****

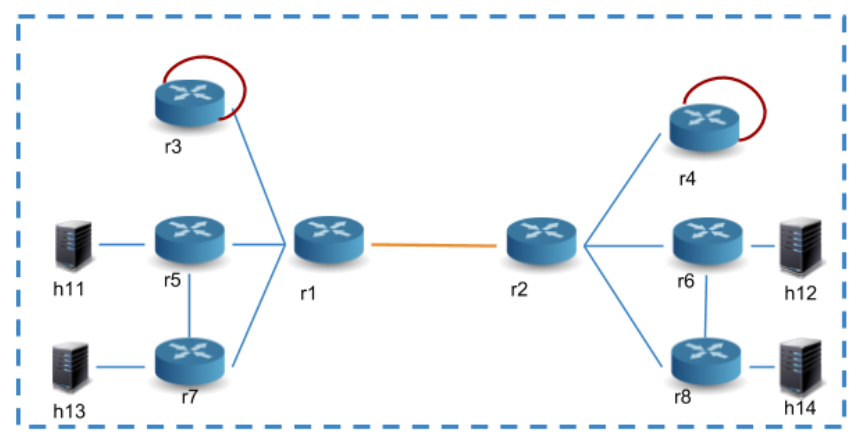
**Redes de Computadoras**

**Práctico 4: Ruteo dinámico.**

**Alumnos:**

* **Heredia, Marco.**
* **Yepez Hinostroza, Franz.**

### Ejercicio 1: Ruteo Dinámico.

****

Podemos ver la asignación de IPs en la siguiente tabla:

|  |  |  |
| --- | --- | --- |
| Nodo | Interfaz compartida con | Dirección IP |
| r1 | r2 | 2001:a:bbbb::2 |
| r1 | r3 | 2001:a:aaaa:1::2 |
| r1 | r5 | 2001:a:aaaa:2::2 |
| r1 | r7 | 2001:a:aaaa:3::2 |
| r3 | r1 | 2001:a:aaaa:1::3 |
| r3 | r3 | 2001:a:aaaa:7::2 |
| r5 | r1 | 2001:a:aaaa:2::3 |
| r5 | r7 | 2001:a:aaaa:4::3 |
| r5 | h11 | 2001:a:aaaa:5::2 |
| r7 | r1 | 2001:a:aaaa:3::3 |
| r7 | r5 | 2001:a:aaaa:4::2 |
| r7 | h13 | 2001:a:aaaa:6::2 |
| h11 | r5 | 2001:a:aaaa:5::3 |
| h13 | r7 | 2001:a:aaaa:6::3 |
| r2 | r1 | 2001:a:bbbb::3 |
| r2 | r4 | 2001:a:bbbb:1::2 |
| r2 | r6 | 2001:a:bbbb:2::2 |
| r2 | r8 | 2001:a:bbbb:3::2 |
| r4 | r2 | 2001:a:bbbb:1::3 |
| r4 | r4 | 2001:a:bbbb:7::2 |
| r6 | r2 | 2001:a:bbbb:2::3 |
| r6 | r8 | 2001:a:bbbb:4::3 |
| r6 | h12 | 2001:a:bbbb:5::2 |
| r8 | r2 | 2001:a:bbbb:3::3 |
| r8 | r6 | 2001:a:bbbb:4::2 |
| r8 | h14 | 2001:a:bbbb:6::2 |
| h12 | r6 | 2001:a:bbbb:5::3 |
| h14 | r8 | 2001:a:bbbb:6::3 |

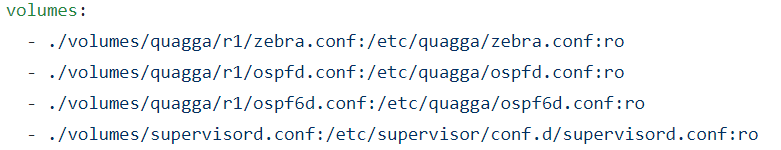
Por empezar, instalamos Docker CE y Docker Compose (herramientas que nos permiten crear y administrar contenedores) y quagga el cual es un software que nos brinda características de enrutamiento, de los cuales para este trabajo se utilizó OSPFv3.

