Trabajo Administración de Sistema - Kubernetes Parte 3

Hayk Kocharyan
757715@unizar.es

21 de junio de 2020

Índice

1. Resumen

En esta tercera parte del proyecto, se exige principalmente un despliegue de Kubernetes que sea capaz de ofrecer un servicio con tolerancia de fallo de **un** maestro. Adicionalmente se exige la puesta en marcha de un sistema de monitorización y alertas del cluster de Kubernetes con Prometheus y un sistema de Dashboard, todo ello mediante la herramienta de gestión de aplicaciones en k8s, Helm.

2. Arquitectura Y Despliegue

2.1. Configuracion del cluster

Como introducción vamos a resumir el esquema de despliegue que se ha seguido. En primer lugar, ha sido necesario modificar por completo los ficheros de *Vagrant* y *Provision.sh*, esto se ha llevado a cabo para poder realizar de manera automática el despliegue del sistema, sin ninguna interacción humana para la configuración.

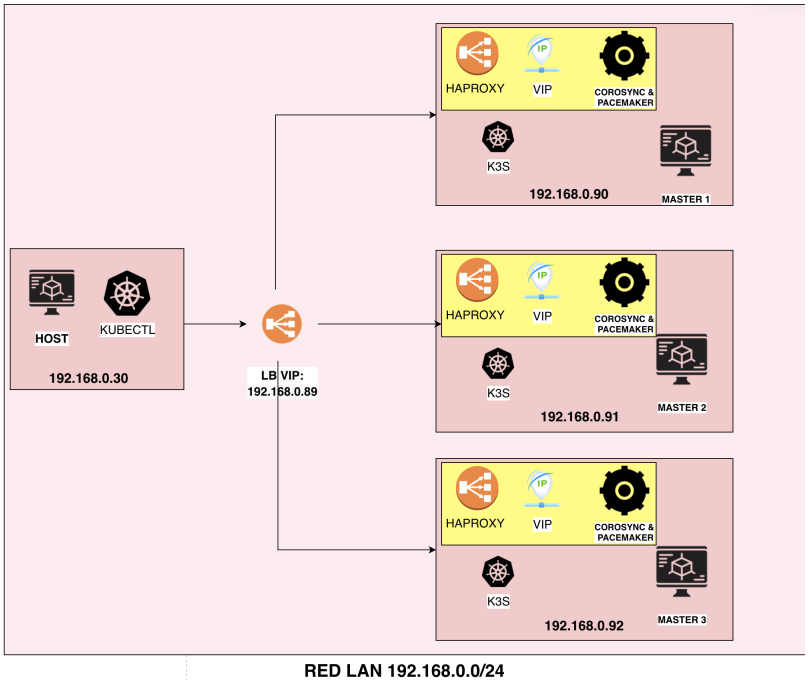

Figura 1: Depliegue de los maestros

Como vemos en la figura 1, se dispone de tres maestros. Se han desplegado 3 maestros ya que DQLite usa el algoritmo de Raft para elegir al líder entre los maestros.
En cuanto a la configuración, los 3 nodos mastros corren en ellos *pacemaker*, *corosync* y *pcs*, además del ya usado *K3s*[2]. Para los 3 nodos maestros, se ha configurado en primer lugar corosync, para

ello se crean unas claves de autentificación, y se comparten entre los 3 nodos. También se modifica la configuración a través del fichero *corosync* y se habilita el servicio.
Tambíen se modela el servicio de haproxy en los 3 nodos, esto se hace a través de otro fichero de configuración, *haproxy.cfg*. Finalmente se habilita el servicio de corosync para los 3.
Para continuar, en uno de los maestros (en nuestro caso elegimos el maestro 1), se configuran los recursos de corosync como es el *virtual Ip* y el recurso de *haproxy*. Estos dos recursos se agrupan, de esta manera siempre prevalecen en un mismo nodo y se puede dar un servicio correcto.[1]
La componente LB que se muestra fuera de los maestros y del host, es una forma de expresar que el balanceador **de capa 7** se encuentra disponible con la VIP, y no es necesario acceder a la IP de un maestro. El balanceador junto a la VIP, se encargará de redirigir la petición de kubectl al nodo correspondiente.[3]

2.2. Configuración de K3s

Como backend se ha optado por DQLite, está en fase experimental pero es muy simple de configurar y por problemas de tiempo ha sido el elegido. Se planteó también un despliegue con etcd pero por problemas en la instalación y el poco tiempo se descartó.
Para confgiruar DQLite, necesitamos lanzar **k3s** en uno de los maestros con la opción ***–cluster-init***, los otros dos maestros se uniran al cluster lanzando **k3s** con la opción ***–server https://IPMaestro1:puerto***.
Sin embargo, los nodos agentes se unen al cluster de kubernetes con la opción ***–server https://VIP:puerto***, es decir, se unen a través de la IP virtual, esto es debido a que necesitan que el balanceador le asigne el maestro que en ese momento se encuentra en ejecución.

