NGINX Y BALANCEO DE CARGA

Autores

Marcos Álvarez Vidal – UO265180
Aitor Llanos-Irazola – UO264476
Javier Pontón González – UO264003

Índice

NGINX

¿Qué es?

NGINX (creado por Igor Sysoev) es servidor HTTP de gran potencia, así como servidor proxy e inverso, con capacidad de servidor proxy de email. NGINX es muy reconocido por su eficiencia, ya que surgió para resolver el problema *C10K* (Dan Kegel, 1999), que consistía en lograr optimizar una red para poder soportar un enorme número de clientes al mismo tiempo (remarquemos que se trata de clientes o conexiones concurrentes, concepto diferente a conexiones por unidad de tiempo, ya que en el primero la solución se basa en una gestión eficiente de las mismas, como hace NGINX, cuando en el segundo es procesarlas más rápidamente).

NGINX se basa más en una arquitectura muy escalable guiada por eventos asíncronos más que en el uso de múltiples hilos, una de las bases para solucionar el problema anterior.

Este problema es muy importante, ya que hoy en día las grandes compañías afrontan cargas mucho mayores que la propuesta del problema mencionado, mismamente pueden llegar a registrar millones de conexiones simultáneas en un mismo nodo sin ir muy lejos.

Todo esto lo consigue usando además una cantidad ínfima y predecible de memoria, siendo perfectamente apto tanto para pequeños servidores como para grandes clústeres de servidores.

Respecto a sus alternativas y uso, destacaremos Apache, al cual en los últimos años está comiendo terreno llegando al punto de ser superado según ciertas estadísticas, pasando de un 60% aproximadamente en 2011, a menos de 30% en 2019, mientras que al mismo tiempo NGINX crece de la nada hasta más de un 30% en las mismas fechas según W3Techs. Es usado en grandes empresas como Netflix, NASA, WordPress… así como en las páginas webs con mayor importancia, y en aumento. Se muestra una tendencia a ser más usado que Apache en sitios web clasificados como TOP, posiblemente debido a su eficiencia y los requisitos de dichas páginas.

Usos

Proxy inverso

Generalmente, un servidor proxy es una interfaz de comunicación en una red que se encarga de las solicitudes y las delega a una computadora de destino, es decir, es el intermediario de la comunicación entre dos máquinas.

Esta estructura se utiliza en redes corporativas para que los dispositivos del cliente puedan controlar el acceso a Internet. En este caso, el servidor configurado como proxy se muestra como la única forma de conectarse a la red pública.

Un servidor de reenvío canaliza todas las solicitudes de la red interna y las transmite con su propia dirección de reenvío al servidor de destino en Internet. Las respuestas del servidor generalmente llegan al proxy antes de que se distribuyan a los distintos dispositivos cliente que permanecen en el anonimato, a menos que sea un proxy transparente.

Mientras que un proxy de reenvío protege los dispositivos cliente, un proxy inverso trabaja de tal manera que proteja a la inversa, es decir, protege a la parte del servidor. Lo que se suele hacer es tener uno o más servidores web que se encarguen de las solicitudes procedentes de internet y las transmita a otro servidor, el cual queremos proteger o que se desconozca. (IONOS, 2020)

Al igual que Varnish, NGINX puede realizar el almacenamiento en caché web, pero muchos administradores desconocen esta faceta. NGINX puede proporcionar contenido estático directamente de una manera muy eficiente y puede actuar como un caché front-end como

Varnish. Aunque la única tarea de Varnish es actuar como caché con opciones avanzadas, NGINX es versátil y nos brinda múltiples opciones.

La decisión de los administradores de sistemas suele estar entre estas dos, pero por lo general no saben cuál escoger, la decisión de usar NGINX solo no es introducir nuevas tecnologías para aprender y mantener en nuestra infraestructura, y parece que la configuración de NGINX es mejor que la configuración de Varnish. (Adrianalonso.es, 2017).

NGINX posibilita actuar de caché, puede servir contenido estático de una manera muy eficiente, y su instalación en nuestro servidor no es complicada.