Utilizando los archivos preconfigurados en el repositorio de git:[*https://github.com/maticue/docker\_quagga.git*](https://github.com/maticue/docker_quagga.git)*.*

Configuramos docker para que funcione con IPv6 como indica en el Readme del repositorio, pusimos en marcha la red y luego la fuimos ampliando de acuerdo a lo que pedía la consigna.

5) Analizamos el archivo docker-compose.yml, el cual tiene distintas partes que se analizarán a continuación:

\*Sección *services*: se declaran todos los contenedores que van a crearse (para este caso los routers y hosts). Contiene algunas subsecciones, como **build**, que crea una imagen de dicho contenedor que se va a poder instanciar las veces que sean necesarias. **volumes**, que contiene 3 parámetros: el primero indica la ruta donde se encuentra el archivo de configuración, el segundo indica la ruta del archivo en el cual se guardara la configuracion en el contenedor y el tercero indica los privilegios de lectura/escritura que se tendrán sobre el archivo, los 3 parametros estan separados por ‘:’ .



**image** hace referencia a qué imagen se usará para crear el contenedor. Una imagen contiene todo lo necesario para crear el contenedor particular. **ports**, sirve para comunicarse entre procesos. Cuando el contenedor se inicia se le asigna un número de puerto, el cual va a escuchar el puerto del proceso zebra (2601). Si el usuario quiere acceder a las configuraciones de zebra, mediante telnet deberá accederse al puerto asignado y docker hará la conexión con el puerto 2601. Lo mismo se puede hacer si se quiere acceder al proceso de ospf para IPv6. Una vez hecha la conexión, se podrá configurar el router casi con los mismos comandos que usan los de CISCO. En **networks**, se especifican las redes a las que va a pertenecer un determinado contenedor o router, y las ip (versión 4 y 6) de sus interfaces.

\*Sección *networks*: crea las redes aisladas, independientes de los contenedores, disponibles para que estos las usen.

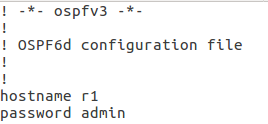
Además de configurar este archivo fue necesario configurar los archivos contenidos en la carpeta /volumes/quagga/ con la información de cada uno de los routers.

5.1) El servicio ospfv3 escucha en el puerto 2606, y para conectarse (desde r1 teniendo en cuenta la configuración por defecto del repositorio), se hace con el comando:

$ *telnet localhost 10013*

6) A cada contenedor se les asocia archivos de configuración. Entre los mas destacados, estan los de ospfv3(ospf6d.conf) , ospf (ospfd.conf) y zebra (zebra.conf). El primero, habilita y configura el protocolo ospfv3, estableciendo costos, area, etc. El segundo, hace lo propio para ospf (IPv4), y el tercero, habilita los enrutamientos para IPv4/6.

6.1) En los 3 archivos nombrados anteriormente se encuentra el password para conectarse por telnet, de acuerdo a cual es el servicio al que nos queremos conectar. En todos los casos la contraseña esta configurada como admin:



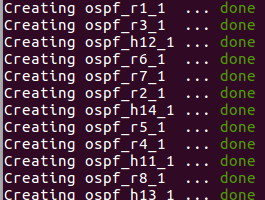
7) Editamos el archivo docker-compose.yml para armar la red propuesta, lo mas importante de esta sección es la configuración de IPs que le dimos a los routers según la tabla de direcciones IP propuesta al comienzo del informe.

8) Para que puedan cargarse las configuraciones de ospf, se crean los archivos necesarios (*ospf6d.conf* y *zebra.conf*) para cada router. Los hosts no necesitan, ya que tienen una ruta por defecto, por donde enviará cualquier paquete cuyo destino no sea una ip propia.

8.1) en *ospf6d.conf,* en la línea con el comando *router ospf6* se esta habilitando el algoritmo de enrutamiento ospfv3, y en *zebra.conf* se habilita el reenvío de paquetes IPv6 con el comando *ipv6 forwarding*.