2.3. Prometheus y Dashboard

Para esta parte del trabajo, como no se disponía de mucho tiempo restante, no se han podido realizar muchas pruebas, sin embargo, si que se ha desplegado en el sistema.
Para desplegar Prometheus, ha sido necesario instalar *Helm* en la máquina que corre Kuebctl, en nuestro caso el Host. A continuación se instala prometheus con este comando 5 y seguido de esto se puede observar como empiezan a lanzarse los pods, deployments y replicasets correspondientes al servicio de monitorización y alertas.
Para poder externalizar el servicio de monitorización, es decir, para que se pueda acceder desde un navegador del host, fue necesario modificar el manifiesto tanto del dashboard *grafana*, como de prometheus, ambos con un comando como este 5.
La modificación consistía en cambiar el tipo de servicio de *ClusterIP* a *LoadBalancer*, con se consigue que el servicio de monitorización y alertas sea accesible desde fuera del cluster.
Por último, para acceder al dashboard *grafana*, es necesario un usuario (admin) y una contraseña, esta se obtiene con este comando 5.

3. Pruebas

Las pruebas que se han realizado en primer lugar han sido con corosync y pacemaker. Estas pruebas consistían en tirar el nodo que contenía los recursos y ver si pacemaker era capaz de mover los recursos a otro maestro de manera correcta.
Otra de las pruebas realizadas ha consistido en lanzar 3 maestros y un worker, y lanzar un pod tipo

busybox, a continuación se procedió a tirar el nodo que contenía el recurso de HAProxy y VIP y ver si seguíamos siendo capaces de acceder al pod, es decir, si seguíamos teniendo acceso al cluster a través de otro maestro.

4. Problemas

Uno de los problemas principales que tuve, el cual me llevó a estar 5 días atascado en el mismo punto, consistía en que tras configurar VIP y HAProxy, no podía acceder al servicio, esto fue debido a que no relanzaba el servicio k3s, por lo que era inviable disponer de un cluster, únicamente se disponía de un nodo.
Otro fallo fue configurar HAProxy como *Open Cluster Framework (OCF)* en lugar de *Linux Standard Base (LSB)*. Esto, por algún motivo que desconozco, era solamente capaz de iniciar HAProxy en único nodo, y además si se reiniciaba el servicio en ese nodo, fallaba y caía el servicio "por múltiples peticiones de inicio".
Tambíen tuve problemas con kubectl, porque en un principio parecía que funcionaba, sin embargo, al mirar el fichero */.kube/config* observé que la ip del server era la de un maestro únicamente y no la ip virtual, por lo que tuve que modificar esto para asegurarme completamente de que accediendo a la VIP se disponia de servicio.
Por otro lado, el comando kubectl que se realiza a la VIP y no al maestro directamente, la VIP no dispone de las credenciales de seguridad como lo disponen cada uno de los nodos maestros, por lo que nos saca un error de acceso por credenciales, esto se soluciona añadiendo la opción *–insecure-skip-tls-verify* una única vez, tras esto ya podemos acceder sin tener que poner la opción en cada comando.
Por último, tuve problemas con la automatización, ya que en primer lugar lo que realizaba era copiar scripts con privision.sh a las máquinas y luego ejecutar estas mediante vagrant ssh, tras esto me di cuenta que era innecesario y redundante ya que podía ejecutar el script desde provisión directamente.

5. Anexo - Comandos Usado

```
helm install stable/prometheus-operator --generate-name
```

```
kubectl edit svc nombrePod
```

```
kubectl get secret podGrafana -o \
jsonpath='{.data.admin\-password\}' | \
base64 --decode ; echo
```

6. Anexo - Ficheros incluidos en la entrega y sus usos

Todos estos ficheros se localian en /vagrant de esta manera podemos copiar en las máquinas a través del directorio compartido con las mv.

- **Authkey**: Fichero de claves para corosyn. Estas se han generado una única vez y se usan para las tres máquinas.
- **k3s**: Ejecutable k3s
- **provision.sh**: Fichero de aprovisionamiento que se encarga de instalar y lanzar todos los servicios.
- **corosync.conf**: Fichero de configuración del servicio corosync.
- **haproxy.cfg**: Fichero de configuración del servicio haproxy.
- **install.sh**: script de inicio de k3s
- **pcmk**: fichero necesario para plugin de pacemaker en corosync
- **vagrantfile**: fichero vagrant

Referencias

[1] Clusterlabs. Corosync resource. https://clusterlabs.org/pacemaker/doc/en-US/Pacemaker/1.1/html/Clusters_from_Scratch/_add_a_resource.html.

[2] DigitalOcean. Ha cluster corosync and pacemaker. https://www.digitalocean.com/community/tutorials/how-to-create-a-high-availability-setup-with-corosync-pacemaker-and-floating

[3] DigitalOcean. Haproxy setup corosync and pacemaker. https://www.digitalocean.com/community/tutorials/how-to-create-a-high-availability-haproxy-setup-with-corosync-pacemaker-and-