Para más información sobre su instalación consultar la siguiente fuente: https://www.sololinux.es/configurar-nginx-como-proxy-inverso/

Arquitectura

NGINX funciona en base a las especificaciones de hardware que tenga disponibles, y usa cuatro tipos de procesos:

- Proceso maestro, que realiza operaciones privilegiadas como acceso a configuraciones y puertos o la creación de los siguientes subprocesos.
- Proceso de cargado de caché, iniciado al principio y con el único propósito de cargar la caché del disco a memoria, tras ello finaliza, y su consumo es muy bajo.
- Proceso de gestión de caché, que se ejecuta periódicamente y *recorta* las entradas de la cache para mantenerlas dentro de los tamaños configurados
- Los *worker processes*, como su nombre indica, son los que realizan el trabajo principal, gestionando las conexiones, tareas de I/O, y comunicación con servidores.
  Generalmente (y como NGINX recomienda) estos se ejecutan cada uno de manera independiente (y *single threaded*) en un núcleo diferente de la CPU para así usar más eficientemente los recursos disponibles.
  La clave de la eficiencia de estos se basa también en su constante funcionamiento, de manera que se evitan los cambios de contexto, ahorrando tiempo. Y el uso de memoria compartida en caso de ser necesario.

Cada proceso worker es creado como se ha dicho antes por el maestro, proveyéndolos de los sockets necesarios para escuchar las peticiones, tras ello, éstos esperan a eventos, iniciados por conexiones entrantes y las envían a como NGINX las llama, máquinas de estado, un concepto usado en la mayoría de los servidores web, y que se encargan de gestionar estas peticiones.

La implementación de esta máquina de estados difiere de la manera común de hacerlo, generalmente se tiende a gestionar cada transacción HTTP en un hilo diferente, resultando en que estos hilos están gran parte del tiempo bloqueados con la espera de respuesta del cliente. Esto es, desde el punto de vista de la eficiencia, muy malo, porque, aunque sea fácil de extender e implementar, el sistema operativo tendrá que mapear cada una de estas conexiones a sus respectivos procesos/hilos, un gran desperdicio de memoria, y del tiempo que llevan las tareas relacionadas a mantener esto.

NGINX nombra a esta implementación como máquina de estados bloqueante, y para solventar lo anterior han realizado una implementación de una máquina de estados no bloqueante, con una arquitectura guiada por eventos.

Cada proceso worker espera a eventos escuchando en los sockets creados inicialmente, cuando ocurre un evento en uno de ellos, el worker crea una conexión, que pasará a ser gestionada, para tras ello, dar paso a la gestión de otra de las conexiones pendientes o escuchar a otras nuevas. Consiguiendo que los workers nunca estén bloqueados esperando a la acción de la otra parte.

Gracias a esto se consigue una capacidad de escalamiento mucho mayor pudiendo hacer frente a cientos de miles de peticiones HTTP por worker, así como a picos de tráfico sin problema.

Módulos

NGINX tiene una gran variedad de módulos para ampliar sus funcionalidades, pero muchas de estas no vienen por defecto al instalarlo. Algunos de los módulos más interesantes son:

NGINX PageSpeed

Es un módulo de código abierto del servidor web NGINX. Este módulo optimiza automáticamente las páginas web para mejorar el rendimiento, a la vez que utiliza servidores http para proporcionar páginas web.

Tiene varios filtros que pueden optimizar automáticamente los archivos, tales como HTML, CSS, JavaScript... NGINX PageSpeed se basa en la biblioteca de optimización de PageSpeed, que se ha implementado en más de 100,000 sitios web y es proporcionada por los proveedores de servicios de alojamiento y CDN más populares como GoDaddy. Proporciona más de 40 filtros optimizados, que incluyen: Optimizar imágenes, concatenación de CSS y JavaScript, la expansión de caché y muchos más. (Linuxenespañol, 2018)

Módulo de idioma

Este módulo analiza el encabezado y proporciona la configuración que vaya más acorde con la región del usuario a partir de una lista de configuraciones compatibles. Este módulo es muy útil si queremos proporcionar varios idiomas a nuestra página desde el caché de una forma correcta.

Upstram Consistent Hash

Es un anillo hash interno consistente para seleccionar el balanceador de carga del nodo inferior derecho. Está diseñado para ser compatible con el módulo php-memcache. Esto significa que puede usar el módulo php-memcache para almacenar el valor en el clúster de caché, y luego NGINX puede encontrar el valor en el clúster y leerlo desde allí. Este módulo solo garantiza su correcto funcionamiento para las versiones 0.7.61 y 0.6.34.

HTTP Memc

Es una versión estándar del módulo “memcached estandar” que además soporta los comandos set, add, y delete entre otros muchos más.

Permite definir una interfaz REST personalizada para sus servidores de una manera eficiente desde NGINX mediante solicitudes independientes.

Implementa el protocolo TCP memcached por sí mismo, basado en el mecanismo upstream. Los flujos de entrada/salida no se bloquean. El módulo en sí no mantiene las conexiones TCP a los servidores memcached ascendentes a través de solicitudes, al igual que otros módulos ascendentes.

OpenSSL Status

Cada proceso de trabajo agrega sus valores SSL a la zona de memoria compartida, desde esta memoria se pueden ver por solicitud http los valores. Es posible recopilar estadísticas

independientes para un servidor virtual o asignar múltiples servidores virtuales a una zona y obtener valores resumidos para ellos.

Un servidor guarda sus estadísticas SSL en una zona definida por la opción *ssl_status_zone* y se puede acceder a las estadísticas desde una ubicación marcada por la opción *ssl_status*, un ejemplo de configuración de la anterior puede ser este:

```
server {
    server_name A;
    ...
    ssl_status_zone zona1;
}

server {
    server_name B;
    ...
    ssl_status_zone zona1;
}

server {
    server_name C;
    ...
    ssl_status_zone zona2;

    location /stat1 {
        ssl_status zona1;
    }

    location /stat2 {
        ssl_status zona2;
    }
}
```

Las estadísticas de los servidores A y B estarán disponibles en la "/stat1" del servidor C, y las estadísticas del servidor C estarán disponibles en la "/stat2" del servidor C. Las estadísticas que no tengan la opción de "ssl_status_zone" no serán recolectadas.

NGINX con PHP-FPM

PHP-FPM hace referencia a "administrador de procesos PHP". Es una interfaz que está programada para funcionar entre el servidor web y los programas de servicio de contenido dinámico. Escucha en un determinado puerto pasando la solicitud entre PHP y el servidor web.

Este módulo consume mucha menos memoria al procesar código PHP y además funciona independientemente del servidor web.

(NGINX, 2020)

Ejemplo de uso

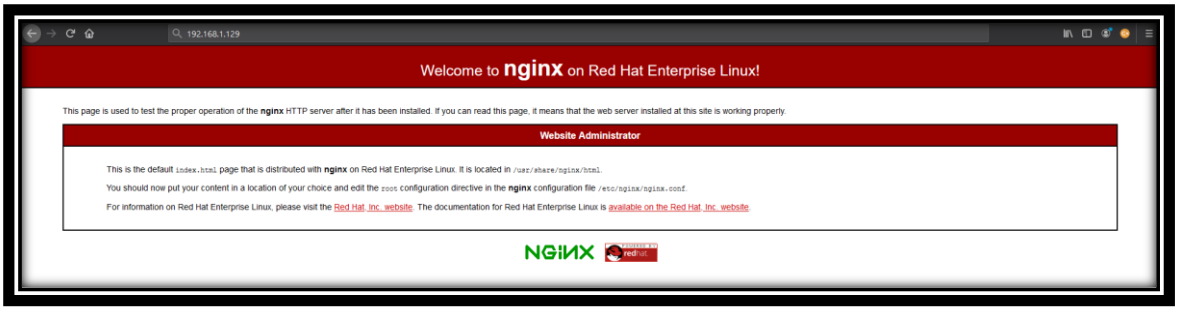
Vamos a configurar NGINX para poder una página web en un servidor con CentOS 8.

Para ello lo primero será instalar NGINX: $ sudo yum install nginx

Una vez instalado vamos a iniciarlo con el siguiente comando: $ sudo systemctl enable nginx y a continuación: $ sudo systemctl start nginx.

