:5556"
--zmq-probe-mode
```

Figura 18: Fichero de configuración de nprobe

Con todo esto, ya solo restaría arrancar y habilitar los servicios.

4.3.2.2. Configuración de Ntop

El siguiente paso es configurar Ntopng para que escuche los flujos que envía nProbe. Para ello, debemos acceder al fichero de configuración del servicio de ntopng: `/etc/ntopng/ntopng.conf`. Ahí vamos a configurar la interfaz dónde escuchará y qué IPs serán consideradas locales:

```
--community=
-i="tcp://*:5556c"
-m="10.10.0.0/23"
```

Figura 19: Fichero de configuración de ntopng

Con todo esto, solo necesitamos hacer un restart del servicio `ntopng` y quedaría todo listo.

4.3.2.3. Configuración del proxy

Aunque sería posible acabar aquí y simplemente utilizar la IP y el puerto para acceder al dashboard de ntopng, esto es bastante engorroso. Lo mejor es, dado que gracias a Nagios ya tenemos un servidor Apache funcionando, usar un proxy que nos redirija las peticiones. Para ello bastaría modiuficar el fichero `ntopng.conf` dentro de la carpeta `/etc/apache2/sites-available` para que incluya el siguiente contenido:

```
<VirtualHost *:80>
    ServerName ntopng.dorayaki.es
    RequestHeader set X-Forwarded-Proto "http"
    ProxyPreserveHost On
    ProxyRequests Off
    <Location />
        ProxyPass http://127.0.0.1:3000/
        ProxyPassReverse http://127.0.0.1:3000/
        RequestHeader set X-Real-IP %{REMOTE_ADDR}s
        RequestHeader set X-Forwarded-For %{X-Forwarded-For}e
    </Location>
</VirtualHost>
```

Si ahora solo habilitamos el sitio y reiniciamos el servicio de Apache, habremos terminado:

```
server3_dorayaki @ server3dorayaki : ~  $ sudo a2ensite ngopng.conf
server3_dorayaki @ server3dorayaki : ~  $ sudo systemctl reload apache2
```

4.3.2.4. Confirmación de funcionamiento

Para comprobar que todo funciona correctamente, accediendo dashboard de ntopng, podemos ver que tenemos una interfaz llamada `tcp://*:5556c` que será la indicada en el fichero de configuración. Si ahora además realizamos tráfico dentro de la subred pública y dentro de la privada (en el que no intervenga el Servidor3) podremos verlo correctamente:

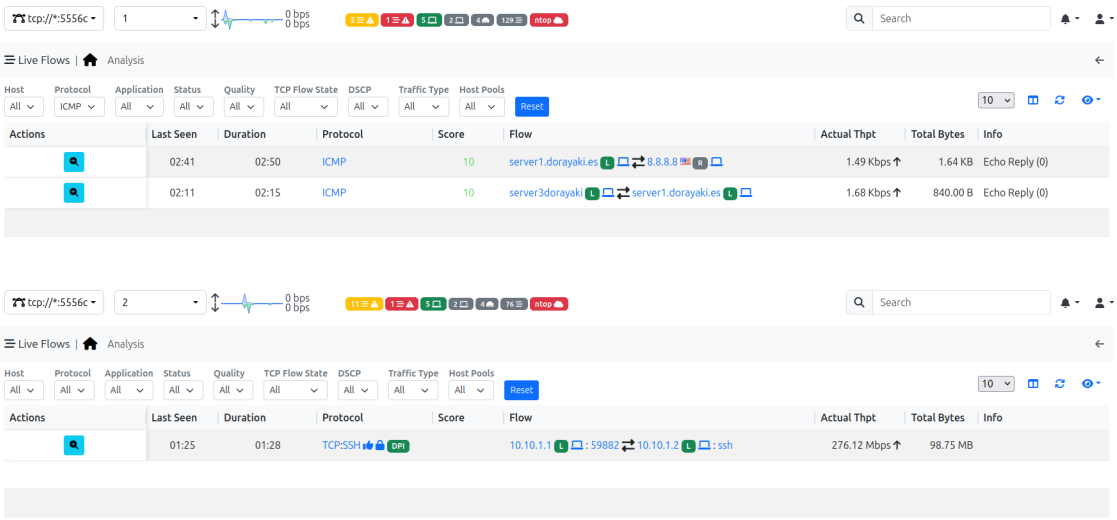


Figura 20: Flujo detectado en ntop

Notese que, dentro de la interfaz, podemos seleccionar un identificador (1 o 2) que representa a cada uno de los flujos. En nuestro caso, el identificador 1 representa la red privada y el 2 la pública. Gracias a todo esto, podemos tener un control exhaustivo y preciso de toda la red de la sede.

4.4. Servicio de base de datos

Hemos decidido implementar en la sede Dorayaki un servicio de bases de datos. La idea es crear una base de datos de tipo SQL que otros servicios o usuarios puedan utilizar. Este servicio, por motivos de seguridad, se encontrará en la red privada. Más concretamente, en el Servidor1.

Aunque hay muchas opciones de bases de datos, hemos decidido utilizar MariaDB, puesto que se trata de un gestor de bases de datos tipo SQL y es completamente gratis y OpenSource.

Para la instalación, para facilitar el proceso, se va a haer uso de Docker[2].

En primer lugar, vamos a crear una red de Docker que usarán los servicios dockerizados que quieran hacer uso de la base de datos. Para ello, hay que ejecutar el siguiente comando:

```
server1_dorayaki @ server1dorayaki : ~  $ docker network create dorayaki_db
```

Ahora, hay que iniciar la instancia de la base de datos. Aunque podría hacerse mediante un comando docker normal, por la sencillez en la ejecución y lo estructurado que resulta, hemos decidido usar docker compose.

[2]Se supondrá instalado Docker dentro del equipo

El docker compose que habría que usar sería el siguiente:

```
services:
  dorayaki_db:
    image: mariadb:latest
    container_name: dorayaki_db
    restart: unless-stopped
    ports:
      - 3306:3306
    environment:
      MARIADB_ROOT_PASSWORD: "asoradareddoraemon"
    volumes:
      - data:/var/lib/mysql:rw
    networks:
      - db_net
volumes:
  data:
    name: dorayaki_db_vol
networks:
  db_net:
    name: dorayaki_db
    external: true
```

Si ejecutamos el compose, lo tendríamos corriendo de forma exitosa. Además, gracias a la política de reinicio, se ejecutará cada vez que arranque el sistema. En este momento, cualquier equipo con un usuario y contraseña puede acceder a la base de datos. La restricción de qué IPs pueden acceder se establecerá usando el firewall.

4.5. DNS

Hemos decidido implementar en la sede Dorayaki un servicio de DNS. La idea es poder acceder a los diferentes servicios que se ofrecen usando nombres y no las IPs. Además, la empresa Dorayaki quiere poder facilitar el uso de sus servicios públicos al resto de usuarios y para ello se precisa de un DNS público.

Para la implementación del DNS público se ha optado por el uso de CyberPanel, pues en él se incluyen otros servicios que nos son interesantes. Es decir, podemos facilitar la implementación de varios servicios con la misma herramienta. Sin embargo, CyberPanel hace uso de PowerDNS como servicio DNS, y este no permite a través de CyberPanel elegir qué dominios se deben mostrar en función de la IP origen. Por lo tanto, si usamos este servidor como único servidor DNS y añadimos los nombres de los servicios privados, permitiríamos que cualquier usuario tuviese acceso a estos datos. Para solventarlo, hemos decidido implementar una arquitectura híbrida: un servidor público mediante CyberPanel en la red pública y un servidor privado solo con PowerDNS en la red privada. De esta manera, el servidor privado proveerá (solo a los usuarios de la red privada) las IPs privadas de cada servicio de la organización. Mientras tanto, el servidor público DNS proveerá la IP pública de los servicios públicos. Con todo esto, garantizamos una gestión adecuada de la resolución de nombres y la seguridad de los servicios. A continuación se detallan ambas implementaciones.

4.5.1. DNS privado

4.5.1.1. DNS Autoritativo

Este servicio se montará dentro del Servidor1. Además, vamos a usar una base de datos SQL para la gestión y el almacenamiento de los registros. Aunque podríamos haber optado por usar ficheros de registros más convencionales, creemos que la base de datos conlleva varios beneficios:

- Los registros se almacenan de forma más organizada y se pueden visualizar las zonas más fácilmente.
- Nos va a permitir tener un lugar centralizado para guardar tanto estos registros privados, como los registros del servidor público

Por tanto, vamos a utilizar la base de datos que ya se encuentra en el propio servidor para gestionar PowerDNS.

Para la instalación, vamos a usar Docker. De nuevo, al igual que con el servicio de BBDD, usaremos un docker compose por su sencillez.

En primer lugar, debemos crear una base de datos y un usuario para que PowerDNS almacene los datos. Para ello, se ejecutan los siguientes comandos dentro del MariaDB Shell como root:

```
CREATE DATABASE pdnsinternal;
CREATE USER 'pdns'@'172.18.%' IDENTIFIED BY 'asoradareddoraemon';
GRANT ALL PRIVILEGES ON pdnsinternal.* TO 'pdns'@'172.18.%';
FLUSH PRIVILEGES;
```

Esto creará un usuario accesible desde cualquier IP dentro de la red creada por docker: dorayaki_db[3]. Una vez creados el usuario y la base de datos, vamos a crear las tablas necesarias. Para ello, usaremos el esquema que ya nos proporciona PowerDNS desde su GitHub oficial y ejecutaremos el siguiente comando:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo mysql -u pdns -p pdnsinternal <schema.mysql.sql
```

Con todo esto, ya tendríamos la base de datos lista. Ahora debemos configurar PowerDNS. Para ello, solo debemos crear el fichero de configuración dentro de la carpeta que tendrá el docker-compose. Lo llamaremos (para mantener nomenclatura) `pdns.conf` y tendrá el siguiente contenido:

```
launch=gmysql
gmysql-host=dorayaki_db
gmysql-port=3306
gmysql-user=pdns
gmysql-password=asoradareddoraemon
gmysql-dbname=pdnsinternal
```

Figura 21: Configuración de PowerDNS para usar MariaDB

Finalmente, solo tenemos que ejecutar el siguiente docker compose:

```
services:
  pdns:
    image: powerdns/pdns-auth-50
    container_name: dorayaki_priv_dns
    restart: unless-stopped
    ports:
      - 10.10.0.2:53:53/udp
    volumes:
      - ./pdns.conf:/etc/powerdns/pdns.conf:ro
    networks:
      - db_net
networks:
  db_net:
    name: dorayaki_db
    external: true
```

Así, ya estaría listo para funcionar. Aunque sí está operativo el servicio, no se estarían resolviendo nombres pues no hay ninguna zona ni registros creados. Para crear el nuestro, vamos a ejecutar los siguientes comandos dentro del shell de MariaDB como el usuario `pdns`:

[3]Para saber qué rango de IPs asigna usamos `docker network inspect dorayaki_db`

```
INSERT INTO domains (name, type) VALUES ('dorayaki.es', 'NATIVE');
INSERT INTO records (domain_id, name, type, content, ttl, prio,
    disabled, auth)
VALUES (1, 'dorayaki.es', 'SOA', 'ns1.dorayaki.es hostmaster.dorayaki.es
        1 10800 3600 604800 3600', 3600, NULL, 0, 1),
       (1, 'dorayaki.es', 'NS', 'ns1.dorayaki.es', 3600, NULL, 0, 1),
       (1, 'ns1.dorayaki.es', 'A', '10.10.0.2', 3600, NULL, 0, 1);

INSERT INTO records (domain_id, name, type, content, ttl, prio)
VALUES (1, 'mail.dorayaki.es', 'A', '10.10.1.2', 3600, 10),
       (1, 'dorayaki.es', 'MX', 'mail.dorayaki.es', 3600, NULL);

INSERT INTO records (domain_id, name, type, content, ttl)
VALUES (1, 'ldap.dorayaki.es', 'A', '10.10.0.2', 3600);

INSERT INTO records (domain_id, name, type, content, ttl)
VALUES (1, 'nagios.dorayaki.es', 'A', '10.10.0.3', 3600),
       (1, 'ntopng.dorayaki.es', 'A', '10.10.0.3', 3600),
       (1, 'overleaf.dorayaki.es', 'A', '10.10.0.3', 3600);
```

Con esto ahora sí que se servirían los nombres de los servicios privados de la sede Dorayaki.

4.5.1.2. DNS Recursor

Aunque esto es suficiente para resolver los nombres de la red, todavía existe un ligero problema. Si le proporcionamos a un sistema Ubuntu tanto el DNS de Google como el nuestro privado mediante DHCP (el primero para dominios externos y el segundo para los internos), siempre consultará el de Google primero y no preguntará al privado. Para solucionar este problema, la solución es implementar un DNS Recursor que sea el único punto de petición de nombres y que reenviará la petición al de Google en caso de ser uno externo y al DNS Autoritativo en caso de resultar una petición de nuestro dominio.

Hemos decidido implementarlo mediante Docker en el Servidor4. Por lo tanto, el servidor que se publicará como servidor DNS será exactamente el Servidor4. Para llevar a cabo el despliegue usaremos una modificación de la imagen oficial de PowerDNS Recursor y un docker-compose para manejarlo de forma más sencilla.

Lo primero que veremos será la configuración del DNS. Esta es sencilla y se especifica mediante un fichero `recursor.yml` que se muestra a continuación:

```
recursor:
  forward_zones:
    - zone: "dorayaki.es"
      forwarders:
        - 10.10.0.2
  forward_zones_recurse:
    - zone: "."
      forwarders:
        - 8.8.8.8
logging:
  common_errors: true
dnssec:
  validation: "off"
```

Aquí se especifica qué zonas se van a servir y a quién hay que hacerle el reenvío de la petición. Además, se deshabilita el DNSSec puesto que el servidor Autoritativo no lo implementa. En cuanto a la imagen de Docker, el Dockerfile que usaremos es el siguiente:

```
FROM powerdns/pdns-recursor-51:latest
COPY recursor.yml /etc/powerdns/recursor.d/recursor.yml
```

Finalmente, solo debemos ejecutar el siguiente docker-compose:

```
services:
  pdns-recursor:
    build: .
    restart: unless-stopped
    ports:
      - 10.10.0.4:53:53/udp
```

Una vez esté ejecutado, ya tendríamos el servidor listo para resolver los dominios (importante habilitar el puerto 53/udp en el firewall)

4.5.2. DNS público

Para el DNS público, como ya se ha comentado, vamos a usar CyberPanel. Su instalación es bastante sencilla y basta con seguir las instrucciones del tutorial. Aunque nosotros vamos a usar SQL remote, **no** debemos marcar esta opción en el instalador. El instalador no termina de funcionar muy bien con esta opción y se generan errores en la instalación. Por lo tanto, realizaremos la migración manualmente.

Una vez se haya instalado el DNS, vamos a configurar CyberPanel para que haga uso del SQL remoto.

En primer lugar, debemos hacer la migración de la base de datos. Para ello, hay que exportar la base de datos creada por CyberPanel y llevarla al Servidor1 para importarla. Para exportarla ejecutamos[4]:

```
server2_dorayaki @ server2dorayaki : ~ $ sudo mysqldump -u root -p –databases cyberpanel >/root/cyberpanel_backup.sql
```

Antes de restaurar los datos, vamos a crear los dos nuevos usuarios que necesitaremos: el que usará CyberPanel para manejar su base de datos (`cyberpanel`) y el que usará para crear más bases de datos (`cyberpanel_admin`). Lo haremos mediante los siguientes comandos dentro del shell de MariaDB como root:

```
CREATE USER 'cyberpanel_admin'@'10.10.1.2' IDENTIFIED BY 'asoradareddoraemon';
GRANT ALL PRIVILEGES ON *.* TO 'cyberpanel_admin'@'10.10.1.2';
FLUSH PRIVILEGES;
```

Una vez hecho esto, restauramos la base de datos:

```
server2_dorayaki @ server2dorayaki : ~ $ sudo mysql -h 10.10.0.2 -u cyberpanel_admin -p </root/cyberpanel_backup.sql
```

Con la base de datos restaurada, debemos crear el usuario que la manejará (desde el shell de MariaDB):

```
CREATE USER 'cyberpanel'@'10.10.1.2' IDENTIFIED BY 'asoradareddoraemon';
GRANT ALL PRIVILEGES ON 'cyberpanel'.* TO 'cyberpanel'@'10.10.1.2';
FLUSH PRIVILEGES;
```

Con MariaDB ya configurado, pasamos a configurar CyberPanel para que lo utilice. Para ello, volviendo al Servidor2, debemos cambiar los datos de conexión de los siguientes archivos:

- `/usr/local/CyberCP/.env`, para el propio CyberPanel
- `/etc/powerdns/pdns.conf`, para PowerDNS de CyberPanel

[4]La contraseña del root se encuentra en el fichero `/etc/cybperpanel/mysqlpassword`

- `/etc/dovecot/dovecot-sql.conf.ext`, para Dovecot de CyberPanel
- `/etc/pure-ftpd/db/mysql.conf`, para Pure-FTP de CyberPanel
- `/etc/postfix/mysql-virtual_*.cf`, para PostFix de CyberPanel

Una vez esté todo modificado, debemos reiniciar todos estos servicios mediante los siguientes comandos:

```
server2_dorayaki @ server2dorayaki : ~ $ sudo systemctl restart lscpd
server2_dorayaki @ server2dorayaki : ~ $ sudo systemctl restart pdns
server2_dorayaki @ server2dorayaki : ~ $ sudo systemctl restart dovecot
server2_dorayaki @ server2dorayaki : ~ $ sudo systemctl restart pure-ftpd
server2_dorayaki @ server2dorayaki : ~ $ sudo systemctl restart postfix
```

Con todo reiniciado, ya tendríamos migrada la base de datos. Ahora podríamos eliminar MariaDB del servidor público:

```
server2_dorayaki @ server2dorayaki : ~ $ sudo apt remove mariadb-server
```

A continuación, procedemos a la configuración del DNS mediante CyberPanel.

