Kubernetes 1/

German Garces. Tiempo dedicado: 5h45min

Resumen

En este trabajo se ha extendido los conocimientos previos de Kubernetes (ya se había estudiado kubernetes en el pasado por cuenta propia) y se han comprendido diferentes elementos que se pueden usar a la hora de gestionar un clúster kubernetes. Se ha aprendido como desplegar un conjunto de pods y servicios y a gestionar estos.

Objetivos y arquitectura de elementos relevantes

Respecto a la arquitectura, se han comprendido varias cosas interesantes. En primer lugar, como se distribuye kubernetes, me explico, a que no solo contiene nodos master y nodos worker que corren pods en ellos. Si no que también existen servicios y add-ons que permiten sacar mas ventajas de un clúster de kubernetes.

También, se ha estudiado lo que propone la documentación oficial: Imagen de la documentación oficial (Si quiero que la memoria me quepa en 5 paginas no puedo poner la imagen)

Y se han entendido las diferentes partes que no se cubren en el trabajo como el kube-controller manager que es un "servicio" para controlar el estado de los nodos, controlar la replicación de pods... También se ha estudiado por encima el kube-scheduler que se encarga de relacionar pods que se estan creando con nodos. También se ha visto la función de kubelet que es la de asegurarse que los contenedores en marcha se encuentran corriendo en un pod.

Explicación aplicaciones desplegadas

Nota

Anteriormente a la realización del trabajo se estudió la documentación de kubernetes y se realizaron prácticas por cuenta propia. Debido a esto, en algunos apartados se ha visto que apenas hay nada que documentar ya que ya se sabía.

Pods

Se ha aprendido como obtener las métricas de los pods mediante el comando kubectl top pod. Se ha aprendido como limitar los recursos que usa un pod mediante

```
resources:
limits:
memory: "64Mi"
cpu: "500m"
```

Un concepto interesante es el hecho de poder aplicar configuraciones a traves de ficheros .yaml ya que permite indicar al sistema cual es el estado que se desea para cada pod.

Labels

Se ha aprendido como añadir etiquetas a pods con el comando kubectl label pods labelex clave=valor. Se ha aprendido que se pueden borrar pods mediante filtros basados en las etiquetas kubectl delete pods -l 'env in (production, development). Todo el tema de las etiquetas da mucho juego a la hora de trabajar con objetos (ya sean pods, servicios, addons...) ya que permiten relacionarlos entre si (modificar un grupo de nodos, relacionar un servicio con un conjunto de pods...).

Nodes

Se ha aprendido como obtener las métricas de los diferentes nodos disponibles mediante kubectl top nodes Se ha aprendido como hacer que un pod corra en un nodo específico mediante nodeSelector y añadiendole una etiqueta shouldrun: here al nodo elegido

Aspectos adicionales

Se ha estudiado la documentación de Kubernetes sobre este apartado y se ha visto que en nodeSelector se pueden usar otro tipo de datos como por ejemplo disktype. También cabe decir que los nodos vienen con una serie de etiquetas estandar las cuales se pueden usar con el mismo propósito. Y por último, se ha estudiado la opción Node affinity, la cual presenta dos opciones: requiredDuringSchedulingIgnoredDuringExecution o preferredDuringSchedulingIgnoredDuringExecution. Se pueden entender como "hard nodeSelector" y como "soft nodeSelector". La parte de ambas opciones IgnoredDuringExecution se encarga de que si durante la ejecución del pod, cambian las etiquetas del nodo de tal manera que ya no se cumplan las reglas de afinidad, el pod seguirá corriendo en dicho nodo.

```
nodeSelectorTerms:
- matchExpressions:
- key: kubernetes.io/e2e-az-name
operator: In
values:
- e2e-az
- e2e-az
```

Logging

Se ha aprendido como observar la salida de los pods tanto de manera convencional como en "stream". Se ha aprendido a observar la salida de pods que ya han terminado con el subcomando -p. Estas herramientas proporcionadas son muy útiles para comprobar el comportamiento esperado de los contenedores.