Por defecto los puertos son el 80 y el 443, entonces vamos a decirle al firewall que permita las conexiones a esos puertos: $ sudo firewall-cmd --permanent --zone=public --add-service=http --add-service=https.

Si accedemos a la IP local del servidor (en mi caso 192.168.1.129) obtendremos la siguiente página:



Ahora que ya está la regla añadida al firewall, vamos a crear una carpeta en "/var/www" que la voy a llamar "sktjpg.com" con la siguiente instrucción: $ sudo mkdir -p /var/www/sktjpg.com y le damos los permisos adecuados a la carpeta con: $ sudo chmod -R /var/www/sktjpg.com y ejecutamos el siguiente comando para que la carpeta pertenezca al usuario actual: $ sudo chown -R $USER:$USER /var/www/sktjpg.com y creamos la carpeta "public_html": $ sudo mkdir -p /var/www/public_html, dentro de este directorio vamos a crear un "index.html" con el siguiente comando: $ sudo nano /var/www/sktjpg.com/index.html y creamos cualquier HTML, por ejemplo:

```
<html>
  <head>
    <title>sktjpg</title>
  </head>
  <body>
    <h1>Esto es la web de sktjpg</h1>
  </body>
</html>
```

A continuación, vamos a crear nuestro archivo virtual host para que NGINX pueda gestionarlo. Para ello crearemos dos carpetas en las que se suelen estructurar todos los archivos de configuración para tener los ficheros bien ordenados y cuando haya que hacer cambios poder acceder a ellos fácilmente.

Crearemos dos carpetas, sites-availables que contendrá los archivos de tipo configuración de los hosts virtuales disponibles en el servidor, y sites-enabled que serán aquellos hosts que están habilitados. Los comandos para crear las dos carpetas son: $ sudo mkdir /etc/nginx/sites-available y $ sudo mkdir /etc/nginx/sites-enabled.

Después de esto tenemos que indicar a NGINX que los archivos de configuración estarán en sites-enabled, para ello modificaremos el archivo de configuración con el comando: $ sudo nano /etc/nginx/nginx.conf y dentro del bloque http:

```
http {…

        include /etc/nginx/sites-enabled/*.conf;
        server_names_hash_bucket_size 64;
```

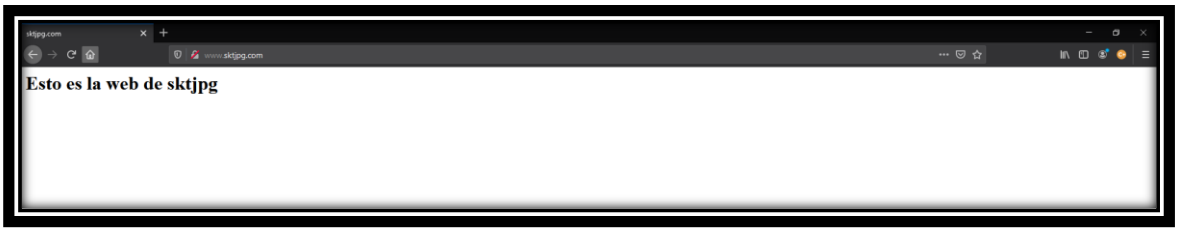
```
}
```

Comprobamos con la orden $ sudo nginx -t que todo lo modificado es válido y no da problemas, su salida debería ser un “ok”.

Ahora vamos a crear el archivo de configuración para la página web, para ello lo mejor es copiar el archivo por defecto y editarlo así nos evitamos escribir y equivocarnos: $ sudo cp nginx.conf.default sites-available/sktjpg.com.conf, una vez creado vamos a editarlo. Modificaremos el “server_name”, en mi caso server name sktjpg.com www.sktjpg.com;, modificaremos el “root”, root /var/www/sktjpg.com/public_html;, también añadimos la sentencia try_files $uri $uri/ = 404; para que las solicitudes no reconocidas nos del error 404 y por último añadimos a la sentencia listen lo siguiente “80 default_server”, es importante quedarse solo con el bloque de “server {}”.

Modificamos los hosts: $ sudo nano /etc/hosts y añadimos lo siguiente 85.49.124.66:80 sktjpg.com para poder entrar con ese nombre a la página

Habilitamos el archivo de configuración con el siguiente comando: $sudo ln -s /etc/nginx/sites-available/sktjpg.com.conf /etc/nginx/sites-enabled/sktjpg.com.conf (con esta instrucción lo que cambiemos en la carpeta sites-available se refleja en la otra, es decir, se encarga de modificar lo que modificamos) reiniciamos NGINX con el comando $sudo systemctl restart nginx y si todo va bien nos mostrará el index.html que creamos antes:



Ventajas

NGINX ofrece numerosas funcionalidades y pros, alguna de sus ventajas más destacables son las siguientes:

- NGINX es un excelente proxy inverso además de ser un buen servidor web ligero y con un alto rendimiento.
- Consume muy pocos recursos al servir contenido estático, gracias a esto le hace ser un buen balanceador de carga o su opción para proxy inverso.
- Permite responder a una gran cantidad de peticiones por segundo aprovechando al máximo todos los hilos y los núcleos que posea nuestro servidor, siendo nuestra configuración mínima.
- NGINX es una solución mucho más optimizada que otros servidores web como Apache, ya que siempre sirve el contenido de una manera optimizada, además si lo combinamos con el módulo.
- NGINX no tiene nada que envidiar a otros servicios para cache, ya que se comporta de una manera muy buena al cachear contenidos web.
- Tiene una versión premium llamada NGINX Plus que añade algunas funcionalidades a la versión de código abierto (OSS), entre ellas destaca la capacidad de ofrecer live streaming mediante HTTP.

- NGINX con cPanel es uno de los paneles de hosting más utilizados para los servidores web, pero en su búsqueda continua en la estabilidad y compatibilidad total no implementaron aun un soporte nativo para servidor web ni para el proxy inverso.
- WordPress es compatible con NGINX.

(Raiolanetworks, 2014)

Balanceadores de carga

¿Qué es un balanceador de carga?

Un Balanceador de carga fundamentalmente es un dispositivo de hardware o software que se pone al frente de un conjunto de servidores que atienden una aplicación y, tal como su nombre lo indica, asigna o balancea las solicitudes que llegan de los clientes a los servidores usando algún algoritmo (desde un simple Round-Robin hasta algoritmos más sofisticados). Resumidamente, es un método para distribuir la carga de trabajo en varias computadoras separadas o agrupadas en un clúster de servidores, pueden ser soluciones hardware, tales como routers y switches que incluyen software de balanceo de carga preparado para ello, o soluciones software que se instalan en el backend de los servidores.

Existen ciertos criterios que debe superar un balanceador de carga para que se considere exitoso, como minimizar tiempos de respuesta, mejorar el desempeño del servicio y evitar la saturación del tráfico de red.

Tipos

Pueden existir dos tipos de balanceadores de carga: Balanceador de carga de aplicaciones o balanceadores de carga de red.

Balanceador de carga de aplicación

Un Balanceador de carga de aplicaciones toma decisiones de direccionamiento en la capa de aplicación (HTTP/HTTPS), admite el direccionamiento basado en rutas y puede direccionar las solicitudes a uno o varios puertos de cada instancia de contenedor del clúster. Un Balanceador de carga de aplicaciones admite el mapeo de puertos de host dinámico

Por ejemplo, si la definición de contenedor de su tarea específica el puerto 80 para un puerto de contenedor NGINX y el puerto 0 para el puerto de host, el puerto de host se elige dinámicamente en el rango de puertos efímeros de la instancia de contenedor. Cuando se inicia la tarea, el contenedor NGINX se registra en el Balanceador de carga de aplicaciones como una combinación de ID de instancia y puerto y el tráfico se distribuye al ID de instancia y al puerto correspondiente a dicho contenedor. Este mapeo dinámico le permite tener varias tareas desde un servicio único en la misma instancia de contenedor.