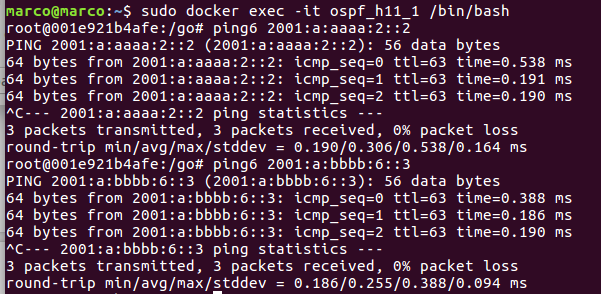
Una vez que todo estuvo configurado, levantamos el entorno docker con el comando:

$ *docker-compose up*

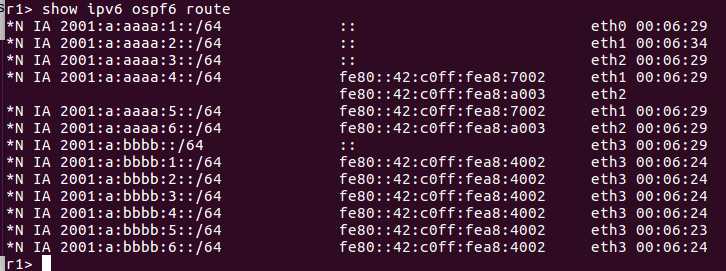


Podemos ver que se creó cada uno de los contenedores.

9)Ahora podemos verificar conectividad entre cada uno de los puntos de la red con el comando ping6. Mostramos alguna captura de esto:



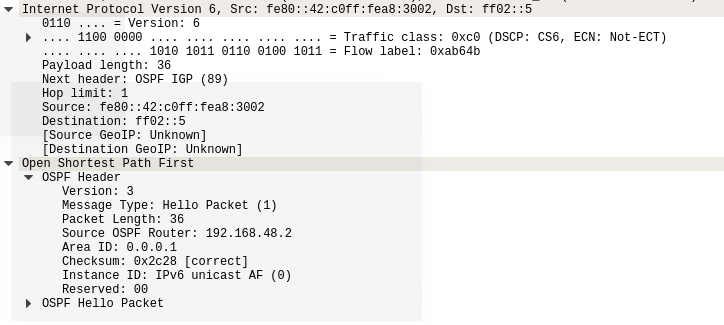
Para ver que las tablas de ruteo muestren las rutas ospf, conectando con telnet a un router se ejecuta el comando *show ipv6 ospf6 route*:

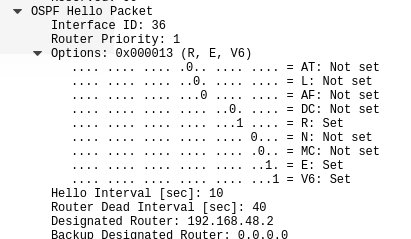


10) Analizamos los paquetes que llegan a uno de los routers con wireshark. Podemos ver que cada 10 segundos llega un mensaje de Hello, ya que está configurado para que se vaya actualizando cada 10 segundos.

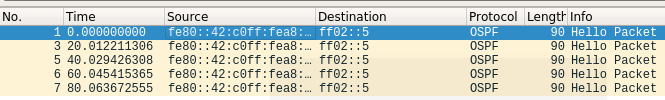


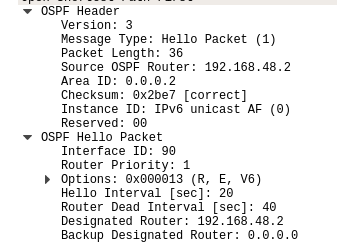
Luego analizando su cabecera podemos ver las IPv6 de destino y origen, el área OSPF al que pertenece y otras configuraciones del protocolo OSPF.





11) Para observar cambios en la configuración OSPF, editamos el archivo ospf6d.conf del router 3 en el cual lo configuramos como un router de área 2 y con un Hello Interval de 20 segundos. Ahora podemos ver estos paquetes en wireshark:





En los cuales vemos que los paquetes llegan cada 20 segundos y en el cual indica que es un router de área 2, pero debido a que funciona como una especie de router de borde sigue habiendo conectividad con el mismo:

