Facultad de Ciencias Exactas, Físicas y Naturales, Laboratorio de Redes y Comunicaciones, Redes de computadoras

Práctico 7: Administración de servicios de red en Linux: Containers, Orquestación en el despliegue de aplicaciones

TROMBOTTO, Agustin Mat:39071116
gutitrombotto@gmail.com
SKRBA, Nikola Mat:17AD22123
skrba.nikola@gmail.com
DEL BOCA, Juan Manuel Mat:37850419
Juanmadelboca@gmail
DI LORENZO, Franco Mat:37619687
francodi65@gmail.com

17 de junio de 2017

Resumen Hoy el día el desarrollo de las conexiones en la Internet estan dando un giro importante. La virtualizacion de las redes a tomado un papel proponderante en las conexiones de las redes. Para ello se utilizan diferentes herramientos y nuevo concepto que ayudan al despligue de redes y a su administracion. La idea es la siguiente: a partir de un hardware establecido, se puede crear grandes conecciones de Internet.

Palabras Clave: Docker, Minikube, Administracion de Red, Orquestacion, DNS, DHCP

1. Introducción

En los TP precedentes, hablábamos principalmente de direcciones IP que forma parte de la capa 3 (red). Una dirección IP permite localizar una maquina. Pero, una maquina puede soportar varias aplicaciones. Por eso, se necesitaba desarrollar un nuevo concepto para distinguir dos aplicaciones en una sola maquina : se llama “puertos de red”. Cada mensajes del expedidor deberán entonces llevar dos números de puertos de red además de los direcciones IP. Ahora, existe dos protocolos de capa 4 : UDP y TCP. En este Trabajo Practico, vamos a analizar los dos protocolos, el primero sobre un servidor DNS y el segundo sobre un servidor http (via Wordpress).

2. Marco Teorico

2.1. ¿Qué son containers? En qué se diferencian con las máquinas virtuales?

Containers y maquinas virtuales (VM) son técnicas de virtualización, es decir que crean una versión virtual de una entidad física. En otras palabras, permiten hacer funcionar varios sistemas operativos o aplicaciones sobra una sola maquina física. Sus beneficios son numerosos, podemos citar : - Optimizar recursos de los servidores =¿reducir costo de material y mejorar rendimiento - Movilidad : se puede desplazar una VM de un servidor físico a otro. - Seguridad : se puede separar servicio que ofrece une servidor (sistema de mensajería, intercambio de ficheros, servidor web, . . .)

Por eso, entendemos el entusiasmo de las empresas por la virtualización.

Para funcionar correctamente, las maquinas virtuales emplean un intermedio entre ellas y el hardware que se llama hipervisor (como Virtual Box), el cual permite asignar de manera pertinente los recursos del hardware (memoria, procesador, . . .). Cada VM imita un servidor con su sistema operativo, sus ficheros, bibliotecas, . . . y la aplicación.

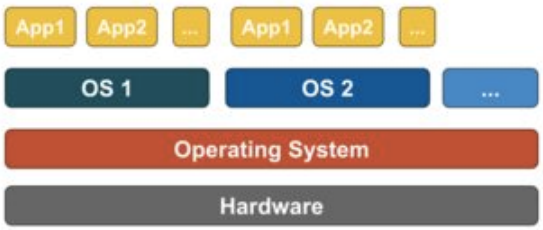

Figura 1: *Hypevisor (MV).*

El problema es que cada iteración del sistema operativo host y de los ficheros puede provocar doblones entre los VM. ¡Eso gasta la memoria del servidor!

La virtualización que usa containers es una método en la cual la capa de virtualización se ejecuta en el seno mismo del sistema operativo, sin emplear los hipervisores. Es decir que comparten un solo sistema operativo (con ficheros, bibliotecas, . . .). Por eso, cada containers incluye solamente la aplicación que maneja y sus ficheros o bibliotecas asociados. Vemos la reducción del pesado de un container que eso implica, con respecto a la VM. Lo que permite instalar 10 a 100 veces mas containers que VM sobre un servidor, iniciar la aplicación con mucho menos tiempo, y transportar la aplicación de servidor en servidor (mas movilidad).

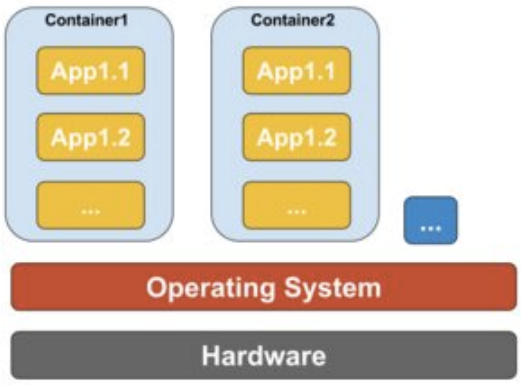

Figura 2: *Container.*

Sin embargo, vemos dos principales desventajas : - menos seguridad : aislamiento menos importante que entre los VM. - Cada containers deben utilizar el mismo OS que de la maquina host. - Anda con Linux (Windows server desde 2016)

En este practico, usamos los containers de tipo Docker. Es el nombre de la empresa que hizo los containers mucho mas manejable.

Para concluir, podemos decir que Docker es una herramienta que puede empaquetar una aplicación (servidor DHCP, DNS, web, o cliente, ...) y sus dependencias en un contenedor virtual que se puede ejecutar en cualquier servidor del mismo tipo.

2.2. ¿Qué es quagga? ¿Qué otros servicios de red ofrece?

Quagga es una suite de software de enrutamiento que implementa los protocolos OSPF, RIP, BGP para los ruters de tipo UNIX (Linux, MacOS). Digamos que es un programa de tipo "demonio" lo que significa que se ejecuta en segundo plano en vez de ser controlado directamente por el usuario. Cada demonio (ospfd, bgpd, ...) deben ser activado y configurado en el ruter. Ademas, soporta las funciones "route maps", "ping", y "traceroute". A diferencia de los ruters Cisco por ejemplo, Quagga no soporta los protocolos DHCP, http, FTP o SSH pero esos protocolos pueden ser activados al nivel de Linux. En definitiva, se puedo comprender como la virtualizacion de un router montado en un linux.

2.3. ¿Qué es kubernetes? ¿Cuál es la diferencia con docker swarm?

Kubernetes y Docker Swarm son « orchestration frameworks ». En otras palabras, manejan la gestión de los containers (de tipo Docker en general). Son plataformas que permiten automatizar el despliegue y los operaciones sobre los containers. Por ejemplo, cuando hay varios containers por un servidor, maneja el uso de RAM, CPU y memoria que necesita cada aplicación. La gran diferencia entre los dos es que Swarm se centra en la facilidad de uso mientras que Kubernetes es mas abierto y flexible. El resultado es que Swarm se instala y se configura mas rápidamente.

2.4. Explicacion de Gitlab

Gitlab es un servicio web de control de versiones y desarrollo de software colaborativo basado en Git. Además de gestor de repositorios, el servicio ofrece también alojamiento de wikis y un sistema de seguimiento de errores, todo ello publicado bajo una Licencia de código abierto. En nuestro caso, creamos un servicio de gitlab dentro de un contenedor en nuestra red. De esta manera, se puede usar el servicio descripto anteriormente dentro de cualquiera de los host en dicha red.

2.5. ¿Por qué es preferible utilizar Ansible cuando las configuraciones se pueden efectuar manualmente ó con un bashscript?

Ansible es un lenguaje de automatización simple que puede describir perfectamente una infraestructura de aplicaciones de TI. Esta es la ventaja principal de este lenguaje. Al permitir automatizar la configuracion de servidores, ya no se requiere ejecutar varios script de bash .ª mano.º configurar una por una las funciones de un servidor. Ademas es fácil de aprender, auto-documentación, y no requiere un grado de grado de grado de informática para leer. La automatización no debe ser más compleja que las tareas que está reemplazando.

3. Procedimiento de Configuracion de la red

3.1. Configuracion de Red con Docker

Se crea una red B la cual se configura y se crea a traves de docker. La arquitectura de la red es mostrada a continuacion:

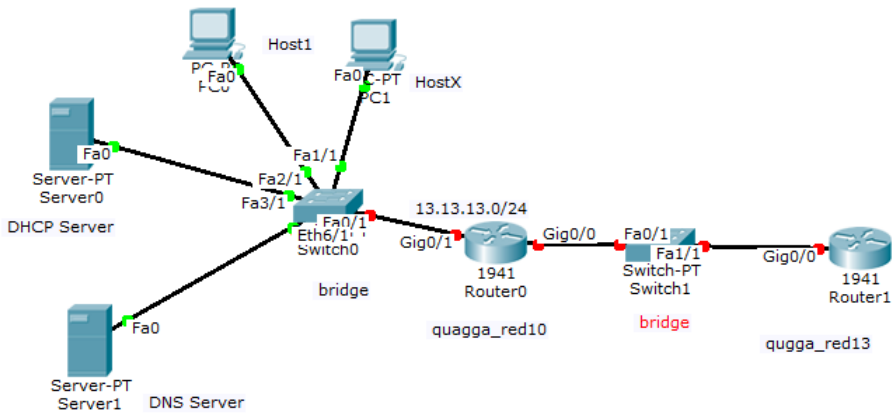

Figura 3: *Topologia de la red.*

Se utiliza un archivo .yml el cual permite autogenerar la red. A traves de docker-compose se çarga.esçarga.ese archivo generando los host, los bridges (switch), y routers (quagga) mostrados en la configuracion. Esto se hace a traves del comando docker-compose up -d. A continuacion se muestra parte de este archivo.

Figura 4: *Archivo de configuracion docker-compose.yml.*

Luego de la configuracion de la red interna, y con el router de borde (quagga red10) se interconectaron con los otros sistemas autonomos compartiendo rutas bgp. Se testeó realizar ping entre los host de los sistemas autonomos de las direntes redes a traves de un switch fisico.

3.2. Configuracion de un servidor con Ansible

La conexion de la red se da a traves de la creacion de maquinas virtuales las cuales se conectan internamente a traves de un bridge. Una VM será el servidor Configuracion, el cual contiene el archivo de configuracion automatica del la otra VM que será el Server (a configurar). Esto se ve a continuacion:

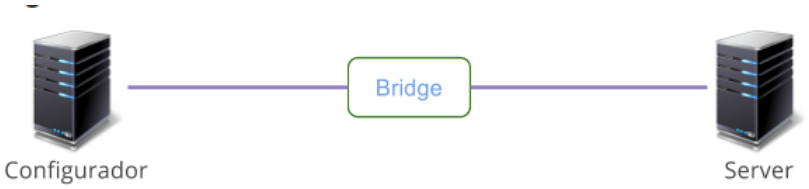

Figura 5: *Topologia de la Red*

Para configurar el servidor utilizando ansible primero procedimos a configurar acceso por medio de ssh sin utilizar contraseña para un usuario root. Esto se logro por medio de una clave de autenticación guardada en ambos extremos, y la modificación de un registro para que permita acceso root de esta manera (por defecto no esta permitido). Luego creamos el archivo yml, el cual el ansible utiliza para configurar el equipo como se ve a continuación:

```yaml
---
- hosts: webservers
  remote_user: root
  become: true
  vars:
   hostname_1: 'cba-efn-ryc-grupoG'
   timezone_1: 'America/Argentina/Cordoba'
   ntp_1: 'time.nist.gov'
  tasks:
    - hostname:
       name: "{{ hostname_1 }}"

    - name: Timezone
      action: command timedatectl set-timezone "{{ timezone_1 }}"

    - user:
        name: james
        shell: /bin/bash
        groups: www-data
        append: yes

    - name: obtengo licencia de nodejs
      apt_key: url=https://deb.nodesource.com/gpgkey/nodesource.gpg.key

    - name: agrego repo para descargar
      apt_repository:
       repo: 'deb https://deb.nodesource.com/node_0.10 {{ ansible_distribution_release }} main'
       update_cache: yes

    - name: Instalo nodejs
      apt: name=nodejs

    - name: NTP
      apt:
       name: ntp
       state: present

    - name: Configuro NTP
      action: command timedatectl set-ntp true

    - name: Seteo direccion 'time.nist.gov'
      copy:
       src: /home/kalantos/Desktop/ntp.conf
       dest: /etc/ntp.conf

    - name: Arranco NTP
      service:
        name=ntp
        state=restarted
```

Figura 6: *Archivo de configuracion ansible.yml*

Finalmente, por medio del comando ansible-playboock ansible.yml, logramos configurar de manera automatica y remota el servidor.

Referencias

[1] Instalación y configuración de servidor DNS: https://www.digitalocean.com/community/tutorials/how-to-configure-bind-as-an-authoritative-only-dns-server-on-ubuntu-14-04

[2] Pagina oficial Docker: https://www.docker.com/

[3] Docker Compose documentation: https://docs.docker.com/compose/compose-file/compose-file-v2/