Balanceador de carga de red

Un Balanceador de carga de red toma las decisiones de direccionamiento en la capa de transporte (TCP/SSL). Puede atender millones de solicitudes por segundo. Una vez que el balanceador de carga ha recibido una conexión, selecciona un destino del grupo de destinos para la regla predeterminada por medio de un algoritmo. Intenta abrir una conexión TCP con el destino seleccionado en el puerto especificado en la configuración del agente de escucha. Reenvía la solicitud sin modificar los encabezados. Un Balanceador de carga de red admite el mapeo de puertos de host dinámico.

Por ejemplo, si la definición de contenedor de su tarea específica el puerto 80 para un puerto de contenedor NGINX y el puerto 0 para el puerto de host, el puerto de host se elige dinámicamente en el rango de puertos efímeros de la instancia de contenedor. Cuando se inicia la tarea, el contenedor NGINX se registra en el Balanceador de carga de red como una combinación de ID de instancia y puerto y el tráfico se distribuye al ID de instancia y al puerto correspondiente a dicho contenedor. Este mapeo dinámico le permite tener varias tareas desde un servicio único en la misma instancia de contenedor.

Balanceo de carga basado en DNS

El servicio DNS trata de resolver los nombres de dominio y traducirlos a direccionamiento IP, facilitando el acceso a los recursos a través de su nombre en lugar de tener que recordar la IP.

En cambio, un servicio tan "*simple*", y a la vez tan importante, como el DNS no sólo realiza funciones de traducción de nombre a IP, sino que también tiene otras cualidades como resolución inversa de DNS, muy útil para la detección de servidores de Spam, y balanceo de carga. Esta última característica nos permite distribuir la carga de nuestros servidores e incluso realizar Geobalanceo (como hacen las herramientas de Cloud como Google o Amazon) basándonos en la dirección IP origen de las solicitudes.

Una vez que el usuario realice la petición DNS sobre el servicio balanceado, el servidor DNS devolverá tantos registros como haya definido en la zona.

Una vez recibida la respuesta DNS, el sistema operativo del usuario elegirá una IP de entre las disponibles para acceder al servicio web. Algunos sistemas operativos elegirán la primera IP, mientras que otros como Windows dependerá de si tienen la pila IPv6 instalada, además de la máscara del cliente. En cualquier caso, si el recurso solicitado es una página web, la mayoría de los navegadores actuales son capaces de detectar de si el recurso es alcanzable, si no lo está utilizará la siguiente IP de la lista entregada por el servidor DNS.

Balanceo de carga centralizado

Como en una relación Maestro/Esclavo, se basa en un nodo maestro que conoce las tareas a realizar y quienes pueden hacerlas (esclavos). Éste envía las tareas a los diferentes nodos esclavos, en cuanto finalizan envían un mensaje al maestro indicando que pueden recibir una nueva.

El algoritmo más recomendado con esta técnica es enviar primero las tareas más costosas, ya que, en caso de no ser así, una vez completadas las pequeñas, los nodos esclavos podrían estar sin hacer nada mientras se completa la más costosa.

En casos donde todas las tareas fueran iguales lo ideal sería una cola FIFO (First In First Out). Podrían darse casos más complejos en los que habría que estudiar más posibilidades.

Balanceo de carga distribuido

En ese tipo de balanceo de carga contamos con múltiples maestros evitando el problema de que un maestro no pueda enviar más de una tarea simultáneamente, ya que podría producir colisiones.

Sería ideal usar este tipo en casos donde haya gran número de tareas pequeñas, ya que los maestros se verán atacados por un gran número de peticiones simultaneas.

Balanceo de Carga en Kubernetes

Para comenzar con la creación de un balanceador de cargas primero debemos crear un clúster en Google Cloud con el IP de alias habilitadas (es una configuración especial de la red de un servidor que permite asociar varias direcciones IP a una única interfaz de red.), para ello usamos el comando:

```
gcloud container clusters create neg-demo-cluster --enable-ip-alias --create-subnetwork="" --network=default --zone=eu-central1-a
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: neg-demo-app # Label for the Deployme
  name: neg-demo-app # Name of Deployment
spec:
  minReadySeconds: 60 # Number of seconds to w
  selector:
    matchLabels:
      run: neg-demo-app
  template: # Pod template
    metadata:
      labels:
        run: neg-demo-app # Labels Pods from t
    spec: # Pod specification; each Pod create
      containers:
      - image: k8s.gcr.io/serve_hostname:v1.4
        name: hostname # Container name
      # Note: The following line is necessary
      # For details, see https://cloud.google.
      terminationGracePeriodSeconds: 60 # Numb
```

Posteriormente procedemos a crear una carga de trabajo en el clúster, en nuestro caso vamos a ejecutar una única instancia de un servidor HTTP con contenedores.