Para crear una zona DNS, solo es necesario ir al apartado DNS y, dentro, a la sección **Create DNS Zone**, meter el nombre del dominio (dorayaki.es) y darle a crear:

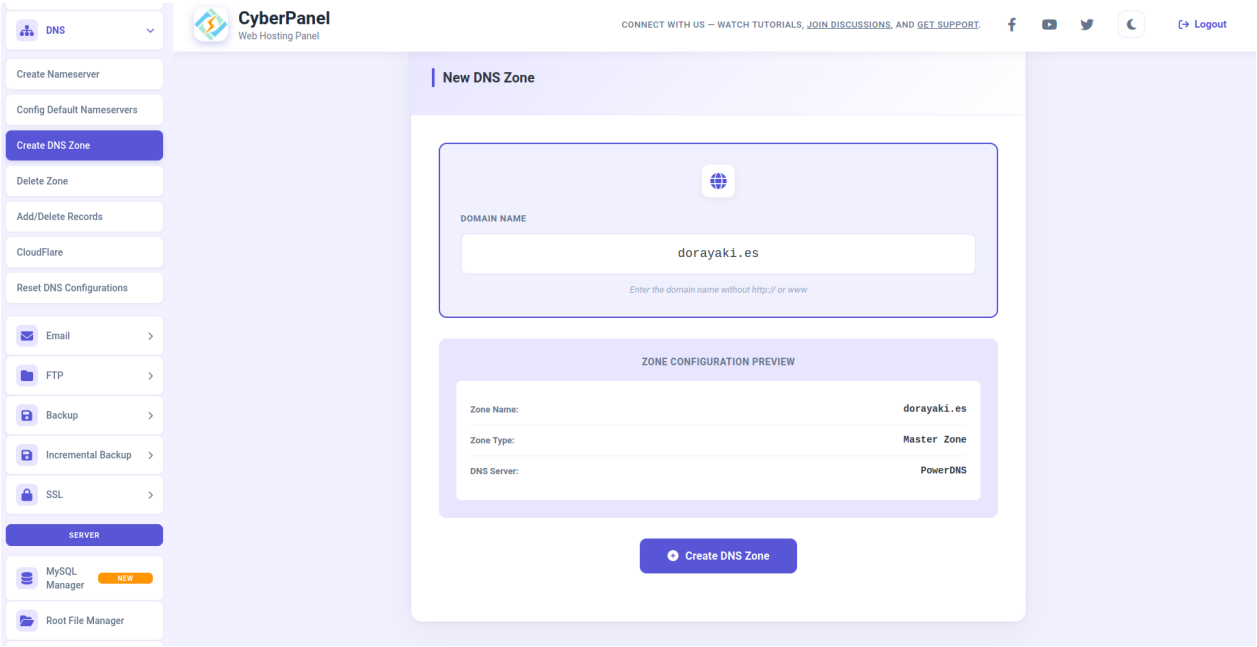


Figura 22: Creación de la zona DNS en CyberPanel

Una vez creada la zona, solo debemos de añadir registros en la sección **Add/Delete records** tal y como podemos ver aquí:

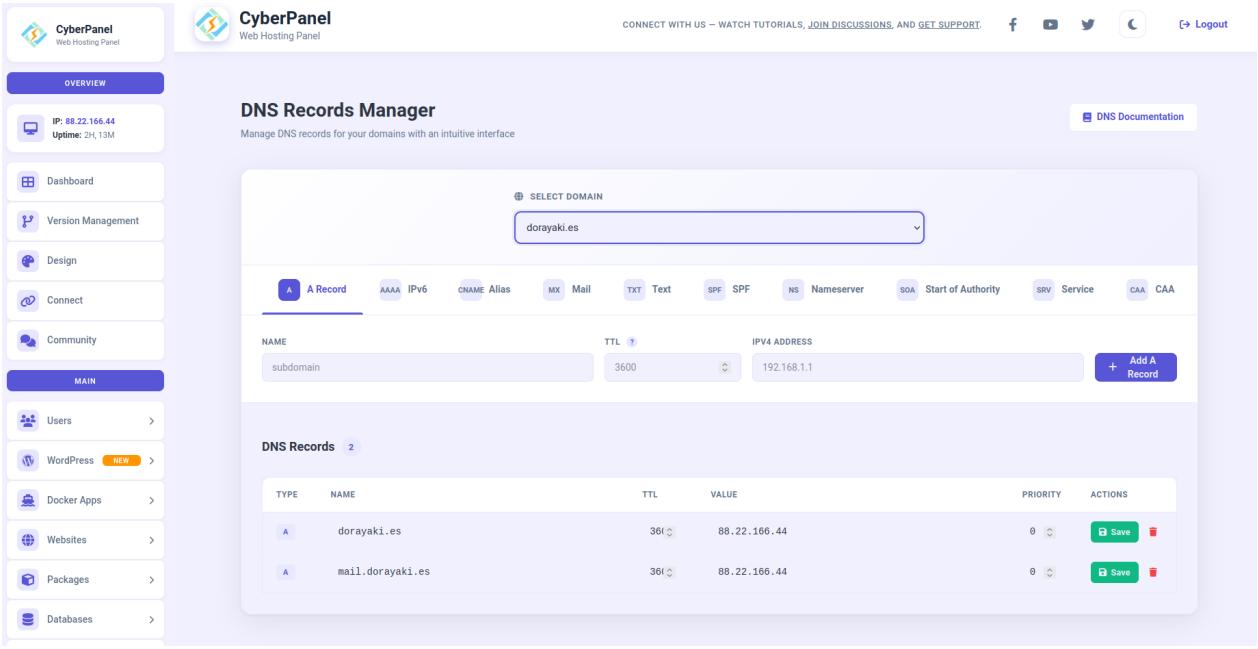


Figura 23: Registros DNS en CyberPanel

Con esto, ya tendríamos un servidor DNS público completamente funcional.

4.6. Servicio Web

Vamos a implementar un servidor web que se pueda acceder desde fuera de la sede. Para ello, usaremos CyberPanel que ya está instalado en el servidor.

En primer lugar,debemos ir a la sección de **Websites** y darle a **Create Website**. Una vez dentro, rellenamos los campos que nos piden y le damos a crear[5]:

[5]Es recomendable darle a **Create Mail Domain** pues más adelante se usará para usar servidor de correo

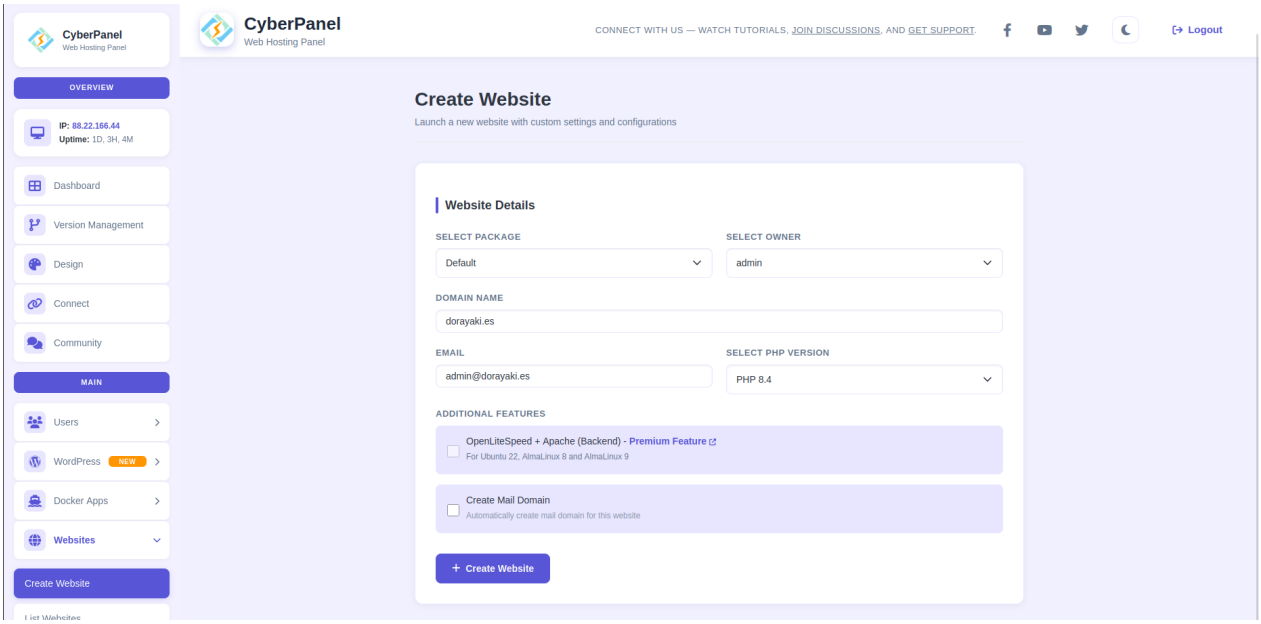


Figura 24: Creación de una página web para Dorayaki

Ahora podemos ir a la sección **List Websites** y ver que se ha creado correctamente:

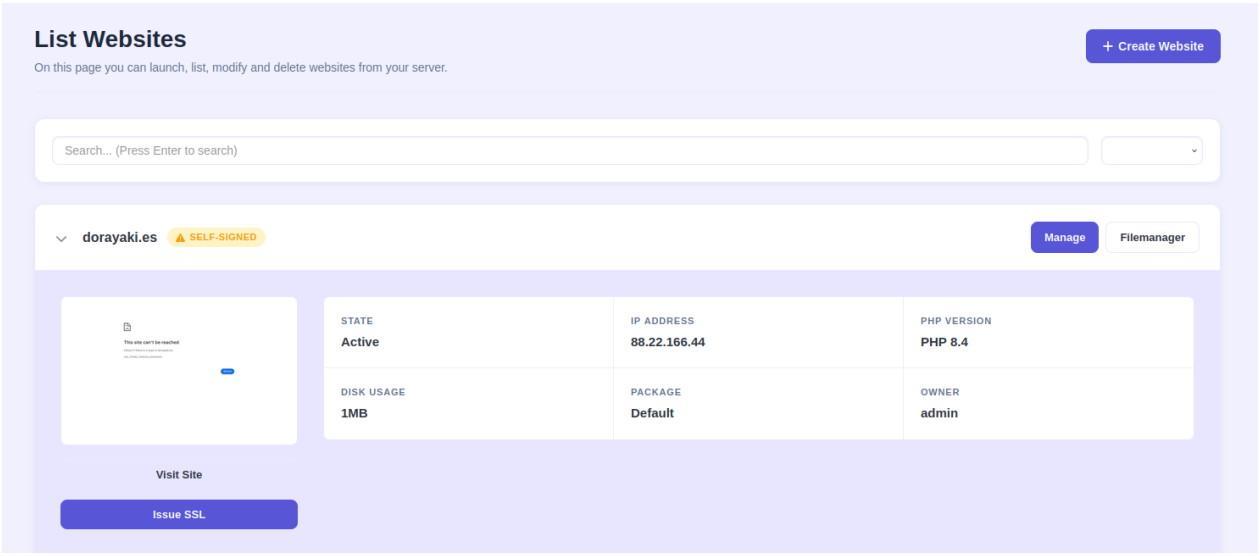


Figura 25: Verificación de la creación de la página web para Dorayaki

Si ahora le damos a **File manager** podremos acceder a un panel donde se puede modificar la página web que se va a servir:

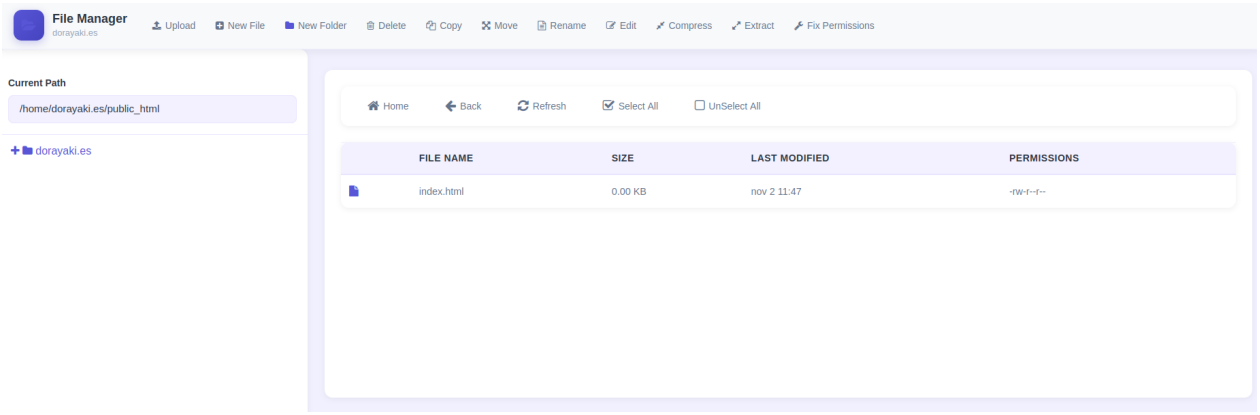


Figura 26: Modificación de la página web para Dorayaki

Una vez modificada, podemos acceder a la web con `https://dorayaki.es` y veremos la página servida:

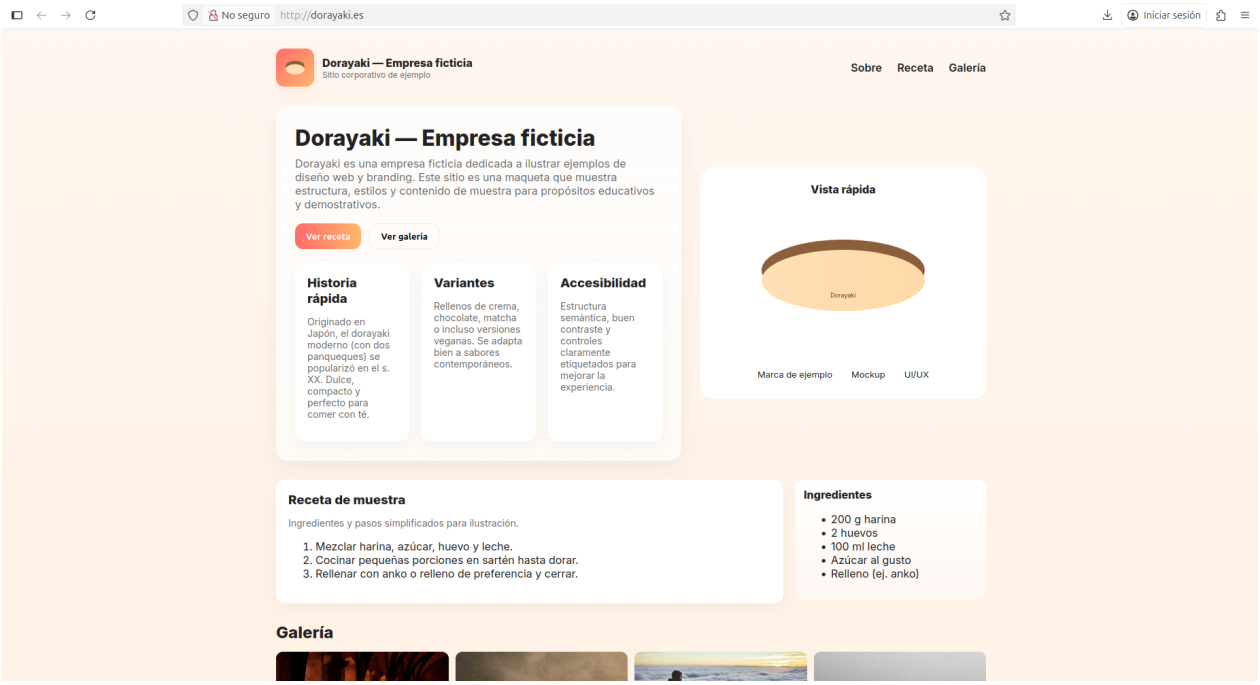


Figura 27: Página web para Dorayaki

4.7. Servicio Mail

También implementaremos un servidor de correo electrónico mediante Cyberpanel.

Como el dominio de correo ya está creado, pues lo creamos al crear la web, solo tenemos que dirigirnos a la sección de **Create Email account** para crear cuentas de correo:

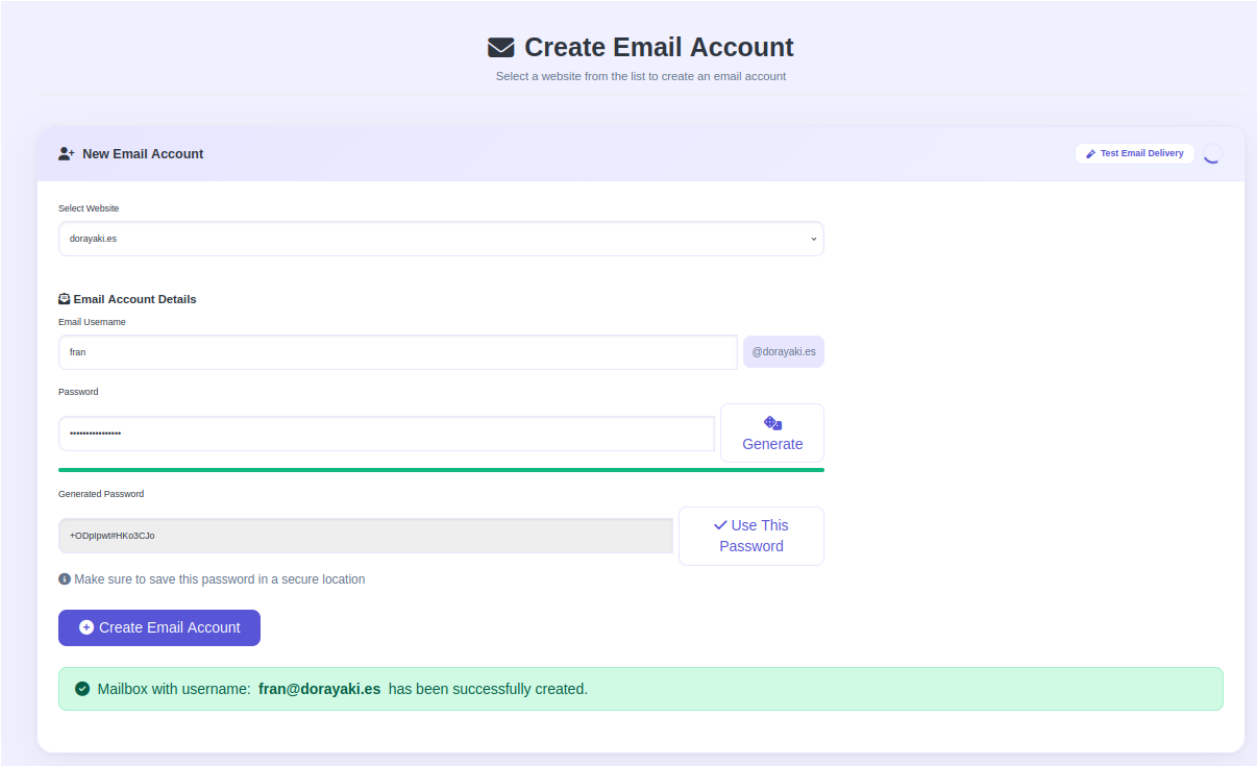


Figura 28: Creación de un email para Dorayaki

Con esto ya se podría acceder a webmail con la cuenta o incluso conectar clientes de correo como Thunderbird:

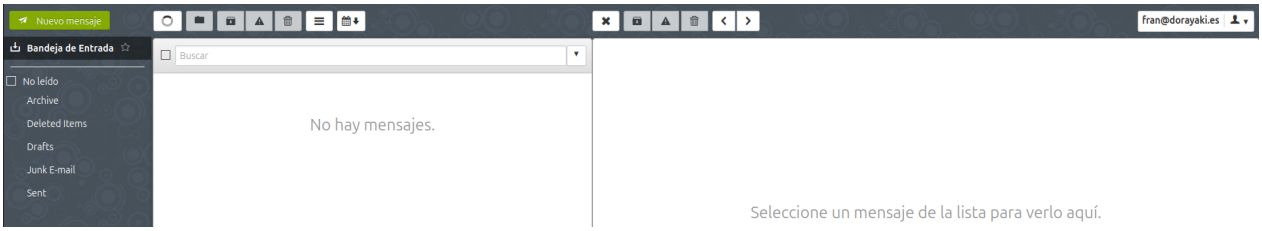


Figura 29: Acceso al webmail de Dorayaki

4.8. Servicio FTP

En este caso vamos a crear un servicio de FTP que usaremos dentro de la sede, de nuevo usando CyberPanel.