API Server access

Se ha aprendido como abrir un proxy con el comando kubectl proxy --port=XXX. Se ha aprendido como explorar las versiones soportadas por la API y sus recursos con el comando kubectl api-versions y kubectl api-resources. La API es esencial en kubernetes ya que el punto central el cual es accedido por usuarios y componentes del clúster.

Deployments

Se ha observado como al realizar una modificación a una serie de pods, estos no se actualizan, si no que finalizan y se crean otros pods con las modificaciones anteriores. Se ha aprendido a observar las réplicas con kubectl get rs. Se ha aprendido que las nuevas réplicas contienen nombres e IPs diferentes. Se ha aprendido como observar la historia de los despliegues mediante kubectl rollout history deploy/sise-deploy. Se ha aprendido como hacer roll back a una versión específica mediante kubectl rollout undo deploy/sise-deploy --to-revision= Por deployment se ha entendido que es el proceso de control que se encarga de llevar al pod del estado actual al estado deseado (estados declarados en su fichero .yaml.

Services

Se ha aprendido como obtener los servicios desplegados mediante kubectl get svc. Se ha aprendido como funciona el redirecionamiento de la IP del pod a la VIP del servicio. Se ha observado como Kubernetes configura por defecto que las peticiones se repartan de manera equitativa a las diferentes réplicas de un pod

```
-A KUBE-SVC-EZC6WLOVQADP4IAW -m statistic --mode random --probability
0.50000000000 -j KUBE-SEP-VU2JGOHHBAZT2PPO
```

Se ha estado estudiando la documentación oficial sobre servicios y se ha visto los diferentes usos que se le pueden asignar, por ejemplo un servicio balanceador de carga, un servicio que descubra otros servicios entre aplicaciones...

Service Discovery

Se ha aprendido como funciona el "DNS-based service discovery" y como acceder a los servicios mediante el DNS ($SVC.$NAMESPACE.svc.cluster.local). Este apartado es bastante intuitivo con los conocimientos previos de la asignatura.

Port Forward Se ha aprendido como redirigir el tráfico de un puerto a otro. Dato que me parece interesante de comentar es que se indicar que el "Port forwarding" no esta hecho para el tráfico en producción, si no para desarrollo y experimentación.

Namespaces Se ha estudiado la documentación oficial sobre este tema y se han visto diferentes "motivaciones" para crear diferentes nombres de espacios. Por ejemplo, delegar autoridades a ciertos usuarios, limitar la cuota de disco y recursos a otros...

Volumes Son efímeros, mueren junto al pod, si se quieren volumenes persistentes hay que usar Persistent Volumes. "Los datos de los volúmenes son preservados aunque los contenedores se reinicien". Diferentes tipos de volumentes: node-local,file-sharing, cloud provider... Como especificar donde montar los volumenes y como:

```
volumeMounts:
- name: xchange
mountPath: "/tmp/xchange"
volumes:
- name: xchange
emptyDir: {} # temp dir that shares a pod`s lifetime
```

Persistent Volumes

Se ha estudiado la documentación oficial y se ha visto que existen 2 maneras de crear volumenes persistentes:

1. De manera estática, es decir, los crea el administrador
2. De manera dinámica, cuando no existen PV que concuerden con el nombre del PV claim, en este caso, Kubernetes crea el volumen persistente. Se ha visto en las clases de teoría que no se pueden hacer volumenes persistentes de volumenes persistentes -> Hay que tenerlo en cuenta ya que no es algo "obvio". El ejemplo de clase de crear un nfs dentro de un fs convencional lo deja ver muy claramente.