En esta implementación, cada contenedor ejecuta un servidor HTTP. El servidor HTTP solo muestra el nombre del host del servidor de la aplicación (el nombre del pod en el que se ejecuta el servidor) como respuesta.

Para crear la implementación, ejecutamos *kubectl apply -f <file.name>.yaml* siendo file.name el nombre con el que hemos guardado el fichero anterior, en nuestro caso como se muestra en el name: neg-demo-app es ese el nombre.

Una vez creada una implementación, tienes que agrupar sus pods en un servicio. Aquí mostramos un servicio de muestra muy básico que crea el balanceador de cargas.

La anotación del servicio, *cloud.google.com/neg: '{"ingress": true}'* habilita el balanceo de cargas nativo del contenedor. Sin embargo, el balanceador de cargas no se creará hasta que se cree un Ingress (un objeto de la API Google Cloud para gestionar accesos externos a los servicios del clúster, habitualmente HTTP), para crear el servicio aplicamos *kubectl apply -f <file.service>.yaml.*

```yaml
apiVersion: v1
    kind: Service
    metadata:
      name: neg-demo-svc # Name of Service
      annotations:
        cloud.google.com/neg: '{"ingress": true}'
    spec: # Service's specification
      type: NodePort
      selector:
        run: neg-demo-app # Selects Pods labelled
      ports:
      - port: 80 # Service's port
        protocol: TCP
        targetPort: 9376
```

Finalmente, para que el balanceador de carga sea efectivo, como decimos en el párrafo anterior, creamos el objeto de la Api denominado Ingress.

Una vez creado el Ingress, se crea un balanceador de cargas de HTTP(S) en el proyecto y se crean NEG en cada zona en la que se ejecuta el clúster. Los extremos en el NEG y los del servicio se mantienen sincronizados. Por ultimo para aplicar el Ingress ejecutamos *kubectl -f <file.ingress>.yaml.*

```yaml
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: neg-demo-ing
spec:
  backend:
    serviceName: neg-demo-svc # Name of the Service targeted by the Ingress
    servicePort: 80 # Should match the port used by the Service
```

Ejemplos de balanceadores de carga

LVS

LVS (Linux Virtual Server) es una herramienta que nos permite gestionar balance de carga en sistemas Linux. Es una solución de código abierto que busca desarrollar un servidor Linux de alto rendimiento que proporcione buena escalabilidad, confiabilidad y robustez usando tecnología clustering. Ofrece balance de carga en la L4 (TCP/UDP) mediante chequeos dinámicos y adaptativos que manejan el pool de carga dependiendo de la carga de sus componentes. Es posible obtener LVS de los repositorios de nuestra distribución Linux (ipvsadm package), aunque en los últimos kernels suele venir ya integrada.

Vamos a mostrar un ejemplo rápido de uso con LVS:

Los siguientes comandos configuran un servidor linux para distribuir peticiones entrantes dirigidas a 207.175.44.110:80 entre una granja de 5 servidores (192.168.10.x).

```
ipvsadm -A -t 207.175.44.110:80 -s rr
```

Con -A le indicamos a LVS que queremos añadir un servicio nuevo, con -t que ese servicio va a ser del tipo TCP, -s va a ser el modo de balanceo de carga, en este caso rr, Round-Robin, de manera que las peticiones se distribuyan de forma igual por los servidores. Con esta instrucción hemos creado un servicio, de tal manera que todas las peticiones que lleguen a 207.175.44.110 puerto TCP 80 se balanceen de forma equitativa entre los servidores.