Lo único que habría que hacer es crear una cuenta en la sección **Create FTP Account** como se muesta a continuación:

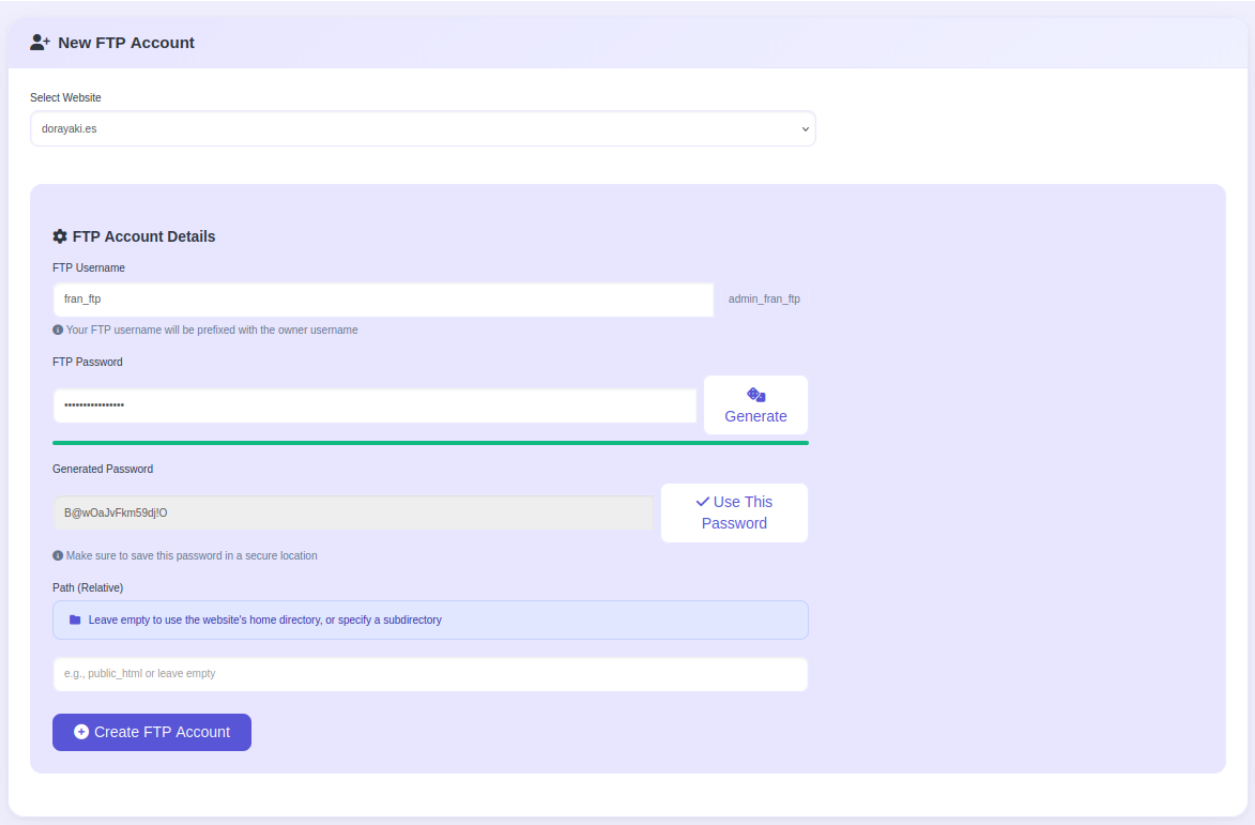


Figura 30: Creación de una cuenta de FTP para Dorayaki

Una vez creada solo nos faltaría probarla desde una terminal:

```
➜  ~ ftp admin_fran_ftp@10.10.1.2
Connected to 10.10.1.2.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
220-You are user number 1 of 50 allowed.
220-Local time is now 19:31. Server port: 21.
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
331 User admin_fran_ftp OK. Password required
Password:
230 OK. Current restricted directory is /
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Figura 31: Acceso al servidor FTP de Dorayaki

4.9. Servicio de directorio LDAP

Poder tener un directorio donde almacenemos toda la información de los usuarios y servidores de la empresa es una gran ventaja. Esto nos permite centralizar y facilitar el acceso, autenticar a los usuarios de forma sencilla y autorizar su acceso a ciertos recursos. Por esto mismo, hemos decidido añadir un servicio de directorio LDAP. Este se encontrará ubicado en el Servidor1 dentro de red privada. Esto es una cuestión de privacidad

pues un LDAP contiene información sensible y no debería ser accesible desde fuera. Vamos a ver cómo se procede con la instalación y la configuración del directorio[6].

Hemos decidido utilizar OpenLDAP como implementación. Su instalación se reduce a ejecutar los siguientes comandos:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo apt install slapd ldap-utils
```

Una vez completada la instalación debemos de ejecutar:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo dpkg-reconfigure slapd
```

Con esto ya tendríamos creado el servicio de directorio. Sin embargo, aún faltaría crear la jerarquía que queremos, ajustar los permisos y habilitar el uso de LDAPS.

4.9.1. Habilitar LDAPS

Lo más recomendable es que las peticiones utilicen TLS, pues contienen contraseñas en claro. Para esto mismo existe LDAPS que es el protocolo LDAP securizado con TLS. Para habilitarlo, primero debemos indicarle a OpenLDAP que tiene que exponer la URI de LDAPS mediante la modificación del fichero `/etc/default/slapd`. Aquí hay que buscar el atributo `SLAPD_SERVICES` y modificarlo para añadir `ldaps:///`:

```
# slapd normally serves ldap only on all TCP-ports 389. slapd can also
# service requests on TCP-port 636 (ldaps) and requests via unix
# sockets.
# Example usage:
# SLAPD_SERVICES="ldap://127.0.0.1:389/ ldaps:/// ldapi:///"
SLAPD_SERVICES="ldap:/// ldapi:/// ldaps:///"
```

Figura 32: Habilitar la URI de LDAPS

A continuación, hay que configurar cuál es la información criptográfica que se va a utilizar por parte del servidor. Para ello, necesitamos tener certificados X509 generados por una CA de confianza. En nuestro caso, usaremos la CA generada al crear el servicio de OpenVPN (sección 6.1). Cuando tengamos el certificado, la clave privada y el certificado de la CA, los colocamos en el sistema asegurándonos de que el grupo `openldap` tiene acceso de lectura. Una vez que tenemos todo esto, debemos insertar la ubicación de los datos criptográficos en la configuración de LDAP. Esto lo haremos mediante un fichero ldif como el que sigue:

```
dn: cn=config
changetype: modify
replace: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/certs/ldap.dorayaki.es.crt
-
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/certs/ldap.dorayaki.es.key
-
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ssl/certs/doraemon_ca.crt
```

Figura 33: Fichero ldif para configuración de LDAPS

Ejecutamos el siguiente comando para que la información se aplique:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo ldapmodify -Y EXTERNAL -H ldapi:/// -f certs.ldif
```

[6]Toda la información de la instalación y configuración ha sido extraída principalmente de los boletines de Servicios Telemáticos Avanzados de la Mención de Redes

Con esto ya solo necesitamos garantizar que el puerto de LDAPS está abierto para que se utilice el mismo.

4.9.2. Jerarquía y usuarios

Nuestro diretorio debe de tener una jerarquía y unos usuarios dentro. Aunque se podría dedicar todo un documento a explorar cómo estructurar correctamente, ese no es el propósito de esta asignatura. Por lo tanto, vamos a montar una jerarquía simple como la que sigue:

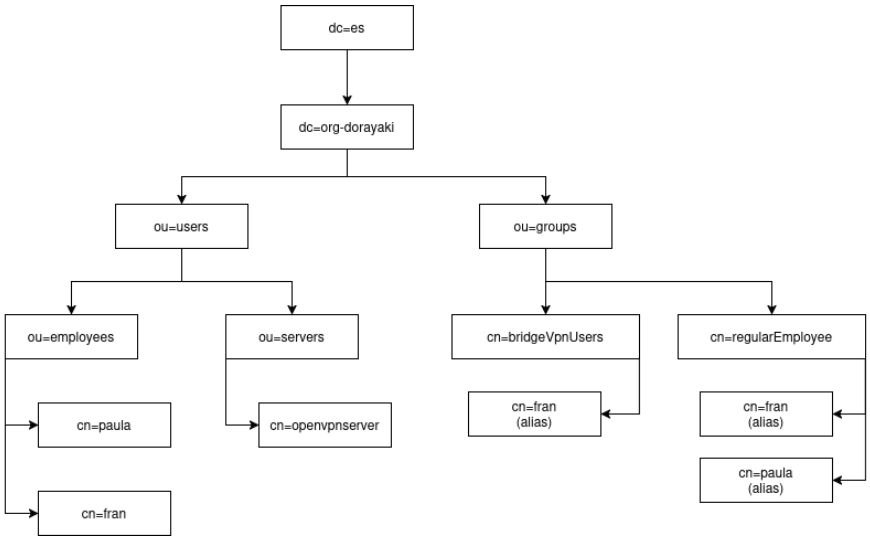


Figura 34: Jerarquía y usuarios del directorio LDAP

4.9.3. Gestión de permisos

Por último, vamos a ver una configuración muy básica de permisos. La idea es simple: un empleado normal solo debería acceder a la información de contacto de los otros empleados y a ninguna información más del directorio. Además, el usuario relativo al servidor OpenVPN solo necesita leer los datos del grupo bridgeVpnUsers. Para garantizar ambas cosas crearemos las siguietnes reglas:

```
dn: olcDatabase={1}mdb,cn=config
changetype: modify
replace: olcAccess
olcAccess: {0}to attrs=userPassword by self write by anonymous auth by * none
olcAccess: {1}to attrs=shadowLastChange by self write by * read
olcAccess: {2}to dn="cn=bridgeVpnUsers,ou=groups,dc=org-dorayaki,dc=es" by dn="cn=openvpnserver,ou=servers,ou=users,dc=org-dorayaki,dc=es" read by * none
olcAccess: {4}to attrs=cn,sn,mobile,mail by group.exact="cn=regularEmployees,ou=groups,dc=org-dorayaki,dc=es" read
olcAccess: {5}to dn.subtree="ou=users,dc=org-dorayaki,dc=es" by self write by * read
olcAccess: {6}to * by * none
```

Figura 35: Reglas ACLS del directorio LDAP

Estas garantizan (en ese orden):

1. Solo un usuario puede modificar su propia contraseña.
2. Solo un usuario puede modificar su propia fecha de cambio de contraseña.
3. El usuario `openvpnserver` es el único que puede ver los usuarios con acceso a la VPN.
4. Los empleados regulares (y solo ellos) solo pueden leer los datos de contacto del resto.

5. Un usuario solo puede modificar sus propios datos.

6. El resto de usuarios del sistema (excluyendo al admin) no pueden hacer nada.

Con esta configuración básica aseguramos una ligera protección de seguridad y evitamos posibles filtraciones de datos.

4.10. Servicio de Overleaf

Vamos a implementar una plataforma self-hosted de Overleaf privado con el objetivo de que la empresa pueda prescindir de suscripciones de pago y seguir teniendo un gran editor de LaTex.

La idea es utilizar el docker-compose proporcionado por el propio Overleaf en su versión Overleaf CE y realizar ciertas modificaciones para que funcione más adecuadamente.

Hemos decidido implementarlo dentro del Servidor3, pues este tiene una ligera ventaja con respecto al resto. Gracias a que el Servidor3 ya está alojando Nagios, este, por defecto, instala un servidor de Apache para servir el dashboard. Por lo tanto, podemos hacer uso de ese mismo servidor Apache para servir nuestra página de Overleaf sin necesidad de alguna configuración extra.

4.10.1. Instalación del servicio

Lo primero que debemos hacer es crear una carpeta en el directorio raíz llamada `overleaf` que es donde almacenaremos toda la información relativa al servicio. Ahí ubicaremos el siguiente archivo `mongodb-init-replica-set.js` provisto por el repositorio oficial de Overleaf CE para la inicialización de la base de datos. Además, también colocaremos el siguiente docker-compose:

```
services:
    sharelatex:
        restart: always
        image: franucles/sharelatex:minted
        container_name: sharelatex
        depends_on:
            mongo:
                condition: service_healthy
            redis:
                condition: service_started
        stop_grace_period: 60s
        ports:
          - 127.0.0.1:8080:80
        volumes:
            - /overleaf/sharelatex_data:/var/lib/overleaf
        environment:
            OVERLEAF_APP_NAME: Overleaf Community Edition
            OVERLEAF_MONGO_URL: mongodb://mongo/sharelatex
            TEXMFVAR: /var/lib/overleaf/data/texmf-var
            OVERLEAF_REDIS_HOST: redis
            REDIS_HOST: redis
            ENABLED_LINKED_FILE_TYPES: 'project_file,project_output_file'
            ENABLE_CONVERSIONS: 'true'
            EMAIL_CONFIRMATION_DISABLED: 'true'
            OVERLEAF_ADMIN_EMAIL: overleaf_admin@dorayaki.es
            DOCKER_RUNNER: 'false'
            SANDBOXED_COMPILES_SIBLING_CONTAINERS: 'false'
    mongo:
        restart: always
        image: mongo:6.0
```

```
    container_name: mongo
    command: '--replSet overleaf'
    volumes:
        - /overleaf/mongo_data:/data/db
        - /overleaf/mongodb-init-replica-set.js:/docker-entrypoint-initdb.d/
                                    mongodb-init-replica-set.js
    environment:
      MONGO_INITDB_DATABASE: sharelatex
    extra_hosts:
      - mongo:127.0.0.1
    healthcheck:
        test: echo 'db.stats().ok' | mongosh localhost:27017/test --quiet
        interval: 10s
        timeout: 10s
        retries: 5
redis:
    restart: always
    image: redis:6.2
    container_name: redis
    volumes:
        - /overleaf/redis_data:/data
```

Aunque hay muchas variables de entorno que tienen valores ya establecidos y recomendados por la documentación, conviene hacer unos pequeños comentarios sobre la estructura.

El servicio está dividido en 3 contenedores: sharelatex, mongo y redis. El primero es el encargado de gestionar todo lo relativo a la Overleaf, como la página web, las compilaciones, el almacenamiento de las imágenes de los proyectos, etc. El segundo es utilizado por sharelatex para almacenar metadata y los .tex de todos los proyectos. Además, también guarda toda la información relativa a los usuarios, sus proyectos asociados, el historial y otros datos de interés. Finalmente, tenemos el tercero que utiliza redis como caché para optimizar accesos y mejorar el rendimiento.
A continuación, se muestra un esquema del servicio completo:

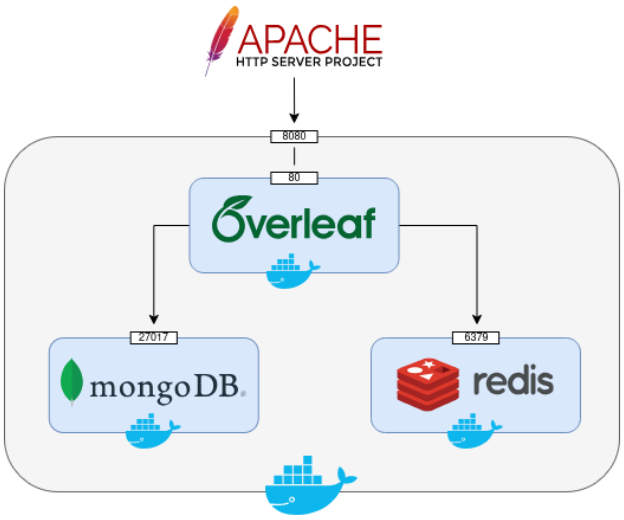


Figura 36: Esquema del servicio de Overleaf

Podemos ver que las imágenes que usan mongo y redis son las oficiales, pero la de sharelatex es personalizada. Esto se debe a que la oficial no viene con todos los paquetes instalados ni configurados para funcionar con minted. Hemos hecho las modificaciones pertinentes[7] y se ha subido a DockerHub para estar disponible.

Para terminar con el docker-compose, podemos apreciar cómo se han montado volúmenes en las carpetas que guardan la información. No se han usado volúmenes gestionados por Docker, pues tenerlos accesibles dentro del sistema de fichero de forma sencilla permitiría implementar un sistema de copias de seguridad rápidamente.

Antes de terminar con el despliegue, debemos configurar el servidor Apache para que funcione. Para ello colocamos el siguiente contenido dentro del fichero `/etc/apache2/sites-available/overleaf.conf`:

```
<VirtualHost *:80>
    ServerName overleaf.dorayaki.es
    LimitRequestFieldSize 65536
    LimitRequestBody 209715200
    RequestHeader set X-Forwarded-Proto "http"
    ProxyPreserveHost On
    ProxyRequests Off
    <Location />
        ProxyPass http://127.0.0.1:8080/
        ProxyPassReverse http://127.0.0.1:8080/
        RequestHeader set X-Real-IP %{REMOTE_ADDR}s
        RequestHeader set X-Forwarded-For %{X-Forwarded-For}e
    </Location>
    ProxyPass "/socket.io/" "ws://sharelatex/socket.io/"
    ProxyPassReverse "/socket.io/" "ws://sharelatex/socket.io/"
    ErrorLog ${APACHE_LOG_DIR}/overleaf_error.log
    CustomLog ${APACHE_LOG_DIR}/overleaf_access.log combined
</VirtualHost>
```

Ahora solo debemos habilitar los módulos específicos para esta configuración, habilitar el sitio y reiniciar el servicio:

```
server3_dorayaki @ server3dorayaki : ~  $ sudo a2enmod proxy proxy_http proxy_wstunnel headers
server3_dorayaki @ server3dorayaki : ~  $ sudo a2ensite overleaf.conf
server3_dorayaki @ server3dorayaki : ~  $ sudo systemctl reload apache2
```

Con todo esto ya tendríamos el servicio listo y preparado para servirse en la dirección `overleaf.dorayaki.es`.

4.10.2. Comprobación de funcionamiento

Para comprobar su funcionamiento basta con entrar a la página web y registrarse con un usuario creado previamente[8]. Si además creamos un proyecto de ejemplo y lo compilamos veremos que funciona a la perfección:

[7] Se omitirán por ser engorrosas y demasiado técnicas para el objetivo del proyecto
[8] La creación del primer usuario se omite por estar especificado en la documentación

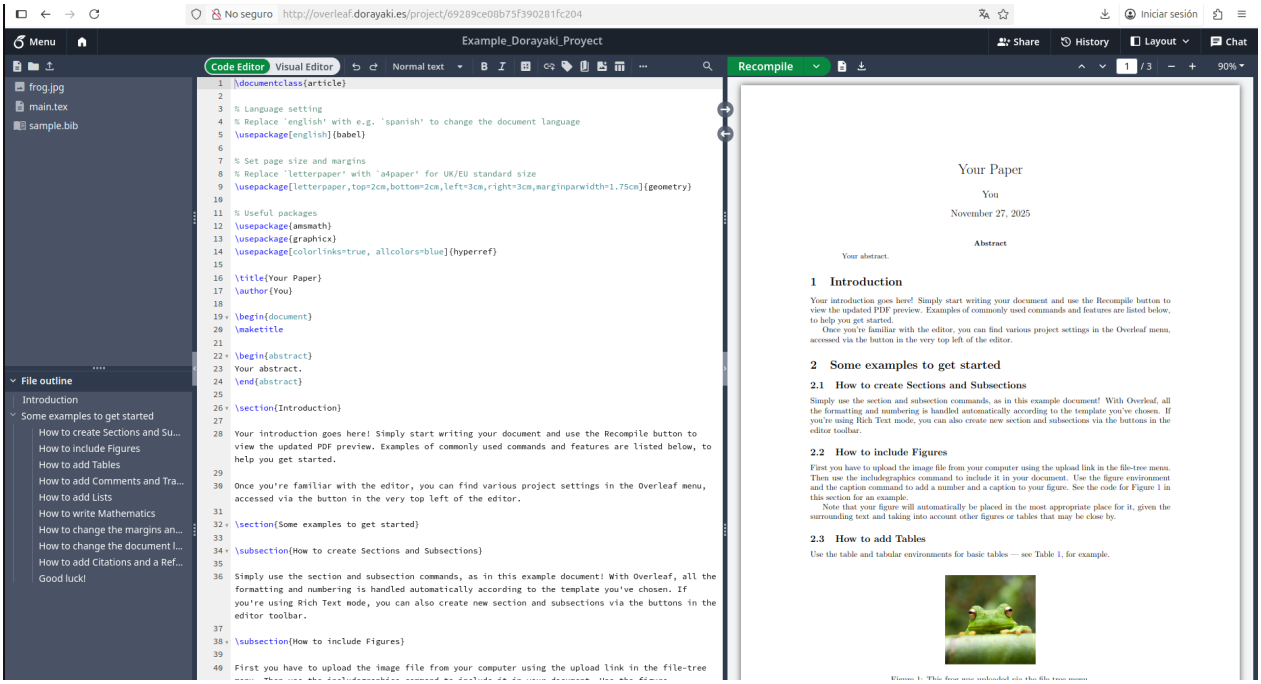


Figura 37: Página de Overleaf en funcionamiento

5. Configuración de Firewall

En esta sección abordamos la resolución de los problemas de seguridad detectados en la configuración inicial de la red (sección 3.5). El objetivo será, mediante iptables, configurar políticas de filtrado que restrinjan las conexiones entre subredes, limiten el tráfico saliente y bloqueen accesos no autorizados, garantizando un funcionamiento seguro y controlado de la red.