StatefulSet

StatefulSet provee ciertas garantías sobre la identidad de los pods que maneja y sobre el orden de inicio -> Esto es muy útil para tema servidores, por ejemplo, que se guarde la dirección DNS aunque un pod se caiga. StatefulSet maneja un volumen persistente para cada pod (para la persistencia del estado) y para la red, usa un headless service (para eso, se especifíca None en clusterIP). StatefulSet no es siempre la mejor opción, a veces renta más definir un recurso a medida.

Jobs

Un trabajo se encarga de que un numero específico de Pods terminen de manera exitosa. Ya sea creando uno o mas Pods. Se ha estudiado la posibilidad de lanzar trabajos de manera periódica mediante CronJob. Muy útiles a la hora de realizar "escaneos del sistema", aunque también se ha visto en la documentacion oficial que esta labor se puede realizar mediante un add-on dashboard.

Explicación funciones, conceptos y recursos en provision.sh y Vagrantfile

Vagrantfile

Aquí se encuentran las configuraciones de las máquinas virtuales a desplegar

```
NODES = [
{ hostname: 'm', type: "master", ip: MASTER, mem: 1000 , m: MASTER },
{ hostname: 'w1', type: "worker", ip: '192.168.1.93', mem: 1000 , m: MASTER
},
{ hostname: 'w2', type: "worker", ip: '192.168.1.94', mem: 1000 , m: MASTER
},]
```

Como se puede ver en la definición, se establecen el nombre de cada máquina, su tipo (master/worker), su dirección ip, la memoria designada (Mb por defecto) y un m que corresponde a la dirección de la máquina MASTER. También cabe destacar que se configura cada máquina para que trabaje con 1 núcleo del CPU y un NIC virtio. virtio es una plataforma para trabajar con entrada salida de manera virtualizada.

```
nodeconfig.vm.provider "virtualbox" do |v|
v.customize ["modifyvm",:id,"--memory",node[:mem],"--cpus","1"]
v.default_nic_type = "virtio"
```

También, es digno de mencionar el siguiente apartado de configuración:

```
nodeconfig.vm.provision "shell",
path: 'provision.sh',
args: [ node[:hostname], node[:ip], node[:m], node[:type] ]
```

La variable shell nos permite ejecutar un script dentro de la máquina virtual que esta siendo creada. En este caso, el script provision.sh el cual se comentará después. Por último, destacar el uso de triggers.

```
nodeconfig.trigger.after :up do |trigger|
trigger.run = \
{inline: "sh -c 'cp k3s.yaml /home/ger/.kube/config'"}
end
```

Se puede observar que este trigger se ejecuta despues de que la máquina haya pasado por el estado :up.

provision.sh

A este script, Vagranfile le pasa el hostname de cada máquina, la ip, la ip del master y el tipo de nodo (master, worker). El script se encarga de establecer la zona horaria de la máquina en cuestión, su configuración básica de internet y los hosts con los que tendrá que comunicarse (el resto de máqinas virtuales creadas). Tras eso, se copia el binario k3s a /usr/local/bin. Por último, en función del tipo de nodo (MASTER, WORKER), se llama al script install.sh que se encarga de instalar un servidor o un agente de K3s con su token y otras configuraciones necesarias.

Problemas encontrados y solución

En el apartado **Services** no se encontraba como hallar la VIP del servicio. Se vió que la ip que van a usar el resto de pods no era esa, si no la del pod del servicio, de redirigir las peticiones se encarga iptables. No se han encontrado más problemas ya que la guía esta muy bien explicada y se complementa muy facilmente con la documentación oficial de kubernetes.

Puesta en marcha

En ´/home/ger/proyecto/Proyecto2Parte1/vagrantk3s/´ ejecutar `vagrant up`. Esperar unos minutos y comprobar que estan los nodos corriendo de manera exitosa con el comando `kubectl get nodes`. Para aplicar pods o servicios cuya configuración se encuentra en ficheros `.yaml` : `kubectl apply -f ruta/Fichero.yaml`