Ahora nos disponemos a crear los servidores.

```
ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.1:80 -m
ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.2:80 -m
ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.3:80 -m
 ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.4:80 -m
ipvsadm -a -t 207.175.44.110:80 -r 192.168.10.5:80 -m
```

Con el parámetro -a , señalamos que queremos agregar un nuevo servidor a nuestro servicio de balance carga que viene determinado por ip:puerto (207.175.44.110:80), con el parámetro -r indicamos la IP de los servidores de nuestra granja (entre los que se balanceará el servicio) y con el parámetro -m, decimos que use como método "masquerading", es decir NAT (las peticiones que lleguen a 207.175.44.110:80 serán traducidas a alguna IP de nuestros servidores).

Ultra Monkey

Ultra Monkey permite crear servicios de red con balanceo de carga y alta disponibilidad. Puede trabajar con clústeres pequeños de dos nodos hasta grandes sistemas.

El sistema es bastante sencillo y formado por software que a los lectores habituales ya les resultará familiar:

- Balanceo de carga rápido con The Linux Virtual Server.
- Alta disponibilidad gracias a Linux-HA-Framework (*Heartbeat*).
- Monitorización a nivel de servicio con ldirectord.
- Soporta topologías de alta disponibilidad y/o balanceo de carga con ejemplos de configuración.
- Expansible con facilidad a un gran numero de servicios virtuales basados en IP con fwmarks.
- Paquetes para Debian Sarge y Red Hat Enterprise Linux 3.
- Todo el código es Open Source.

Como vemos para el balanceo de carga se realiza con el software LVS explicado previamente.

Pirhana

Consiste en un balanceador de carga únicamente disponible en RedHat caracterizado por tener una interfaz web, ser de fácil configuración, contar con una documentación extensa y con failover

protection y además soporta varios algoritmos. Su instalación como cualquier programa de RedHat está disponible con *yum install pirhana*.

Referencias

Adrianalonso.es (2017) Acelera tu web con NGINX como caché de proxy inverso. Recuperado: https://adrianalonso.es/devops/acelera-tu-web-con-nginx-como-cache-de-proxy-inverso/

IONOS (2020) Servidor proxy inverso: componente central de la seguridad. Recuperado: https://www.ionos.es/digitalguide/servidores/know-how/que-es-un-servidor-proxy-inverso/

Linuxenespañol (2018) Cómo implementar el modulo PageSpeed en NGINX. Recuperado: https://www.xn--linuxenespaol-skb.com/tutoriales/como-implementar-el-modulo-pagespeed-en-nginx/

NGINX Wiki (2020) NGINX 3 Party Modules. Recuperado: https://www.nginx.com/resources/wiki/modules/

Phoenixnap, Cómo confiurar el host virtual NGINX (bloques de servidor) en centos7. Recuperado:https://phoenixnap.com/kb/how-to-set-up-nginx-server-blocks-virtual-hosts-centos-7

Raiolanetworks (2014) NGINX: ¿Qué ventajas e inconvenientes tiene? Recuperado: https://raiolanetworks.es/blog/nginx/

Sysadminsdecuba (2018) Compilando NGINX con módulos adicionales. Recuperado: https://www.sysadminsdecuba.com/2018/11/compilando-nginx-con-modulos-adicionales/

NGINX Wiki (2020) Introducción. Recuperado: https://www.nginx.com/resources/wiki/

Kegel.com (2014) The C10K problem. Recuperado: http://www.kegel.com/c10k.html

NGINX Wiki (2015) How We Designed for Performance & Scale. https://www.nginx.com/blog/inside-nginx-how-we-designed-for-performance-scale/

Balanceadores de Carga

https://docs.aws.amazon.com/es_es/AmazonECS/latest/userguide/load-balancer-types.html

Balanceo de Carga en Kubernetes

https://cloud.google.com/kubernetes-engine/docs/how-to/container-native-load-balancing?hl=es-419

Balanceo de Carga LVS

https://soporte.itlinux.cl/hc/es/articles/200121888-5-Balanceo-de-Carga-con-LVS

Ultra Monkey Balanceador de Carga

http://bytecoders.net/content/ultra-monkey-alta-disponibilidad-y-balanceo-de-carga.html

Piranha Balanceador de Carga

https://helloit.es/2012/12/balanceador-de-carga-con-lvs-y-piranha/