5.1. Comprobación e interpretación de las reglas de filtrado

Los dos primeros ejercicios del boletín correspondiente a esta sección consistían en interpretar las reglas de filtrado activas en las distintas máquinas de la topología y, posteriormente, vaciar aquellas que no fueran necesarias manteniendo las reglas de NAT.

En nuestro caso, tras las configuraciones realizadas en las secciones anteriores del proyecto, las únicas reglas existentes en las tablas de iptables corresponden precisamente a las reglas de traducción de direcciones (NAT) necesarias para el funcionamiento del enrutamiento entre redes y el acceso a Internet.

Por tanto, al listar las reglas mediante:

```
user @ hostname : ~  $ sudo iptables -L -v
user @ hostname : ~  $ sudo iptables -t nat -L -v
```

se comprobó que no existen reglas de filtrado adicionales en las cadenas INPUT, OUTPUT o FORWARD, manteniéndose únicamente las relativas a la tabla NAT.

En consecuencia, no fue necesario eliminar ninguna regla antes de proceder con el resto de ejercicios, ya que la configuración actual de los cortafuegos parte de un estado limpio y funcional, conservando únicamente las reglas de NAT establecidas para cada sede.

5.2. Política de las cadenas

Tenemos que establecer ciertas políticas para las cadenas INPUT, OUTPUT y FORWARD en cada uno de los dispositivos. De esta manera, si un paquete no coincide con ninguna de las reglas de firewall definidas, se le aplicará esta acción por defecto.

	INPUT	**FORWARD**	**OUTPUT**
Routers	DROP	ACCEPT	ACCEPT
Servidores	DROP	DROP	ACCEPT
Hosts	ACCEPT	DROP	DROP

Tabla 1: Políticas de las cadenas según el dispositivo

5.3. Configuración de los routers

Vamos a ver las diferentes configuraciones de firewall que hay que realizar dentro de los servidores. En primer lugar, necesitamos que las reglas se mantengan entre reinicios. Por lo tanto, vamos a utilizar el paquete `iptables-persistent` que nos permite mantener las reglas mediante el comando `netfilter-persistent`.

Lo primero que haremos será impedir que nuestro router aparezca en un `traceroute`. Para ello, debemos descartar todos los paquetes con TTL exceeded y lo haremos con la regla:

```
router @ organizacion : ~ $ sudo iptables -A OUTPUT -p ICMP –icmp-type time-exceeded -j DROP
```

Ahora vamos a asegurar que el router solo sea accesible mediante SSH desde una IP concreta de la subred. Esto lo realizaremos de la siguiente forma:

```
# Router sede Dorayaki solo accesible desde server1
router_dorayaki @ routerdorayaki : ~ $ sudo iptables -A INPUT -s 10.10.0.2 -p TCP –dport ssh -m state –state NEW -j ACCEPT
router_dorayaki @ routerdorayaki : ~ $ sudo iptables -A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
# Router sede no confiable solo accesible desde el host
router_nc @ routernc : ~ $ sudo iptables -A INPUT -s 10.10.1.2 -p TCP –dport ssh -m state –state NEW -j ACCEPT
router_nc @ routernc : ~ $ sudo iptables -A INPUT -m state –state ESTABLISHED,RELATED -j ACCEPT
```

Otra regla que deberá estar siempre presente en ambos routers es permitir las respuestas entrantes de las peticiones DNS que al ser mediante UDP no se ven afectadas por la reglas stateful que hemos puesto anteriormente:

```
router_dorayaki @ routerdorayaki : ~ $ sudo iptables -A INPUT -p UDP –sport domain -j ACCEPT
router_nc @ routernc : ~ $ sudo iptables -A INPUT -p UDP –sport domain -j ACCEPT
```

En el caso del router "No confiableza habríamos terminado, pero en el caso del router de la sede Dorayaki aún no estaría.

Como vamos a monitorizar la actividad del router desde el Servidor3, necesitamos habilitar tanto SNMP como ICMP para el mismo. Para ello añadimos las siguientes reglas:

```
router_dorayaki @ routerdorayaki : ~ $ sudo iptables -A INPUT -p UDP -s 10.10.0.3 –dport snmp -j ACCEPT
router_dorayaki @ routerdorayaki : ~ $ sudo iptables -A INPUT -p ICMP -s 10.10.0.3 -j ACCEPT
```

Además, como tenemos una VPN corriendo en el router en el puerto 1995 debemos de habilitarlo:

```
router_dorayaki @ routerdorayaki : ~  $ sudo iptables -A INPUT -p UDP –dport 1194 -j ACCEPT
```

Finalmente, para garantizar que no haya ninguna fuga de paquetes hacia la red privada vamos a añadir una última regla de firewall:

```
router_dorayaki @ routerdorayaki : ~  $ sudo iptables -A FORWARD ! -s 10.10.0.0/23 -d 10.10.0.0/24
-j DROP
```

5.4. Configuración de los servidores

Ahora vamos a ver las diferentes configuraciones de firewall que hay que realizar dentro de los servidores. Tal y como sucede en los routers, necesitamos que las reglas se mantengan entre reinicios. Por lo tanto, vamos a utilizar el paquete `iptables-persistent` que nos permite mantener las reglas mediante el comando `netfilter-persistent`.

5.4.1. Restricción de conexiones ssh entre subredes distintas

Uno de los principales problemas de seguridad detectados en la configuración inicial era la posibilidad de acceder mediante SSH desde la sede "No confiable.ª los dispositivos de la red privada de Dorayaki. Para solucionar esta vulnerabilidad se configuraron reglas de firewall que limitan el acceso SSH únicamente a los equipos pertenecientes a la misma subred, rechazando cualquier intento de conexión proveniente de redes externas.

Inicialmente, podría plantearse una configuración *stateless* en la que se aceptaran todas las conexiones SSH desde la subred local y se descartaran las restantes. Sin embargo, esta aproximación permitiría que cualquier equipo interno realizara mapeos de puertos hacia el servidor y no distinguiría entre paquetes nuevos y los pertenecientes a una sesión ya abierta.

Para evitar este problema, se optó por una configuración *stateful* mediante el uso del módulo state de iptables, que permite controlar el tráfico en función del estado de la conexión. De este modo, sólo se aceptan nuevas conexiones SSH originadas desde la subred local y los paquetes asociados a sesiones ya establecidas o relacionadas, garantizando la persistencia de la conexión y bloqueando intentos no autorizados.

Las reglas básicas aplicadas a todos los servidores de la red privada de la sede Dorayaki fueron las siguientes[9]:

```
# Aceptar las solicitudes de nuevas conexiones ssh solo desde la subred local
server @ dorayaki : ~  $ sudo iptables -A INPUT -s 10.10.0.0/24 -p TCP –dport ssh -m state –state
NEW -j ACCEPT

# Permitir solo paquetes que estén relacionados con una conexión ya existente
server @ dorayaki : ~  $ sudo iptables -A INPUT -m state –state ESTABLISHED,RELATED -j
ACCEPT
```

Con esta configuración, los servidores aceptan únicamente las conexiones SSH originadas dentro de su propia subred (10.10.0.0/24). Esto se debe a que cuando aparece conexión externa, esta no coincide con ninguna de las reglas anteriores y se le aplica la política por defecto que es DROP. Tras su aplicación, se verificó que las conexiones internas entre los equipos de Dorayaki se mantenían estables, mientras que los intentos de acceso SSH desde la red "No confiable.eran correctamente bloqueados.

Aunque las reglas anteriores son comunes, cada servidor de Dorayaki requiere configuraciones adicionales según los servicios que presta.

[9]Unos comandos análogos se ejecutan con los de la red pública solo que cambiando la red

5.4.2. Servidor 1

La primera regla que tenemos que añadir y que estará en todos los servidores será la de permitir el tráfico loopback:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -i lo -j ACCEPT
```

Otra regla que deberá estar siempre presente es permitir las respuestas entrantes de las peticiones DNS que al ser mediante UDP no se ven affectadas por la reglas stateful que hemos puesto anteriormente:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -p UDP –sport domain -j
ACCEPT
```

El Servidor1 ofrece el servicio DHCP, por lo que debe permitir tráfico UDP entre los clientes y el servidor en los puertos 67 y 68.
Si únicamente se aplicaran las reglas SSH comunes, los clientes de la red no podrían obtener configuración IP.

Para mantener la funcionalidad DHCP, se añadieron las siguientes reglas:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -s 10.10.0.0/24 -p UDP –sport
68 –dport 67 -j ACCEPT
```

Por otro lado, en el Servidor1 también se aloja el directorio LDAP. Por lo tanto, debemos de permitir que se acceda desde fuera mediante LDAPS y para ello debemos añadir las siguientes reglas:

```
# LDAPS para consultas seguras externas
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -s 10.10.0.0/23 -p TCP –dport
ldaps -m state –state NEW -j ACCEPT
```

Para permitir la monitorización tenemos que aceptar los SNMP e ICMP que provengan del Servidor3:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -p UDP -s 10.10.0.3 –dport
snmp -j ACCEPT
server1_dorayaki @ server1dorayaki : ~ $ sudo iptables -A INPUT -p ICMP -s 10.10.0.3 -j ACCEPT
```

Como el Servidor1 alojará el DNS Autoritativo, debemos de permitir las peticiones de DNS entrantes:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -p UDP -s 10.10.0.0/24 –dport
domain -j ACCEPT
```

Como veremos más adelante, tenemos que permitir las conexiones a nuestra base de datos. Como actualmente solo se debe permitir el acceso desde el servidor en la red pública de la sede Dorayaki, solo vamos a restringir el acceso a este servidor. Sin embargo, si tuvieramos varios, sería ideal permitirlo a toda la red o a un conjunto de IPs:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -p TCP -s 10.10.1.2 –dport
mysql -j ACCEPT
```

Finalmente, aunque se comentará más adelante, también conviene tener las reglas que se usarán para K8s considerando que este servidor es un nodo worker. Estas serían las siguientes:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -p TCP –dport 10250 -m comment –comment 'K8s API' -j ACCEPT
server1_dorayaki @ server1dorayaki : ~  $ sudo iptables -A INPUT -p TCP –dport bgp -m comment –comment 'Calico BGP' -j ACCEPT
```

No se requiere añadir reglas de salida (OUTPUT), ya que la política por defecto es ACCEPT.

5.4.3. Servidor 2

Como se indicó anteriormente, la primera regla será la de permitir el tráfico loopback:

```
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -i lo -j ACCEPT
```

Del mismo modo, debemos permitir las respuestas entrantes de las peticiones DNS que al ser mediante UDP no se ven affectadas por la reglas stateful que hemos puesto anteriormente:

```
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -p UDP –sport domain -j ACCEPT
```

Se debe de incluir la regla para aceptar SNMP e ICMP entrante para la monitorización:

```
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -p UDP -s 10.10.0.3 –dport snmp -j ACCEPT
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -p ICMP -s 10.10.0.3 -j ACCEPT
```

Además de estas reglas, como el Servidor2 contendrá servicios públicos de SMTP, IMAP, POP3 y DNS deberemos de habilitar los respectivos puertos con las siguientes reglas:

```
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -p TCP -m multiport –dport smtp,imap,pop3 -j ACCEPT
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -p UDP –dport domain -j ACCEPT
```

Finalmente, como el Servidor2 también tiene un servicio de FTP privado de la empresa, deberemos de permitir los accesos al puerto de FTP a desde la red privada

```
server2_dorayaki @ server2dorayaki : ~  $ sudo iptables -A INPUT -p TCP -s 10.10.0.0/24 -m multiport –dport ftp,ftp-data -j ACCEPT
```

5.4.4. Servidor 3

Como se indicó anteriormente, la primera regla será la de permitir el tráfico loopback:

```
server3_dorayaki @ server3dorayaki : ~  $ sudo iptables -A INPUT -i lo -j ACCEPT
```

Del mismo modo, debemos permitir las respuestas entrantes de las peticiones DNS que al ser mediante UDP no se ven affectadas por la reglas stateful que hemos puesto anteriormente:

```
server3_dorayaki @ server3dorayaki : ~  $ sudo iptables -A INPUT -p UDP –sport domain -j ACCEPT
```

Por otro lado, el Servidor3 aloja los servicios de monitorización de red (Ntop y Nagios), por lo que requiere una configuración más abierta dentro del entorno interno. Además de las reglas SSH comunes, fue necesario permitir tráfico adicional para los puertos específicos de cada servicio, así como para ICMP y SNMP, empleados en la monitorización.

Se añadieron las siguientes reglas adicionales:

```
# HTTP/HTTPS para interfaces web
server3_dorayaki @ server3dorayaki : ~  $ sudo iptables -A INPUT -s 10.10.0.0/24 -p TCP –dport
http:https -m state –state NEW -j ACCEPT

# ICMP para monitorización
server3_dorayaki @ server3dorayaki : ~  $ sudo iptables -A INPUT -p ICMP -s 10.10.0.0/23 -j
ACCEPT
server3_dorayaki @ server3dorayaki : ~ $ sudo iptables -A INPUT -p ICMP -s localhost -j ACCEPT
# Puerto que usará nProbe para monitorización
server3_dorayaki @ server3dorayaki : ~ $ sudo iptables -A INPUT -p TCP –dport 5556 -m comment
–comment 'nProbe inputs' -j ACCEPT
# Puerto NFS
server3_dorayaki @ server3dorayaki : ~ $ sudo iptables -A INPUT -P TCP –dport 2049 -m comment
–comment 'NFS' -j ACCEPT
```

Podemos apreciar como las comunicaciones de las interfaces web solo tienen reglas para las nuevas comunicaciones. Esto se debe a que gracias a la restricciones en las conexiones ssh (sección 5.4.1) ya tenemos una regla que permite la entrada de todos los paquetes relacionados con conexiones existentes.

Finalmente, aunque se comentará más adelante, también conviene tener las reglas que se usarán para K8s considerando que este servidor es un nodo worker. Estas serían las siguientes:

```
server3_dorayaki @ server3dorayaki : ~  $ sudo iptables -A INPUT -p TCP –dport 10250 -m com-
ment –comment 'K8s API' -j ACCEPT
server3_dorayaki @ server3dorayaki : ~ $ sudo iptables -A INPUT -p TCP –dport bgp -m comment
–comment 'Calico BGP' -j ACCEPT
```

5.4.5. Servidor 4

Como se indicó anteriormente, la primera regla será la de permitir el tráfico loopback:

```
server4_dorayaki @ server4dorayaki : ~  $ sudo iptables -A INPUT -i lo -j ACCEPT
```

Del mismo modo, debemos permitir las respuestas entrantes de las peticiones DNS que al ser mediante UDP no se ven affectadas por la reglas stateful que hemos puesto anteriormente:

```
server4_dorayaki @ server4dorayaki : ~  $ sudo iptables -A INPUT -p UDP –sport domain -j
ACCEPT
```

Como el Servidor4 alojará el DNS Recursor, debemos de permitir las peticiones de DNS entrantes:

```
server4_dorayaki @ server4dorayaki : ~  $ sudo iptables -A INPUT -p UDP -s 10.10.0.0/24 –dport
domain -j ACCEPT
```

Finalmente, aunque se comentará más adelante, también conviene tener las reglas que se usarán para K8s considerando que este servidor es un nodo maestro. Estas serían las siguientes:

```
server4_dorayaki @ server4dorayaki : ~ $ sudo iptables -A INPUT -p TCP –dport 6443 -m comment
–comment 'K8s API Server' -j ACCEPT
server4_dorayaki @ server4dorayaki : ~ $ sudo iptables -A INPUT -p TCP –dport bgp -m comment
–comment 'Calico BGP' -j ACCEPT
```

5.5. Configuración de los hosts

En los hosts necesitamos garantizar 2 cosas: se deben descartar los paquetes ICMP entrantes y se deben bloquear todas las conexiones salientes salvo HTTP/HTTPS. A nuestro entender, estas dos cosas son algo restrictivas de más puesto que si no permitimos peticiones salientes de DNS no tendremos un acceso a Internet razonable. Además, luego entrarán en juego protocolos como IMAP o POP3 que necesitarán enviar peticiones, pero por ahora ignoraremos estos últimos.

Para conseguir la funcionalidad que deseamos debemos introducir los siguientes comandos:

```
host1 @ host1dorayaki : ~ $ sudo iptables -A OUTPUT -p TCP –dport http:https -j ACCEPT
host1 @ host1dorayaki : ~ $ sudo iptables -A OUTPUT -p UDP –sport domain -j ACCEPT
host1 @ host1dorayaki : ~ $ sudo iptables -A INPUT -p icmp -j DROP
```

Con esto y las políticas por defecto ya lo tendríamos configurado para el host de la sede Dorayaki.

En cuanto al host de la sede "No confiable", este sería análogo, pero tiene que añadir una regla extra:

```
host2 @ host2nc : ~ $ sudo iptables -A OUTPUT -p UDP –dport 1194 -j ACCEPT
```

Esta le permitirá establecer tanto el túnel de red como el de enlace con el servidor.

Sin embargo, necesitamos que estas reglas se preserven cuando se reinicia el dispositivo. Para ello se va a usar el servicio `iptables.service` que se instala mediante el comando `dnf install iptables-services`. Solo necesitamos realizar `service iptables save` para guardar la configuración actual, habilitar el servicio para que arranque al inicio y ya tendríamos las reglas de forma persistente.

6. Configuración de los túneles

Nosotros hemos decidido implementar un túnel mediante OpenVPN. El motivo por el que se han descartado otras tecnologías estudiadas se puede encontrar en el Anexo B

6.1. Túnel OpenVPN

OpenVPN es una tecnología que nos permite establecer un túnel que solvente todos los problemas que presentaban tanto GRE como L2TP. Además, nos permite tanto un túnel a nivel de red (como GRE) como de enlace (L2TP). Nosotros hemos decidido implementar esta tecnología en vez de las anteriores pues consideramos que proporciona la misma funcionalidad sin las problemáticas de seguridad y manejo que ya habíamos comentado.

Para la gestión de los certificados nosotros hemos optado por la implementación de una CA externa. De esta manera podemos generar certificados tanto para clientes y servidores simulando que es una agencia externa quien lo maneja. Además, de esta manera podremos usar esta CA para futuras cuestiones como HTTPS, SMTPS o IMAPS entre otros. La creación de la CA no entraña ningún misterio y solo se debe seguir el tutorial oficial de OpenVPN.

Ahora vendría la configuración tanto del cliente como del servidor. Nosotros hemos decidido montar un tunel a nivel de enlace. De esta forma, los usuarios que dispongan de la autorización correcta podrán obtener una IP de la red privada de la empresa. Esto está enfocado a trabajadores en remoto que necesiten acceso a los

servidores privados de la empresa.
Por simplicidad, se asumirá que tanto el cliente como el servidor tienen en su poder tanto sus claves públicas/-privadas, como el secreto compartido `ta.key` (generado por la CA) y el fichero de parámetros Diffie-Hellman.

6.1.1. Túnel de enlace

Tanto en el servidor como en el cliente vamos a seguir las mismas instrucciones del tutorial antes mencionado para montar el túnel. Sin embargo, hay que comentar un par de cosas adicionales para dejarlo tal y cómo queremos.

La primera de todas tiene que ver con el modo promiscuo. Para que esto funcione correctamente hay que indicarle a VirtualBox que la interfaz que hemos conectado al puente debe de permitir el modo promiscuo. En particular, debemos de permitir que la interfaz de la red interna privada tenga el modo promiscuo habilitado:

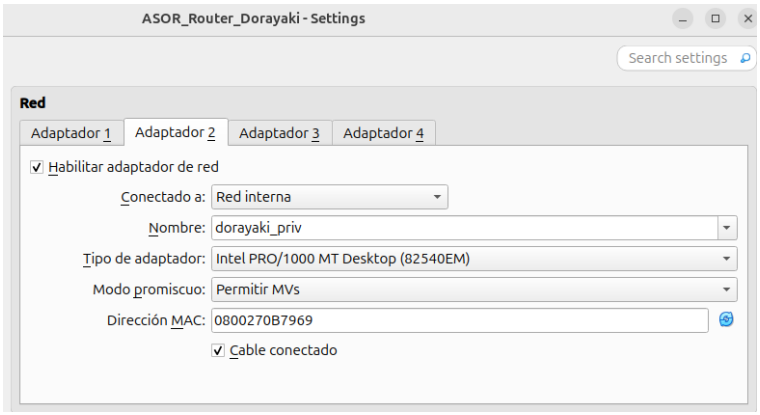


Figura 38: Modo promiscuo activado en VirtualBox

Una vez hecho esto, necesitamos crear la interfaz virtual bridge que usaremos para conectar el túnel tap. Además, como la red que vamos a puentear es la `10.10.0.0/24`, necesitamos conectar también su interfaz a ese puente. Para ello, hay que modificar de la siguiente manera el fichero de configuración de `netplan`:

```
network:
  version: 2
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: no
    enp0s9:
      dhcp4: no
      addresses:
        - 10.10.1.1/24
      nameservers:
        addresses:
          - 10.10.0.4
    enp0s10:
     dhcp4: no
     addresses:
       - 172.16.0.1/24
  bridges:
    br0:
      interfaces:
```

```
    - enp0s8
  dhcp4: no
  addresses:
    - 10.10.0.1/24
  nameservers:
    addresses:
      - 10.10.0.4
```

Además de esto, debemos de asegurar que cuando se arranca el servidor la VPN está lista para ser usada. Para ello, habría que habilitar la VPN mediante `systemctl enable openvpn@dorayaki_tap` y luego habría que garantizar que la interfaz se levanta y se conecta al puente (`br0`) que hemos creado. Para ello, hemos creado un servicio que se ejecuta justo después de `openvpn@dorayaki_tap` de forma que ejecute el siguiente script y lo monte:

```
#!/bin/bash
/sbin/ip link set tap0 up
/sbin/ip link set tap0 master br0
```

Figura 39: Script para montar y conectar el túnel de enlace

Las IPs que se le asignarán a los usuarios de la VPN serán `10.10.0.200-10.10.0.254`. Para que no haya problemas también se han eliminado estas IPs del rango asignable por el DHCP.

Como en la red privada disponemos de un servidor DNS, necesitamos informarle al cliente de que debe usar un DNS concreto y para ello añadimos en el fichero de configuración la línea `push 'dhcp-option DNS 10.10.0.2'` para indicarle que queremos que use ese servidor DNS.

Ahora llega el aspecto de la autorización. Tal y como hemos mencionado antes, debemos de restringir el uso de esta VPN a ciertos usuarios. Tal y como está ahora mismo cualquier usuario con un certificado válido podría acceder a la VPN. Para solventar esto usaremos LDAP. La idea es sencilla:

1. El servidor OpenVPN recibe una petición de conexión con el certificado X509
2. El servidor autentica que el usuario es quien dice ser y procede a establecer la conexión TLS
3. Antes de completar el establecimiento de la conexión TLS, el servidor extrae el nombre del certificado y le pregunta al servidor LDAP si tiene autorización para montar el túnel
4. Si tiene autorización, establece el túnel. En caso contrario, rechaza la conexión

Para hacer esto hemos hecho uso del grupo `bridgeVpnUsers`. Si un usuario se encuentra dentro del grupo, podrá montar un túnel.

Dentro de la configuración del servidor hemos incluido la directiva `tls-verify <ruta del script>` y de esta forma podemos indicar un script que ejecutaremos para verificar la conexión TLS. En nuestro caso, vamos a usar el manual de OpenVPN para constuir el siguiente script[10]:

[10]La contraseña está en texto plano. Sin embargo, los permisos del archivo impiden que cualquier usuario que no sea el root pueda leer el fichero

```
#!/bin/bash
CN=$(echo "$X509_0_CN")
RESULT=$(ldapsearch -D cn=openvpnserver,ou=servers,ou=users,dc=org-dorayaki,dc=es \
        -w "1234" \
        -H ldaps://ldap.dorayaki.es \
        -b cn=bridgeVpnUsers,ou=groups,dc=org-dorayaki,dc=es \
       | grep "$CN")
RESULT_CODE=$(echo $?)
exit $RESULT_CODE
```

Figura 40: Script para la verificación de los usuarios

La idea es usar el usuario `openvpnserver` para consultar si dentro del grupo `bridgeVpnUsers` existe un miembro con el nombre del certificado que nos ha llegado. En caso afirmativo, aceptamos la conexión. De lo contrario, la rechazamos.
Con todo esto, ya tendríamos en perfecto funcionamiento también la VPN de enlace.

7. Clúster de K8s

En esta sección vamos a pasar a explicar cómo se ha montado el clúster de K8s y como se ha hecho uso de él para el despliegue de un servidor web privado de la empresa. Para el montaje y la configuración del clúster, se deberá de consultar el Anexo C.

7.1. Despliegue del servidor web

La idea es montar un servidor web interno de la empresa que esté distribuido en K8s. Para ello, vamos a desplegar un servicio NGINX que tomará los ficheros mostrar desde un directorio montado con NFS.

Para comenzar, lo primero que tenemos que hacer es asegurar que todos los servidores del clúster tienen la capacidad de montar un sistema de ficheros NFS y para eso hay que instalar el paquete `nfs-common` en todos ellos. Una vez que se tiene esto listo y se tiene la IP del servidor NFS (en nuestro caso la `10.10.0.3`) ya podemos proceder con el despliegue.

La idea es sencilla es crear un Persistent Volume (PV) que indique cómo montar el directorio NFS donde se va a encontrar la página web. Además, crearemos una Persistent Volume Claim (PVC) para que los nodos sepan qué tipo de PV tienen que montar. Para configurar el servidor NGINX, vamos a utilizar un ConfigMap que contendrá el contenido relativo a toda la configuración del servidor (`nginx.conf`). También desplegaremos un Deployment con 3 réplicas que será el encargado de servir la página. Finalmente, haremos uso de un NodePort[11] para poder acceder al servicio sin ningún tipo de problema.

Todo esto lo realizaremos sobre un namespace propio para poder tener más control sobre la gestión de la infraestructura. La arquitectura final se puede consultar en el Anexo D.

7.1.1. Namespace

En primer lugar, vamos a crear el namesapace llamado **dorayaki**. Esto sería tan sencillo como aplicar el siguiente yaml:

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dorayaki
```

[11] Aunque sería más idóneo un Ingress, pero no queremos complicarlo más

7.1.2. Persistent Volume

Ahora vamos a crear el Persistent Volume que nos permita indicar las características del volumen. Este será de tipo NFS y será ReadWriteMany pues varios pods deberán poder leer y escribir en el mismo. Además, gracias a las etiquetas garantizaremos que el PVC elegirá el PV que queremos. Para crearlo, solo debemos aplicar el siguiente yaml:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-private-pv
  namespace: dorayaki
  labels:
    type: nfs
    purpose: nginx-content
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  nfs:
    server: 10.10.0.3
    path: /nginx
```

7.1.3. Persistent Volume Claim

Ahora vamos a crear el Persistent Volume Claim que usarán los Deployments para solicitar el volumen a montar. Este tendrá las mismas características y solicitrará las mismas etiquetas que tiene el PV. Para crearlo, solo debemos aplicar el siguiente yaml:

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-private-pvc
  namespace: dorayaki
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
  selector:
    matchLabels:
      type: nfs
      purpose: nginx-content
```

7.1.4. ConfigMap

Ahora vamos a crear el ConfigMap que contendrá la configuración del servidor NGINX. Esta será la estándar y consistirá solo en escuchar en el puerto 80 y servir de la carpeta `/usr/share/nginx/html`. Para crearlo, solo debemos aplicar el siguiente yaml:

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-private-configmap
  namespace: dorayaki
```

```
data:
  nginx.conf: |
    server {
      listen 80;
      server_name localhost;
      location / {
        root /usr/share/nginx/html;
      }
    }
```

7.1.5. Service

Ahora vamos a crear un Service para poder acceder al NGINX sin problema. Nosotros hemos elegido el puerto TCP **30080** que deberemos de habilitar en los firewall de todos los nodos del clúster. Para crear el service solo debemos aplicar el siguiente yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-nodeport
  namespace: dorayaki
spec:
  selector:
    app: nginx
  type: NodePort
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      nodePort: 30080
```

7.1.6. Deployment

Ahora vamos a crear el Deployment que nos controlará las instancias del NGINX. Esta nos desplegará las 3 réplicas y se encargará de montar los volúmenes tanto del PV como del ConfigMap correctamente. Hay que tener en cuenta que el contenido de la página web deberá estar dentro de la carpeta `web` del directorio exportado `nginx` del servidor NFS. El yaml quedaría como sigue:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  namespace: dorayaki
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
```

```
    volumeMounts:
    - name: web-content
      mountPath: /usr/share/nginx/html
      subPath: web
    - name: web-config
      mountPath: /etc/nginx/conf.d
  volumes:
    - name: web-content
      persistentVolumeClaim:
        claimName: nginx-private-pvc
    - name: web-config
      configMap:
        name: nginx-private-configmap
```

7.1.7. Comprobación de funcionamiento

Gracias a que tenemos el dashboard de K8s, podemos ver cómo todos los recursos que hemos creado se encuentran dentro de nuestro namespace:

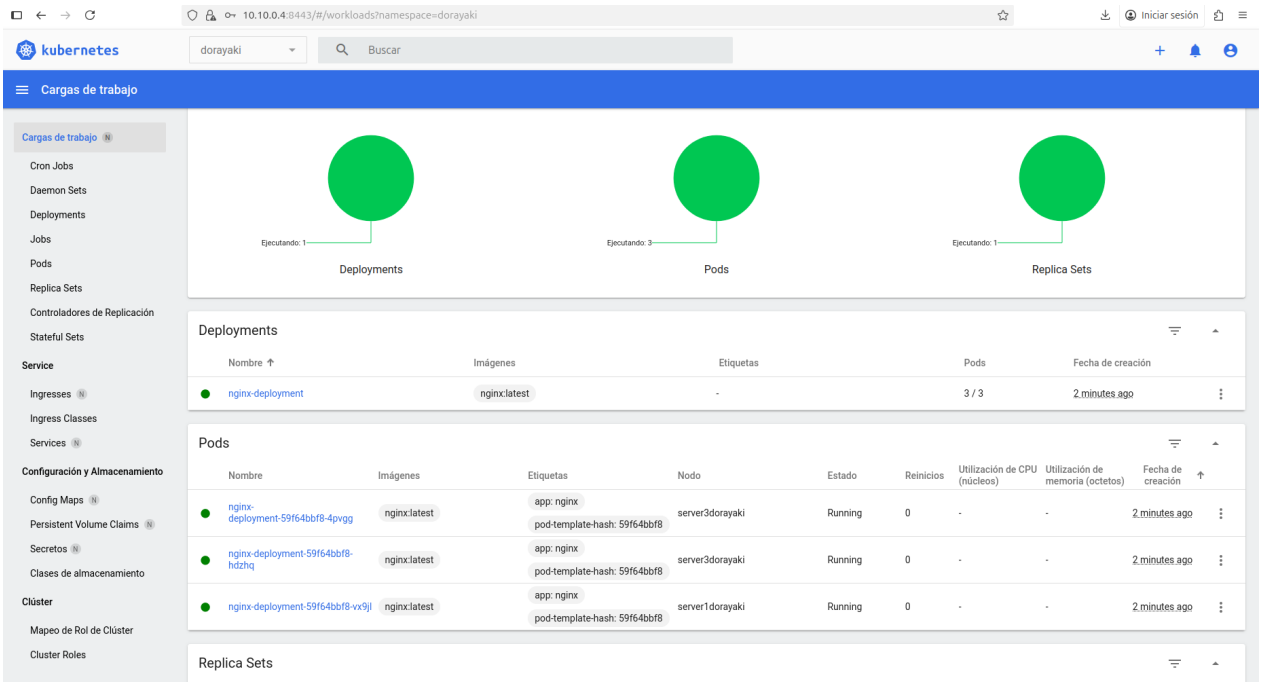


Figura 41: Dashboard de K8s en el namespace **dorayaki**

Además, podemos ver cómo si hacemos una petición HTTP al nodo maestro con el puerto 30080 se nos servirá la página web que hemos colocado en el servidor NFS:

Figura 42: Página de ejemplo del servidor web interno

8. Auditoría de seguridad

En esta sección vamos a ver qué herramientas y procedimientos se pueden utilizar para llevar a cabo una auditoria de seguridad. La idea es instalar una serie de herramientas tanto dentro de la sede como en un dispositivo externo a la misma. El dispositivo externo será el que utilizará un posible auditor para realizar la auditoría. Como una buena auditoría puede requerir tanto acceso externo a la sede como disponer de un acceso dentro de la propia red, nosotros hemos decidido instalar dentro del host 3 la configuración necesaria para acceder a la VPN del nivel de enlace. De esta manera, se podrá simular tanto un ataque externo como uno interno por parte de uno de los trabajadores.

8.1. nmap

La primera herramienta que se usará será `nmap` que se usará para hacer un escaneo preliminar de los puertos. En primer lugar, se debe instalar la herramienta mediante el comando

```
host3 @ host3 : ~  $ sudo apt install nmap
```

Una vez instalado, bastaría con iniciar un escaneo de los puertos como sigue:

```
→  ~ nmap 10.10.0.0/23
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-12-09 11:27 CET
Nmap scan report for overleaf.dorayaki.es (10.10.0.3)
Host is up (0.0074s latency).
Not shown: 950 filtered tcp ports (no-response), 46 closed tcp ports (conn-refused)
PORT    STATE SERVICE
22/tcp  open  ssh
80/tcp  open  http
111/tcp open  rpcbind
179/tcp open  bgp

Nmap scan report for 10.10.0.4
Host is up (0.0075s latency).
Not shown: 997 closed tcp ports (conn-refused)
PORT    STATE SERVICE
22/tcp  open  ssh
111/tcp open  rpcbind
179/tcp open  bgp
```

Figura 43: Escaneo de toda la red mediante nmap

Como podemos ver, nos muestra algunos de los servicios disponibles en algunos de los servidores. Sin embargo, podemos apreciar que el servidor público no aparece y eso se debe a que nmap primero comprueba la disponibilidad del servidor mediante ICMP para luego intentar el escaneo. Como el servidor bloquea los ICMP, nmap no puede detectar que hay un servidor disponible y no intenta el mapeo. Por lo tanto, podemos recurrir a un mapeo directamente a la IP con la opción `-Pn` y podemos ver como sí funciona:

```
→  ~ nmap 10.10.1.2 -Pn
Starting Nmap 7.94SVN ( https://nmap.org ) at 2025-12-09 11:41 CET
Nmap scan report for 10.10.1.2
Host is up (0.0064s latency).
Not shown: 994 filtered tcp ports (no-response)
PORT     STATE  SERVICE
20/tcp   closed ftp-data
21/tcp   open   ftp
25/tcp   open   smtp
110/tcp  open   pop3
143/tcp  open   imap
8090/tcp open   opsmessaging

Nmap done: 1 IP address (1 host up) scanned in 4.75 seconds
```

Figura 44: Escaneo del servidor público mediante nmap

8.2. Greenbone

En este apartado vamos a ver la herramienta llamada Greenbone (antes conocida como OpenVAS) que nos servirá para detectar vulnerabilidades.

Su instalación es sencilla y se realizará en mediante el docker proporcionado por la página oficial:

```
host3 @ host3 : ~ /openvas $ curl -f -O -L https://greenbone.github.io/docs/latest/_static/docker-compose.yml
host3 @ host3 : ~ /openvas $ docker compose up -d
```

Una vez dentro de la página web, solo tenemos que ir a **Scans** y realizar un escaneo completo de los equipos que queremos o incluso de la red completa. A continuación se muestra los escaneos correspondientes a los servidores de la sede Dorayaki:

Figura 45: Escaneo de la sede Dorayaki

Nosotros hemos hecho escaneos con acceso de SSH y por eso nos muestra vulnerabilidades de altísimos riesgo. Sin embargo, si entramos dentro podemos ver que son cuestiones de paquetes desactualizados que requieren la versión de Ubuntu para solucionarse. A continuación se muestra una del Servidor2:

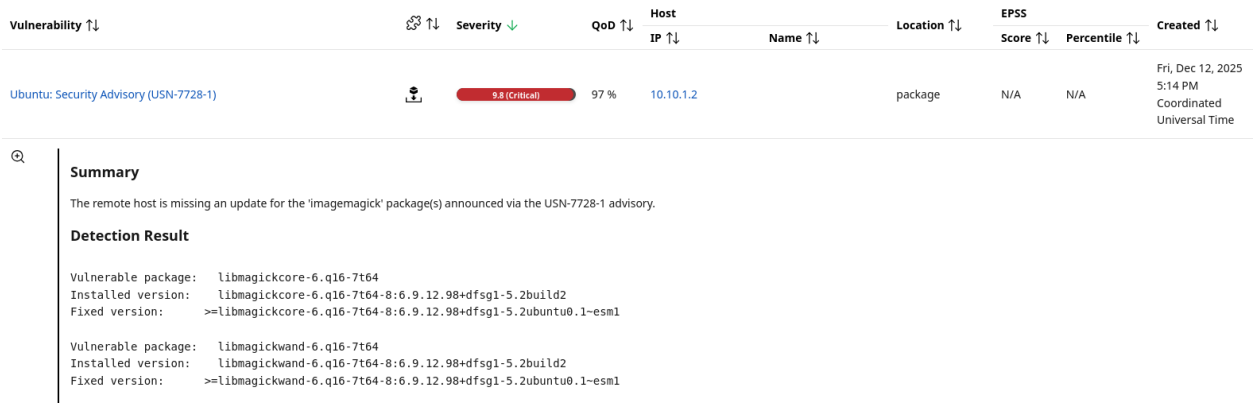


Figura 46: Vulnerabilidad detectada del servidor público

Cabe mencionar que esta herramienta es muy poderosa y nos permite realizar escaneos muy exhaustivos y con una gran personalización. No obstante, nosotros no hemos realizado configuración más profundas puesto que se saldría del objetivo de la memoria.

8.3. Snort

En este apartado vamos a ver Snort. Esta es una herramienta que permite la detección de intrusos en nuestra red y de posibles ataques. La hemos instalado dentro del router de la organización pues nos parecía el punto más idóneo para registrar todos los posibles ataque a la red. La instalación ha sido muy sencilla ejecutando el comando:

```
router_dorayaki @ routerdorayaki : ~  $ sudo apt install snort
```

Esto nos instala la versión 2.9 que aunque no es la más reciente, es más que suficiente para lo que haremos aquí.

En esta instalación, ya se descargan y configuran las `community-rules` que son reglas creadas por la comunidad listas para detectar numerosas amenazas. Además, también ha creado de forma automática un servicio que analiza todas las interfaces de red que tenga el router. Si ahora intentamos hacer un escaneo de puertos a alguna de las IPs de dentro de la red, veremos como Snort nos muestra alertas dentro del fichero `/var/log/snort/snort.alert.fast`:

```
12/04-18:15:44.590695  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification: Attempted Information Leak] [Priority: 2] {TC
P} 10.10.0.200:44750 -> 10.10.0.4:705
12/04-18:15:44.590694  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification: Attempted Information Leak] [Priority: 2] {TC
P} 10.10.0.200:44750 -> 10.10.0.4:705
12/04-18:15:44.590694  [**] [1:1421:11] SNMP AgentX/tcp request [**] [Classification: Attempted Information Leak] [Priority: 2] {TC
P} 10.10.0.200:44750 -> 10.10.0.4:705
12/04-18:15:44.598758  [**] [1:1418:11] SNMP request tcp [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.1
0.0.200:58462 -> 10.10.0.4:161
12/04-18:15:44.598757  [**] [1:1418:11] SNMP request tcp [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.1
0.0.200:58462 -> 10.10.0.4:161
12/04-18:15:44.598757  [**] [1:1418:11] SNMP request tcp [**] [Classification: Attempted Information Leak] [Priority: 2] {TCP} 10.1
0.0.200:58462 -> 10.10.0.4:161
```

Figura 47: Alertas generadas con Snort de un mapeo con nmap

A. Verificación de conectividad

Finalmente, para comprobar que las conexiones entre dispositivos se establecen correctamente, se incluyen algunas evidencias de la ejecución del comando ping y ssh entre dispositivos:

```
host1_dorayaki@asorhost1:~$ ping -c 1 10.10.1.1
PING 10.10.1.1 (10.10.1.1) 56(84) bytes of data.
64 bytes from 10.10.1.1: icmp_seq=1 ttl=64 time=0.668 ms

--- 10.10.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.668/0.668/0.668/0.000 ms
host1_dorayaki@asorhost1:~$ ping -c 1 10.10.1.2
PING 10.10.1.2 (10.10.1.2) 56(84) bytes of data.
64 bytes from 10.10.1.2: icmp_seq=1 ttl=63 time=1.23 ms

--- 10.10.1.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.228/1.228/1.228/0.000 ms
host1_dorayaki@asorhost1:~$ ping -c 1 10.10.0.1
PING 10.10.0.1 (10.10.0.1) 56(84) bytes of data.
64 bytes from 10.10.0.1: icmp_seq=1 ttl=64 time=0.592 ms

--- 10.10.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.592/0.592/0.592/0.000 ms
host1_dorayaki@asorhost1:~$ ping -c 1 10.10.0.3
PING 10.10.0.3 (10.10.0.3) 56(84) bytes of data.
64 bytes from 10.10.0.3: icmp_seq=1 ttl=64 time=0.277 ms

--- 10.10.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.277/0.277/0.277/0.000 ms
```

Figura 48: Conexión satisfactoria entre Host1 y el resto de componentes de la sede Dorayaki.

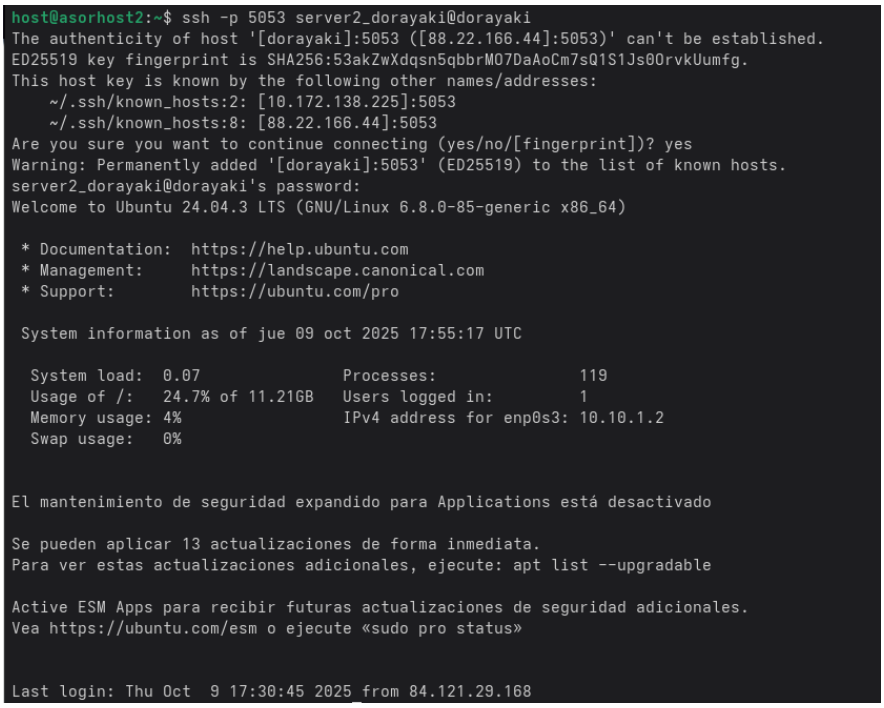


Figura 49: Conexión satisfactoria por ssh del Host2 al Servidor2 de Dorayaki.

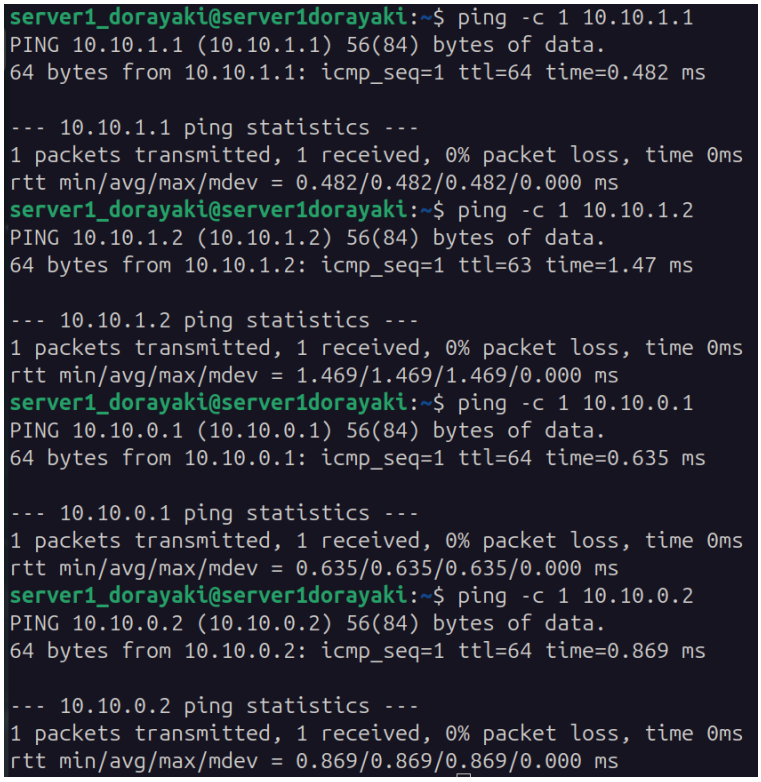


Figura 50: Conexión satisfactoria entre Server1 y el resto de componentes de la sede Dorayaki.

```
server2_dorayaki@server2dorayaki:~$ ping -c 1 10.10.0.1
PING 10.10.0.1 (10.10.0.1) 56(84) bytes of data.
64 bytes from 10.10.0.1: icmp_seq=1 ttl=64 time=0.667 ms

--- 10.10.0.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.667/0.667/0.667/0.000 ms
server2_dorayaki@server2dorayaki:~$ ping -c 1 10.10.0.2
PING 10.10.0.2 (10.10.0.2) 56(84) bytes of data.
64 bytes from 10.10.0.2: icmp_seq=1 ttl=63 time=1.22 ms

--- 10.10.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.215/1.215/1.215/0.000 ms
server2_dorayaki@server2dorayaki:~$ ping -c 1 10.10.0.3
PING 10.10.0.3 (10.10.0.3) 56(84) bytes of data.
64 bytes from 10.10.0.3: icmp_seq=1 ttl=63 time=1.22 ms

--- 10.10.0.3 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 1.224/1.224/1.224/0.000 ms
server2_dorayaki@server2dorayaki:~$ ping -c 1 10.10.1.1
PING 10.10.1.1 (10.10.1.1) 56(84) bytes of data.
64 bytes from 10.10.1.1: icmp_seq=1 ttl=64 time=0.743 ms

--- 10.10.1.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.743/0.743/0.743/0.000 ms
```

Figura 51: Conexión satisfactoria entre Server2 y el resto de componentes de la sede Dorayaki.

```
host@asorhost2:~$ ping -c 1 10.10.2.1
PING 10.10.2.1 (10.10.2.1) 56(84) bytes of data.
64 bytes from 10.10.2.1: icmp_seq=1 ttl=64 time=0.651 ms

--- 10.10.2.1 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.651/0.651/0.651/0.000 ms
```

Figura 52: Conexión satisfactoria entre Host2 y el router de "No confiable".

B. Estudio de otros túneles

B.1. Túnel GRE

Un túnel GRE permite conectar de forma sencilla dos subredes y con poco overhead. Esto lo haría la tecnología perfecta para conectas la sede Dorayaki con la sede "No confiable"de forma que se pueda acceder de forma remota a lo dispositivos. Sin embargo, GRE no dispone por sí mismo de un mecanismo para encriptar, autenticar y proteger los mensajes. La forma de sortear este problema es utilizando IPSec como protocolo de seguridad.

Aunque ya hemos comentado que es posible establecer un túnel seguro con GRE + IPSec, nosotros hemos considerado que no es recomendable realizarlo. A continuación enumeramos los motivos por los que hemos decidido no implementarlo:

1. Aunque se puede securizar GRE mediante IPSec, este último tiene una gestión compleja y muchas veces no es compatible con algunos escenarios

2. GRE utiliza el puerto 47 (IP) por lo que montarlo tras un NAT resulta complicado. En nuestro escenario

como cada uno de nosotros tiene una sede ambos tenemos que hacer NAT con los routers y no siempre pueden estar conectados a la misma red. Además, como recurrimos a las redes de nuestras casa el propio router no es configurable para dejar pasar las tramas GRE.

3. Con GRE no existe un mecanismo de para que cada uno de los extremos del túnel informe de cuales son las subredes a las que se pueden acceder a través de él. Esto ocasiona que haya que realizar una configuración extra por parte de ambos routers para comunicarse las rutas.

4. Un túnel GRE cuando se establece es una interfaz virtual y no es más que encapsular los paquetes con una cabecera GRE. Por lo tanto, no hay ningún mecanismo para saber si el túnel, aunque esté establecido por un extremo, ha sido establecido por las dos partes. Si enviásemos paquetes por el túnel, se perderías y no recibiríamos respuesta y quizá sea porque el otro extremo no haya montado el túnel.

Por todos estos motivos, hemos decidido no implementar un túnel GRE en el proyecto.

B.2. Túnel L2TP

Un túnel L2TP nos permite simular que un dispositivo remoto se encuentra conectado a la misma LAN que el otro extremo del túnel. Es extremadamente útil cuando se quieren permitir trabajo remoto como si se estuviera en la sede. Sin embargo, L2TP no dispone de encriptación e integridad (aunque sí autenticación). Al igual que GRE, se puede sortear este problema mediante el uso de IPSec.

De nuevo, aunque es viable establecer el túnel L2TP + IPSec, nosotros hemos considerado que no es recomendable realizarlo. A continuación enumeramos los motivos por los que hemos decidido no implementarlo:

1. Aunque se puede securizar L2TP mediante IPSec, este último tiene una gestión compleja y muchas veces no es compatible con algunos escenarios.

2. Con L2TP no existe un mecanismo de para que cada uno de los extremos del túnel informe de cuales son las subredes a las que se pueden acceder a través de él. Esto ocasiona que haya que realizar una configuración extra por parte de ambos routers para comunicarse las rutas. Lo más cercano que existe sería hacer uso de un script a ejecutar en el cliente cuando se levante y se cierre el túnel mediante `ip-up` e `ip-down`. Sin embargo, aunque lo hemos probado y es viable, consideramos que esto implica una configuración extra compleja en el cliente. Esto dificulta bastante el uso del túnel por un usuario menos experimentado.

Por todos estos motivos, hemos decidido no implementar un túnel L2TP en el proyecto.

C. Instalación de K8s

C.1. Configuración del clúster

Aquí vamos a explicar cómo se ha desplegado todo el clúster de K8s. Aunque se podría haber seguido un esquema de clúster de pruebas usando Minikube, nosotros hemos decidido seguir un esquema más cercano a la realidad usando `kubeadm`. Montaremos un clúster con un solo nodo maestro que tendrá acoplado tanto el `control-plane` como el `etcd` dentro del Servidor4 y luego 2 nodos workers que serán el Servidor1 y el Servidor3 de la red privada. A continuación vamos a mostrar los pasos a seguir tanto en el nodo maestro como en los workers. Antes de pasar con los pasos concretos, vamos a ver la configuración común a ambos nodos.

C.1.1. Configuración general de los nodos

C.1.1.1. Configuraciones previas

En primer lugar, vamos a necesitar tener Docker instalado. Vamos a suponer Docker instalado. Ahora tenemos que habilitar ciertos módulos. En particular: `overlay` y `br_netfilter`

Para hacerlo, hay que usar el directorio `/etc/modules-load.d/` y dentro crear un archivo con el nombre `k8s.conf` y el siguiente contenido:

```
overlay
br_netfilter
```

Además, hay que indicarle al sistema que debemos actuar como router y que también tenemos que aplicar las reglas de iptables a la interfaz bridge. Para ello, tenemos que usar el directorio `/etc/sysctl.d/` dentro crear un archivo con el nombre `k8s.conf` y el siguiente contenido:

```
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
```

Ahora aplicamos la configuración mediante el comando

```
user @ host : ~ $ sudo sysctl –system
```

Con todo esto, ya tendríamos los módulos cargados correctamente.

Tal y como se indica en la documentación oficial de K8s, es recomendable desactivar la memoria swap para un correcto funcionamiento. Para ello, debemos de ejecutar el siguiente comando:

```
user @ host : ~ $ sudo swapoff -a
```

Aunque esto la desactivará durante esta sesión, volverá a montarse la próxima vez que se inicie la máquina. Para evitarlo, habría que eliminar (aunque mejor comentar) la entrada del swap de `/etc/fstab` que habrá una entrada del estilo:

```
    /swap.img       none    swap    sw      0       0
```

De nuevo, tal y como se indica en la documentación oficial de K8s, se recomienda desactivar el manejo de los cgroups de containerd por parte del driver `cgroupfs` y usar el driver `systemd`. Para hacer este cambio, habría que ejecutar los siguientes comandos:

```
sudo containerd config default | sudo tee /etc/containerd/config.toml >/dev/null 2>&1
sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
sudo systemctl restart containerd
```

C.1.1.2. Instalación de K8s

Ahora ya se puede proceder con la instalación propiamente dicha. Para ello, y siguiendo los pasos que indica la documentación oficial de K8s, vamos a ejecutar los siguiente comandos:

```
# Install all the required packages to proceed with the installation
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl gpg
# Add the K8s' official GPG key
# (only the 1.34 version, if a newer is need check out the documentation)
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.34/deb/Release.key | \
    sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
# Add the official K8s' repo
# (only the 1.34 version, if a newer is need check out the documentation)
echo \
'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] \
https://pkgs.k8s.io/core:/stable:/v1.34/deb/ /' | \
sudo tee /etc/apt/sources.list.d/kubernetes.list
# Install K8s
```

```
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
# Enable kubelet
sudo systemctl enable --now kubelet
```

Con todo esto, ya tenemos K8s instalado junto con la herramienta `kubeadm` que usaremos para gestionar el clúster.

C.1.2. Configuración nodo maestro

C.1.2.1. Firewall

Lo primero sería garantizar que el tráfico del loopback está permitido y que todas las peticiones relacionadas con conexiones ya establecidas se mantengan. Sin embargo, eso ya lo hemos cubierto en la sección del firewall y no tenemos aquí que preocuparnos.

El principal puerto que hay que habilitar es el correspondiente al Kubernetes API Server que usarán los nodos workers para comunicarse con el nodo maestro. Este servicio utiliza el puerto 6443 y debe estar habilitado para todas las IPs pues se usará tanto desde la red privada como desde la red de los pods. Para habilitarlo, debemos de ejecutar el siguiente comando:

```
server4_dorayaki @ server4dorayaki : ~ $ sudo iptables -A INPUT -p TCP –dport 6443 -j ACCEPT
```

Ahora faltaría habilitar los puertos para la CNI que vamos a montar. Como será con Calico, esta solo necesita el puerto 179 para la comunicación BGP y se habilitaría con el siguiente comando:

```
server4_dorayaki @ server4dorayaki : ~ $ sudo iptables -A INPUT -p TCP –dport bgp -j ACCEPT
```

C.1.2.2. Creación del clúster

Ahora vamos a proceder con la inicialización del clúster mediante el comando:

```
    sudo kubeadm init --pod-network-cidr=10.200.0.0/16 \
                                        --apiserver-advertise-address=10.10.0.4 --v=5
```

Aquí estamos indicando que la red de nuestros pods será la `10.200.0.0/16` y que la IP en la que la api escuchará será la de nuestro servidor (la `10.10.0.4`). Una vez ejecutado el comando completamente vamos a llevar a cabo la sugerencia que nos permitirá gestionar el clúster sin ser usuario root:

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Además, se habrá generado un comando para unir a los nodos workers que hay que guardar, el nuestro es este:

```
kubeadm join 10.10.0.4:6443 \
        --token 106uic.b0h5sm0i2u5jm69f \
        --discovery-token-ca-cert-hash \
        sha256:6a9fed1933c90db5810cb6c5bf99f78384c7033de75443d94af119e09e0c0217
```

Finalmente, solo nos quedaría montar la red de pods (CNI). Aunque el clúster esté montado, no se crea por defecto ningún tipo de red virtual que permita conectar a los pods entre ellos. Por lo tanto, tenemos que crearla nosotros. Hay una infinitud de opciones, pero una de las más sencillas es Calico. Será la que instalemos aquí mediante el siguiente comando:

```
# Descargamos el YAML de la configuración
server4_dorayaki @ server4dorayaki : ~ $ curl https://raw.githubusercontent.com/projectcalico/calico
-O
# Creamos la CNI
server4_dorayaki @ server4dorayaki : ~  $ kubectl create -f calico.yaml
```

Con todo esto, ahora sí que tenemos perfectamente configurado el clúster y listo para funcionar.

C.1.3. Configuración nodos worker

C.1.3.1. Firewall

Lo primero sería garantizar que el tráfico del loopback está permitido y que todas las peticioens relacionadas con conexiones ya establecidas se mantengan. Sin embargo, eso ya lo hemos cubierto en la sección del firewall y no tenemos aquí que preocuparnos.

El primero puerto que hay que habilitar es el correspondiente al kubelet API y que permite que los nodos maestros se comuniquen con el worker. Este es el puerto 10250 y lo habilitamos de la siguiente manera:

```
worker @ worker : ~  $ sudo iptables -A INPUT -p TCP –dport 10250 -j ACCEPT
```

Con esto ya tendríamos lo que necesitamos para la comunicación. Ahora faltaría habilitar los puertos para la CNI que vamos a montar. Como será con Calico, esta solo necesita el puerto 179 para la comunicación BGP y se habilitaría con el siguiente comando:

```
worker @ worker : ~  $ sudo iptables -A INPUT -p TCP –dport bgp -j ACCEPT
```

C.1.3.2. Unión al clúster

Esto es tan sencillo como ejecutar el comando que hemos comentado anteriormente en los nodos workers:

```
server1_dorayaki @ server1dorayaki : ~  $ sudo kubeadm join 10.10.0.4:6443
–token 106uic.b0h5sm0i2u5jm69f
–discovery-token-ca-cert-hash
sha256:6a9fed1933c90db5810cb6c5bf99f78384c7033de75443d94af119e09e0c0217
server1_dorayaki @ server1dorayaki : ~  $ sudo kubeadm join 10.10.0.4:6443
–token 106uic.b0h5sm0i2u5jm69f
–discovery-token-ca-cert-hash
sha256:6a9fed1933c90db5810cb6c5bf99f78384c7033de75443d94af119e09e0c0217
```

Ahora podemos ejecutar el siguiente comando en el nodo mastro para ver qué nodos hay en el clúster:

```
server4_dorayaki @ server4dorayaki : ~  $ kubectl get nodes
```

Y veremos el siguiente resultado:

```
server4_dorayaki@server4dorayaki:~/recursor-dns$ kubectl get nodes
NAME              STATUS   ROLES           AGE   VERSION
server1dorayaki   Ready    <none>          11d   v1.34.2
server3dorayaki   Ready    <none>          12m   v1.34.2
server4dorayaki   Ready    control-plane   12d   v1.34.2
```

Figura 53: Nodos del clúster

C.2. Instalación del Dashboard

Aunque el clúster ya estaría instalado y funcionando, resulta algo engorroso trabajar siempre con la línea de comandos `kubectl` y para ello K8s ofrece su propio dashboard. Vamos a instalarlo también para trabajar con el clúster de forma sencilla.

La propia documentación te informa que, actualmente, la forma de instalarlo es mediante `helm` así que usaremos ese método.

Lo primero es instalarlo y para ello (siguiendo su documentación oficial) hay que ejecutar el siguiente comando en el nodo maestro:

```
server4_dorayaki @ server4dorayaki : ~ $ curl https://raw.githubusercontent.com/helm/helm/main/sc
helm-4 — bash
```

Una vez que tenemos este gestor, tal y como nos dice la documentación del dashboard de K8s, debemos de ejecutar los siguientes dos comandos:

```
# Add kubernetes-dashboard repository
helm repo add kubernetes-dashboard https://kubernetes.github.io/dashboard/
# Deploy a Helm Release named "kubernetes-dashboard" using the kubernetes-dashboard chart
helm upgrade --install kubernetes-dashboard kubernetes-dashboard/kubernetes-dashboard \
             --create-namespace --namespace kubernetes-dashboard
```

Esto despliega el dashboard, pero no es accesible. Para permitir el acceso, habrá que redirigir el puerto del servicio al puerto local que queramos. Aunque se debería hacer de forma adecuada usando un Ingress, por simplicidad, vamos a usar simplemente una redirección mediante el propio `kubectl`. Para ello, solo necesitamos abrir el puerto 8443 a la red privada[12] mediante el comando:

```
server4_dorayaki @ server4dorayaki : ~ $ sudo iptables -A INPUT -p TCP -s 10.10.0.0/24 –dport
8443 -j ACCEPT
```

Antes de activar la redicción de puertos, vamos a crear el usuario que usaremos para acceder al dashboard. Para ello, debemos hacer hacer un `kubectl apply -f` a un yaml con el siguiente contenido:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
```

Con esto crearemos un usuario con todos los permisos del clúster y podremos obtener su token de acceso

[12] Lo ideal sería restringir el acceso a un conjunto de IPs, pero lo haremos a toda la red ahora por simplificar

mediante el comando:

```
server4_dorayaki @ server4dorayaki : ~ $ kubectl -n kubernetes-dashboard create token admin-user
```

Y ese será el token que usaremos para acceder.

Finalmente, podemos hacer el proxy del Dashboard con el comando:

```
kubectl -n kubernetes-dashboard --address 10.10.0.4 \
                                port-forward svc/kubernetes-dashboard-kong-proxy 8443:443
```

Aquí podemos ver el Dashboard en funcionamiento:

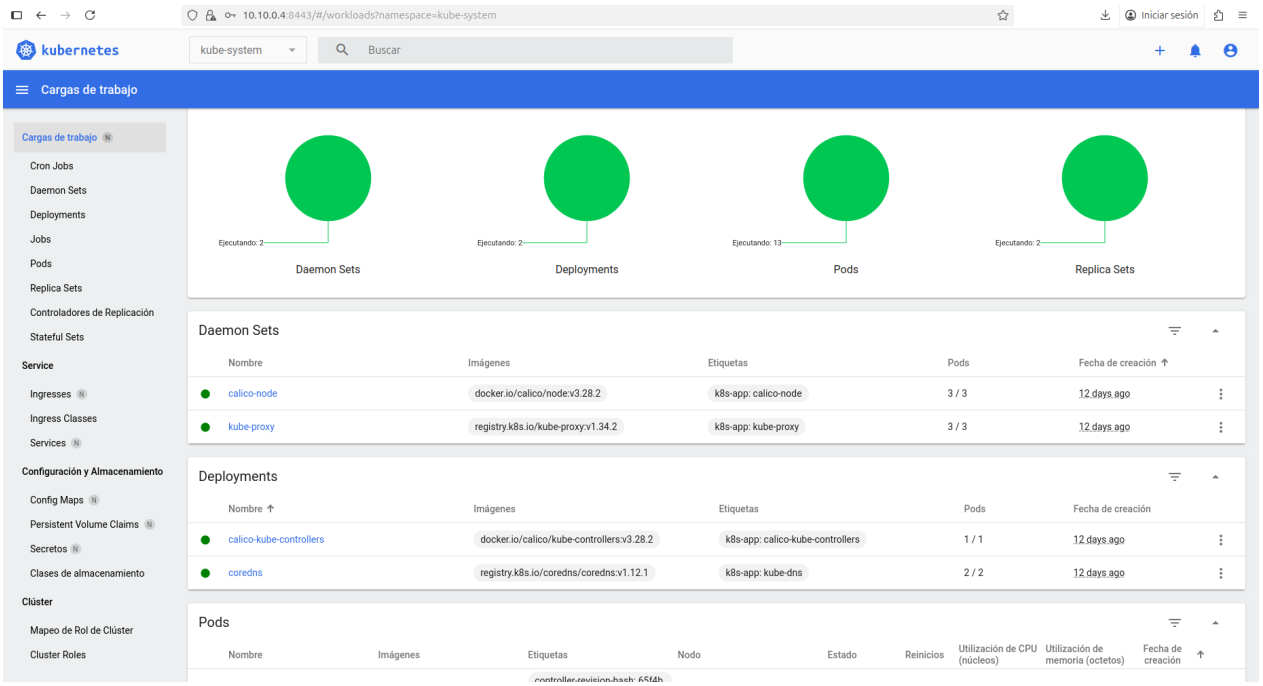


Figura 54: Dashboard de K8s

D. Arquitectura final web K8s

A continuación vamos a mostar un esquema que muestra cómo quedaría nuestra arquitectura finalmente con el los servidores que intervienen y los pods Nginx desplegados. Aunque en el deployment se ha especificado 3 réplicas, en el esquema solo aparecerán 2 pues se trata de un esquema conceptual para representar la arquitectura. Además, también hemos decidido simplificar el esquema eliminando aquellos elementos que no terminan de aportar gran valor como son todos los pods relacionados con la red de Calico o el API server. A continuación se muestra el esquema en cuestión:

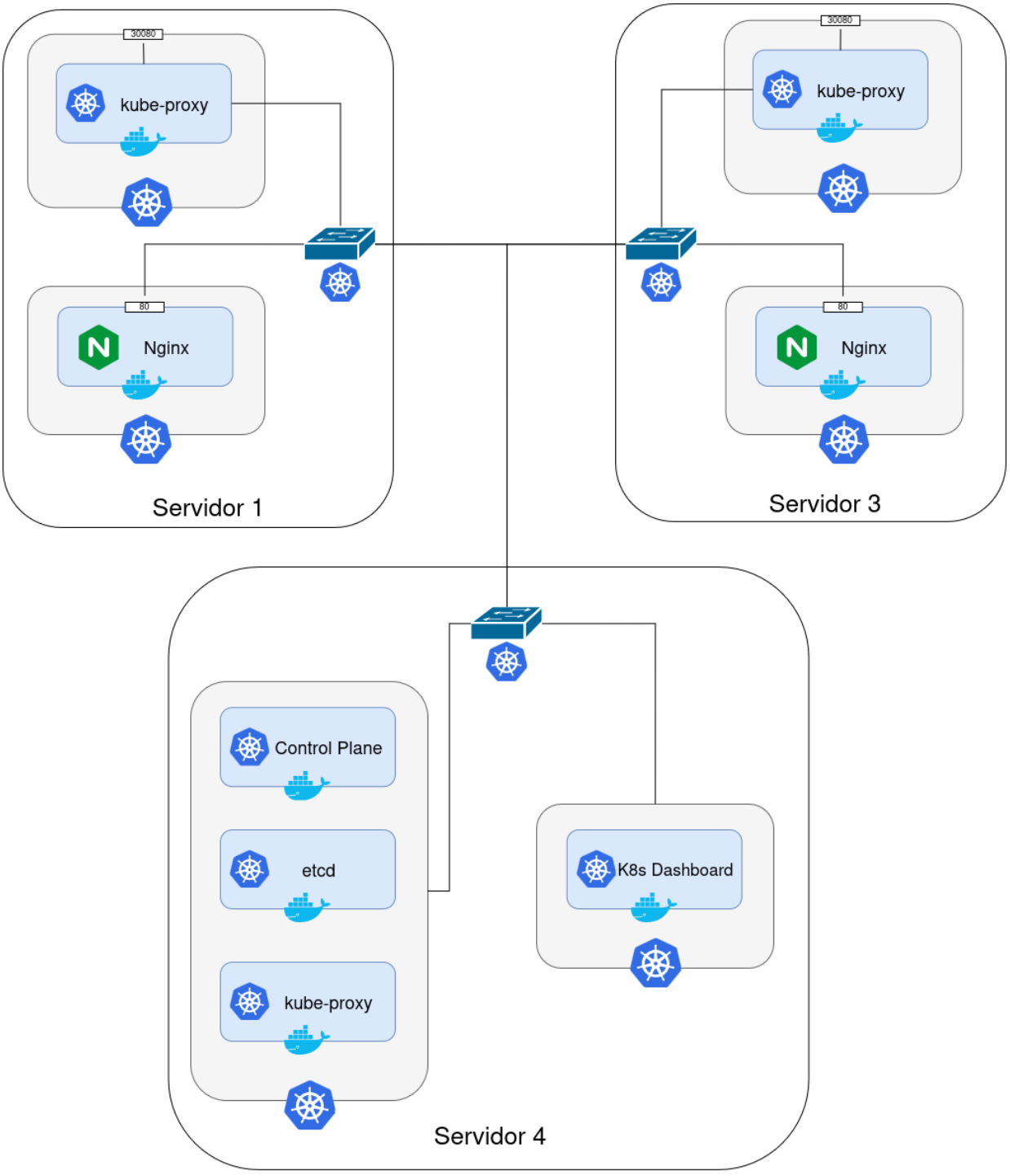


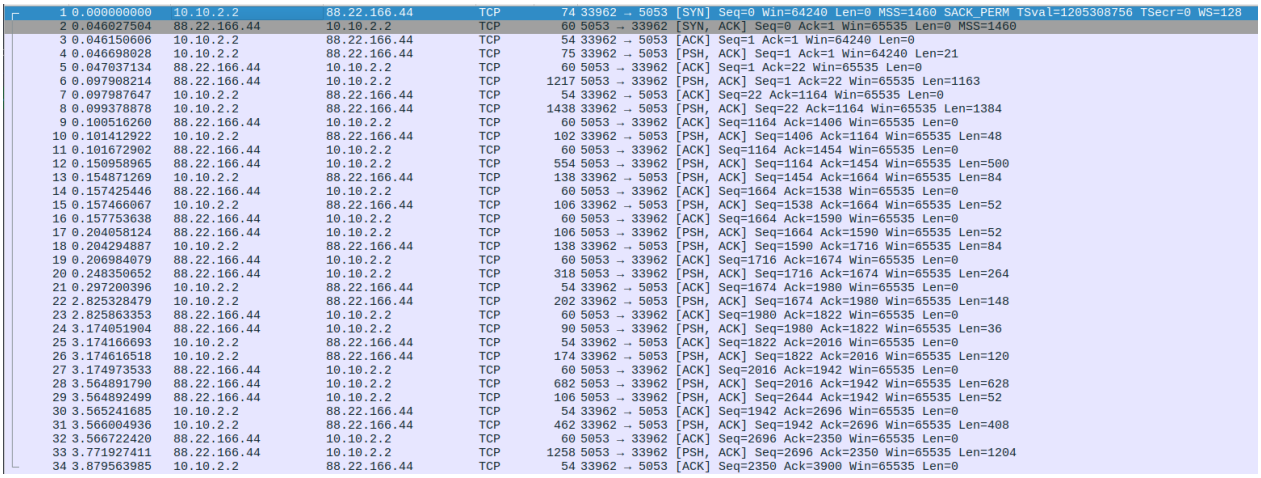
Figura 55: Esquema de la arquitectura de K8s

E. Trazas

En esta sección vamos a ver todas las trazas de Wireshark provenientes de nuestros dispositivos virtualizados.

E.1. Host 2 (sede "No confiable")

Vamos a ver cómo se inicia la conexión SSH desde el host 2 de la red "No confiable".



1	0.000000000	10.10.2.2	88.22.166.44	TCP	74 33962 → 5053 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1205308756 TSecr=0 WS=128
2	0.046027504	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
3	0.046150606	10.10.2.2	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1 Ack=1 Win=64240 Len=0
4	0.046698028	10.10.2.2	88.22.166.44	TCP	75 33962 → 5053 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=21
5	0.047037134	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1 Ack=22 Win=65535 Len=0
6	0.097908214	88.22.166.44	10.10.2.2	TCP	1217 5053 → 33962 [PSH, ACK] Seq=1 Ack=22 Win=65535 Len=1163
7	0.097987647	10.10.2.2	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=22 Ack=1164 Win=65535 Len=0
8	0.099378878	10.10.2.2	88.22.166.44	TCP	1438 33962 → 5053 [PSH, ACK] Seq=22 Ack=1164 Win=65535 Len=1384
9	0.100516260	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1164 Ack=1406 Win=65535 Len=0
10	0.101412922	10.10.2.2	88.22.166.44	TCP	102 33962 → 5053 [PSH, ACK] Seq=1406 Ack=1164 Win=65535 Len=48
11	0.101672902	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1164 Ack=1454 Win=65535 Len=0
12	0.150958965	88.22.166.44	10.10.2.2	TCP	554 5053 → 33962 [PSH, ACK] Seq=1164 Ack=1454 Win=65535 Len=500
13	0.154871269	10.10.2.2	88.22.166.44	TCP	138 33962 → 5053 [PSH, ACK] Seq=1454 Ack=1664 Win=65535 Len=84
14	0.157425446	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1664 Ack=1538 Win=65535 Len=0
15	0.157466067	10.10.2.2	88.22.166.44	TCP	106 33962 → 5053 [PSH, ACK] Seq=1538 Ack=1664 Win=65535 Len=52
16	0.157753638	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1664 Ack=1590 Win=65535 Len=0
17	0.204058124	88.22.166.44	10.10.2.2	TCP	106 5053 → 33962 [PSH, ACK] Seq=1664 Ack=1590 Win=65535 Len=52
18	0.204294887	10.10.2.2	88.22.166.44	TCP	138 33962 → 5053 [PSH, ACK] Seq=1590 Ack=1716 Win=65535 Len=84
19	0.206984079	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1716 Ack=1674 Win=65535 Len=0
20	0.248350652	88.22.166.44	10.10.2.2	TCP	318 5053 → 33962 [PSH, ACK] Seq=1716 Ack=1674 Win=65535 Len=264
21	0.297200396	10.10.2.2	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1674 Ack=1980 Win=65535 Len=0
22	2.825328479	10.10.2.2	88.22.166.44	TCP	202 33962 → 5053 [PSH, ACK] Seq=1674 Ack=1980 Win=65535 Len=148
23	2.825863353	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=1980 Ack=1822 Win=65535 Len=0
24	3.174051904	88.22.166.44	10.10.2.2	TCP	90 5053 → 33962 [PSH, ACK] Seq=1980 Ack=1822 Win=65535 Len=36
25	3.174166693	10.10.2.2	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1822 Ack=2016 Win=65535 Len=0
26	3.174616518	10.10.2.2	88.22.166.44	TCP	174 33962 → 5053 [PSH, ACK] Seq=1822 Ack=2016 Win=65535 Len=120
27	3.174973533	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=2016 Ack=1942 Win=65535 Len=0
28	3.564891790	88.22.166.44	10.10.2.2	TCP	682 5053 → 33962 [PSH, ACK] Seq=2016 Ack=1942 Win=65535 Len=628
29	3.564892499	88.22.166.44	10.10.2.2	TCP	106 5053 → 33962 [PSH, ACK] Seq=2644 Ack=1942 Win=65535 Len=52
30	3.565241685	10.10.2.2	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1942 Ack=2696 Win=65535 Len=0
31	3.566004936	10.10.2.2	88.22.166.44	TCP	462 33962 → 5053 [PSH, ACK] Seq=1942 Ack=2696 Win=65535 Len=408
32	3.566722420	88.22.166.44	10.10.2.2	TCP	60 5053 → 33962 [ACK] Seq=2696 Ack=2350 Win=65535 Len=0
33	3.771927411	88.22.166.44	10.10.2.2	TCP	1258 5053 → 33962 [PSH, ACK] Seq=2696 Ack=2350 Win=65535 Len=1204
34	3.879563985	10.10.2.2	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=2350 Ack=3900 Win=65535 Len=0

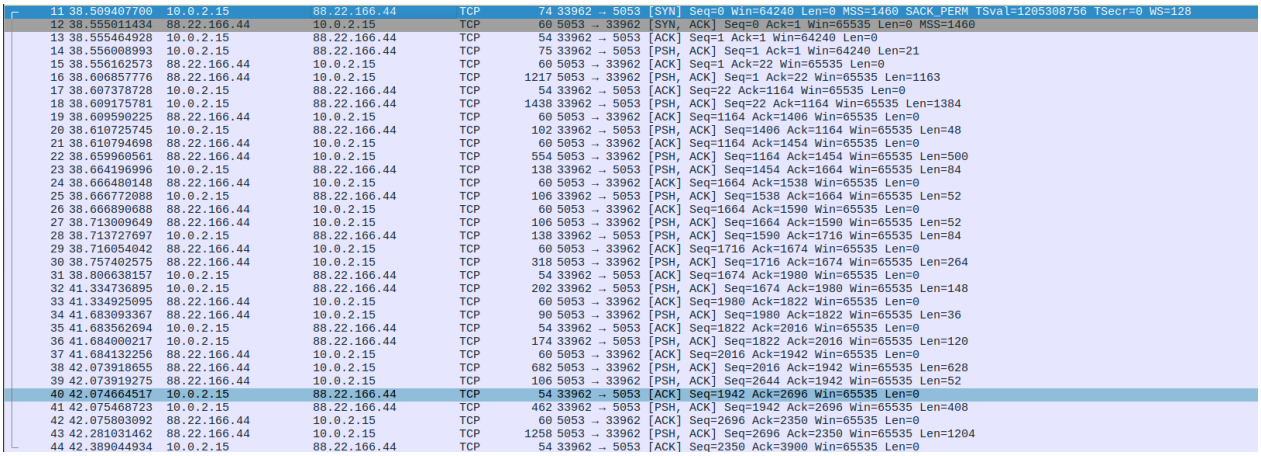
Figura 56: Trazas host 2 (sede "No confiable")

Podemos ver cómo la IP destino es la `88.22.166.44` que corresponde a la IP pública del equipo que contiene la sede Dorayaki. Además, podemos ver que el puerto destino es el `5053` que corresponde al asignado por la sede Dorayaki para ese servicio.

Por otro lado, podemos ver que el puerto origen es uno generado aleatoriamente (`33962`) y que la IP origen corresponde a la del host 2 (`10.10.2.2`). Más adelante veremos como el router de la sede "No confiable"hará source NAT y modificará esos parámetros.

E.2. Router (sede "No confiable")

Veamos cómo los paquetes fluyen desde el host atravesando el router:



11	38.509407700	10.0.2.15	88.22.166.44	TCP	74 33962 → 5053 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=1205308756 TSecr=0 WS=128
12	38.555011434	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
13	38.555464928	10.0.2.15	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1 Ack=1 Win=64240 Len=0
14	38.556008993	10.0.2.15	88.22.166.44	TCP	75 33962 → 5053 [PSH, ACK] Seq=1 Ack=1 Win=64240 Len=21
15	38.556162573	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1 Ack=22 Win=65535 Len=0
16	38.606857776	88.22.166.44	10.0.2.15	TCP	1217 5053 → 33962 [PSH, ACK] Seq=1 Ack=22 Win=65535 Len=1163
17	38.607378728	10.0.2.15	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=22 Ack=1164 Win=65535 Len=0
18	38.609175781	10.0.2.15	88.22.166.44	TCP	1438 33962 → 5053 [PSH, ACK] Seq=22 Ack=1164 Win=65535 Len=1384
19	38.609590225	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1164 Ack=1406 Win=65535 Len=0
20	38.610725745	10.0.2.15	88.22.166.44	TCP	102 33962 → 5053 [PSH, ACK] Seq=1406 Ack=1164 Win=65535 Len=48
21	38.610794698	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1164 Ack=1454 Win=65535 Len=0
22	38.659960561	88.22.166.44	10.0.2.15	TCP	554 5053 → 33962 [PSH, ACK] Seq=1164 Ack=1454 Win=65535 Len=500
23	38.664196996	10.0.2.15	88.22.166.44	TCP	138 33962 → 5053 [PSH, ACK] Seq=1454 Ack=1664 Win=65535 Len=84
24	38.666480148	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1664 Ack=1538 Win=65535 Len=0
25	38.666772088	10.0.2.15	88.22.166.44	TCP	106 33962 → 5053 [PSH, ACK] Seq=1538 Ack=1664 Win=65535 Len=52
26	38.666890688	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1664 Ack=1590 Win=65535 Len=0
27	38.713009649	88.22.166.44	10.0.2.15	TCP	106 5053 → 33962 [PSH, ACK] Seq=1664 Ack=1590 Win=65535 Len=52
28	38.713727697	10.0.2.15	88.22.166.44	TCP	138 33962 → 5053 [PSH, ACK] Seq=1590 Ack=1716 Win=65535 Len=84
29	38.716054042	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1716 Ack=1674 Win=65535 Len=0
30	38.757402575	88.22.166.44	10.0.2.15	TCP	318 5053 → 33962 [PSH, ACK] Seq=1716 Ack=1674 Win=65535 Len=264
31	38.806638157	10.0.2.15	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1674 Ack=1980 Win=65535 Len=0
32	41.334736895	10.0.2.15	88.22.166.44	TCP	202 33962 → 5053 [PSH, ACK] Seq=1674 Ack=1980 Win=65535 Len=148
33	41.334925095	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=1980 Ack=1822 Win=65535 Len=0
34	41.683093367	88.22.166.44	10.0.2.15	TCP	90 5053 → 33962 [PSH, ACK] Seq=1980 Ack=1822 Win=65535 Len=36
35	41.683562694	10.0.2.15	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1822 Ack=2016 Win=65535 Len=0
36	41.684000217	10.0.2.15	88.22.166.44	TCP	174 33962 → 5053 [PSH, ACK] Seq=1822 Ack=2016 Win=65535 Len=120
37	41.684132256	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=2016 Ack=1942 Win=65535 Len=0
38	42.073918655	88.22.166.44	10.0.2.15	TCP	682 5053 → 33962 [PSH, ACK] Seq=2016 Ack=1942 Win=65535 Len=628
39	42.073919275	88.22.166.44	10.0.2.15	TCP	106 5053 → 33962 [PSH, ACK] Seq=2644 Ack=1942 Win=65535 Len=52
40	42.074664517	10.0.2.15	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=1942 Ack=2696 Win=65535 Len=0
41	42.075468723	10.0.2.15	88.22.166.44	TCP	462 33962 → 5053 [PSH, ACK] Seq=1942 Ack=2696 Win=65535 Len=408
42	42.075803092	88.22.166.44	10.0.2.15	TCP	60 5053 → 33962 [ACK] Seq=2696 Ack=2350 Win=65535 Len=0
43	42.281031462	88.22.166.44	10.0.2.15	TCP	1258 5053 → 33962 [PSH, ACK] Seq=2696 Ack=2350 Win=65535 Len=1204
44	42.389044934	10.0.2.15	88.22.166.44	TCP	54 33962 → 5053 [ACK] Seq=2350 Ack=3900 Win=65535 Len=0

Figura 57: Trazas router (sede "No confiable")

Aquí podemos ver cómo los paquetes del SSH con puerto destino `5053` y la misma IP antes descrita han visto modificada su IP origen. Esto se debe a que el router ha realizado source NAT para enmascarar la red privada. Además, la IP se ha sustituido por la `10.0.2.15` que corresponde a la interfaz del adaptador NAT

del router. Esto es para que el equipo anfitrión sepa cómo se debe encaminar el paquete respuesta cuando lo obtenga.

E.3. Router (sede dorayaki)

Veamos a ver cómo son los paquetes que llegan a la sede dorayaki:

1	0.000000000	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
2	0.000533514	10.0.2.15	31.221.203.21	TCP	58	5053 → 11648 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.000630336	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.000780108	31.221.203.21	10.0.2.15	TCP	78	11648 → 5053 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=24
5	0.001003378	10.0.2.15	31.221.203.21	TCP	54	5053 → 11648 [ACK] Seq=1 Ack=25 Win=64216 Len=0
6	0.024746538	10.0.2.15	31.221.203.21	TCP	97	5053 → 11648 [PSH, ACK] Seq=1 Ack=25 Win=64216 Len=43
7	0.024959489	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=25 Ack=44 Win=65535 Len=0
8	0.028423941	10.0.2.15	31.221.203.21	TCP	1174	5053 → 11648 [PSH, ACK] Seq=44 Ack=25 Win=64216 Len=1120
9	0.028561169	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=25 Ack=1164 Win=65535 Len=0
10	0.367738997	31.221.203.21	10.0.2.15	TCP	1442	11648 → 5053 [PSH, ACK] Seq=25 Ack=1164 Win=65535 Len=1388
11	0.409433693	10.0.2.15	31.221.203.21	TCP	54	5053 → 11648 [ACK] Seq=1164 Ack=1413 Win=65535 Len=0
12	0.442132014	31.221.203.21	10.0.2.15	TCP	762	11648 → 5053 [PSH, ACK] Seq=1413 Ack=1164 Win=65535 Len=708
13	0.442575278	10.0.2.15	31.221.203.21	TCP	54	5053 → 11648 [ACK] Seq=1164 Ack=2121 Win=65535 Len=0
14	0.447780698	31.221.203.21	10.0.2.15	TCP	102	11648 → 5053 [PSH, ACK] Seq=2121 Ack=1164 Win=65535 Len=48
15	0.448148400	10.0.2.15	31.221.203.21	TCP	54	5053 → 11648 [ACK] Seq=1164 Ack=2169 Win=65535 Len=0
16	0.457956565	10.0.2.15	31.221.203.21	TCP	626	5053 → 11648 [PSH, ACK] Seq=1164 Ack=2169 Win=65535 Len=572
17	0.458170718	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2169 Ack=1736 Win=65535 Len=0
18	0.562955894	31.221.203.21	10.0.2.15	TCP	70	11648 → 5053 [PSH, ACK] Seq=2169 Ack=1736 Win=65535 Len=16
19	0.569696063	31.221.203.21	10.0.2.15	TCP	106	11648 → 5053 [PSH, ACK] Seq=2185 Ack=1736 Win=65535 Len=52
20	0.570069897	10.0.2.15	31.221.203.21	TCP	54	5053 → 11648 [ACK] Seq=1736 Ack=2237 Win=65535 Len=0
21	0.570191816	10.0.2.15	31.221.203.21	TCP	106	5053 → 11648 [PSH, ACK] Seq=1736 Ack=2237 Win=65535 Len=52
22	0.570292937	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2237 Ack=1788 Win=65535 Len=0
23	0.688923477	31.221.203.21	10.0.2.15	TCP	138	11648 → 5053 [PSH, ACK] Seq=2237 Ack=1788 Win=65535 Len=84
24	0.692779576	10.0.2.15	31.221.203.21	TCP	106	5053 → 11648 [PSH, ACK] Seq=1788 Ack=2321 Win=65535 Len=52
25	0.695184083	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2321 Ack=1840 Win=65535 Len=0
26	0.797400161	31.221.203.21	10.0.2.15	TCP	154	11648 → 5053 [PSH, ACK] Seq=2321 Ack=1840 Win=65535 Len=100
27	0.809507014	10.0.2.15	31.221.203.21	TCP	90	5053 → 11648 [PSH, ACK] Seq=1840 Ack=2421 Win=65535 Len=36
28	0.811768842	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2421 Ack=1876 Win=65535 Len=0
29	0.889143374	10.0.2.15	31.221.203.21	TCP	682	5053 → 11648 [PSH, ACK] Seq=1876 Ack=2421 Win=65535 Len=628
30	0.889311781	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2421 Ack=2504 Win=65535 Len=0
31	0.933644581	31.221.203.21	10.0.2.15	TCP	106	11648 → 5053 [PSH, ACK] Seq=2421 Ack=2504 Win=65535 Len=52
32	0.935099470	10.0.2.15	31.221.203.21	TCP	106	5053 → 11648 [PSH, ACK] Seq=2504 Ack=2473 Win=65535 Len=52
33	0.937464312	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2473 Ack=2556 Win=65535 Len=0
34	1.053311222	31.221.203.21	10.0.2.15	TCP	138	11648 → 5053 [PSH, ACK] Seq=2473 Ack=2556 Win=65535 Len=84
35	1.055638153	10.0.2.15	31.221.203.21	TCP	90	5053 → 11648 [PSH, ACK] Seq=2556 Ack=2557 Win=65535 Len=36
36	1.058013675	31.221.203.21	10.0.2.15	TCP	60	11648 → 5053 [ACK] Seq=2557 Ack=2592 Win=65535 Len=0
37	1.168723706	31.221.203.21	10.0.2.15	TCP	106	11648 → 5053 [PSH, ACK] Seq=2557 Ack=2592 Win=65535 Len=52
38	1.171143120	10.0.2.15	31.221.203.21	TCP	126	5053 → 11648 [PSH, ACK] Seq=2592 Ack=2609 Win=65535 Len=72

Figura 58: Trazas router (sede dorayaki)

Podemos ver que la IP origen es completamente diferente a la `10.0.2.15` que veíamos en la última traza. Esto se debe a que el router del ordenador anfitrión de la sede Dorayaki también ha realizado source NAT y ha cambiado la IP por la pública del anfitrión de la sede "No confiable"(`31.221.203.21`). Lo que se mantiene intacto es el puerto destino y la IP destino. Esto se debe a que el router aún no ha hecho el destination NAT.

E.4. Server 2 (sede dorayaki)

Finalmente, veamos cómo llegan los paquetes hasta el servidor público:

1	0.000000000	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [SYN] Seq=0 Win=65535 Len=0 MSS=1460
2	0.000065565	10.10.1.2	31.221.203.21	TCP	58	22 → 11648 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3	0.000442250	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.000590031	31.221.203.21	10.10.1.2	SSHv2	78	Client: Protocol (SSH-2.0-libssh2_1.11.1)
5	0.000606162	10.10.1.2	31.221.203.21	TCP	54	22 → 11648 [ACK] Seq=1 Ack=25 Win=64216 Len=0
6	0.024165850	10.10.1.2	31.221.203.21	SSHv2	97	Server: Protocol (SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.14)
7	0.024789625	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=25 Ack=44 Win=65535 Len=0
8	0.027943046	10.10.1.2	31.221.203.21	SSHv2	1174	Server: Key Exchange Init
9	0.028409713	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=25 Ack=1164 Win=65535 Len=0
10	0.367705155	31.221.203.21	10.10.1.2	TCP	1442	11648 → 22 [PSH, ACK] Seq=25 Ack=1164 Win=65535 Len=1388 [TCP segment of a reassembled PDU]
11	0.408871712	10.10.1.2	31.221.203.21	TCP	54	22 → 11648 [ACK] Seq=1164 Ack=1413 Win=65535 Len=0
12	0.442075162	31.221.203.21	10.10.1.2	SSHv2	762	Client: Key Exchange Init
13	0.442129786	10.10.1.2	31.221.203.21	TCP	54	22 → 11648 [ACK] Seq=1164 Ack=2121 Win=65535 Len=0
14	0.447677876	31.221.203.21	10.10.1.2	SSHv2	102	Client: Elliptic Curve Diffie-Hellman Key Exchange Init
15	0.447716069	10.10.1.2	31.221.203.21	TCP	54	22 → 11648 [ACK] Seq=1164 Ack=2169 Win=65535 Len=0
16	0.457396480	10.10.1.2	31.221.203.21	SSHv2	626	Server: Elliptic Curve Diffie-Hellman Key Exchange Reply, New Keys, Encrypted packet (plaintext_len=276), Unknown (157)
17	0.458012900	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2169 Ack=1736 Win=65535 Len=0
18	0.562899462	31.221.203.21	10.10.1.2	SSHv2	70	Client: New Keys
19	0.569533873	31.221.203.21	10.10.1.2	SSHv2	106	Client: Encrypted packet (plaintext_len=36), Unknown (18)
20	0.569633061	10.10.1.2	31.221.203.21	TCP	54	22 → 11648 [ACK] Seq=1736 Ack=2237 Win=65535 Len=0
21	0.569742128	10.10.1.2	31.221.203.21	SSHv2	106	Encrypted packet (plaintext_len=36), Unknown (115)[Malformed Packet]
22	0.570104587	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2237 Ack=1788 Win=65535 Len=0
23	0.688936795	31.221.203.21	10.10.1.2	SSHv2	138	Encrypted packet (plaintext_len=68)[Malformed Packet]
24	0.692230371	10.10.1.2	31.221.203.21	SSHv2	106	Encrypted packet (plaintext_len=36), Unknown (249)[Malformed Packet]
25	0.695093669	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2321 Ack=1840 Win=65535 Len=0
26	0.797426650	31.221.203.21	10.10.1.2	SSHv2	154	Encrypted packet (plaintext_len=84), Unknown (156)[Malformed Packet]
27	0.809011152	10.10.1.2	31.221.203.21	SSHv2	90	Encrypted packet (plaintext_len=20), Channel Failure[Malformed Packet]
28	0.811594599	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2421 Ack=1876 Win=65535 Len=0
29	0.888618701	10.10.1.2	31.221.203.21	SSHv2	682	Server: Encrypted packet (plaintext_len=612), Unknown (117)
30	0.889143899	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2421 Ack=2504 Win=65535 Len=0
31	0.933702950	31.221.203.21	10.10.1.2	SSHv2	106	Encrypted packet (plaintext_len=36), Unknown (192)[Malformed Packet]
32	0.934564937	10.10.1.2	31.221.203.21	SSHv2	106	Encrypted packet (plaintext_len=36), Unknown (14)[Malformed Packet]
33	0.937410552	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2473 Ack=2556 Win=65535 Len=0
34	1.053284361	31.221.203.21	10.10.1.2	SSHv2	138	Encrypted packet (plaintext_len=68), Unknown (198)[Malformed Packet]
35	1.055112623	10.10.1.2	31.221.203.21	SSHv2	90	Encrypted packet (plaintext_len=20), Unknown (219)[Malformed Packet]
36	1.057879198	31.221.203.21	10.10.1.2	TCP	60	11648 → 22 [ACK] Seq=2557 Ack=2592 Win=65535 Len=0
37	1.168672969	31.221.203.21	10.10.1.2	SSHv2	106	Encrypted packet (plaintext_len=36), Unknown (244)[Malformed Packet]
38	1.170633212	10.10.1.2	31.221.203.21	SSHv2	126	Encrypted packet (plaintext_len=20), Unknown (108)[Malformed Packet]

Figura 59: Trazas server 2 (sede dorayaki)

Podemos apreciar cómo aquí ya sí que se ha cambiado el puerto destino y la IP destino. Esto nos indica que el router de la sede Dorayaki ha realizado el destination NAT correctamente y ha encaminado el paquete al servidor correspondiente (y a su IP asignada). Por ende, podemos apreciar la IP destino `10.10.1.2` y el puerto `22` correspondiente al SSH.

F. Servicios en contenedores y K8s

A continuación se muestra un diagrama completo de todos los servicios que han sido dockerizados e incluidos en el clúster de K8s:

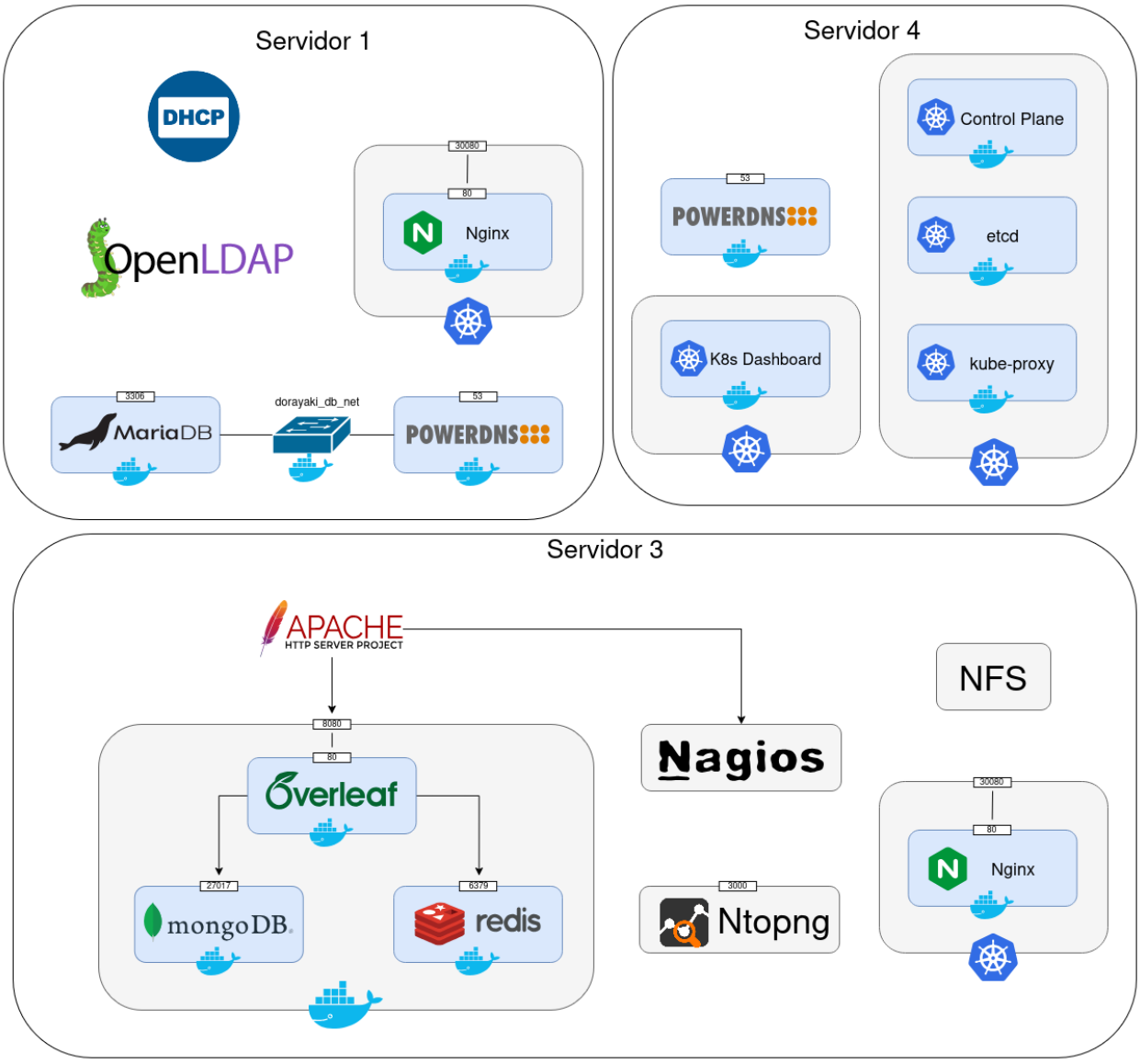


Figura 60: Diagrama de todos los servicios en contenedore y K8s