README.md

Miniproyecto Sistemas Distribuidos

Universidad ICESI
Curso: Sistemas Distribuidos
Docente: Daniel Barragán C.
Tema: Kubernetes
Estudiante 1: Luis Rosales - A00212740 **Estudiante 2:** Christian Cárdenas - A00315320 **Estudiante 3:** Paula Bolaños - A00068008 **URL:** www.github.com/paulaandrea27/sd-project

Objetivos

- Identificar los componentes de un cluster de kubernetes
- Desplegar un cluster de kubernetes
- Desplegar aplicaciones en un cluster de kubernetes empleando sus propiedades de volumenes persistentes, balanceo de carga y descubrimiento de servicio
- Diagnosticar y ejecutar de forma autónoma las acciones necesarias para lograr infraestructuras estables

Prerrequisitos

- Cluster de Kubernetes

Descripción

Debera realizar el despliegue de un cluster de kubernetes en al menos tres nodos independientes (un nodo maestro y al menos dos nodos de trabajo). Para ello realice la instalación y configuración de las dependencias necesarias (kubeadm, kubectl, kubelet), y el despliegue de los pods de red necesarios para la comunicación. Todo el proceso debera ser debidamente documentado

Para la prueba del funcionamiento del cluster deberá realizar el despliegue de un pod con al menos un servicio web que se ejecute en el framework de su preferencia. La especificación del deployment debe cumplir con los siguientes requerimientos:

- Cantidad de replicas: 3
- Asignar una etiqueta que identifique al pod

La especificación del servicio debe mapear un puerto de cada pod a un balanceador de carga, de esta manera la aplicación desplegada debe poder ser accedida a través de un navegador o cliente de consola (curl, wget)

Opcional:

- Plantee e implemente una estrategia para el monitoreo de los contenedores (Pods) puede emplear alguna de las siguientes tecnologías: kubernetes health checks, efk stack, cadvisor, otros.
- Plantee e implemente una estrategia para la automatización de la conexión a través de tokens

Nota

- No se requiere que incluya archivos de la automatización del cluster o infraestructura base. El empleo de tecnologías de automatización para la infraestructura base es a consideración del estudiante

Actividades

1. Consigne la documentación que incluya los comandos de linux necesarios para el aprovisionamiento del cluster de kubernetes.

- Instalación y configuración de kubeadm, kubectl, kubelet

Una vez se tengan los nodos con Centos 7 actualizados y con docker instalado, se procede a instalar kubeadm, kubectl y kubelet, para ello se deben ejecutar los siguientes comandos:

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://packages.cloud.google.com/yum/doc/r
EOF
```

Ahora, se debe deshabilitar SELinux:

```
setenforce 0
sed -i --follow-symlinks 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/sysconfig/selinux
```

Después, para evitar broblemas que se puedan presentar con el enrutamiento del tráfico se añade lo siguiente al archivo /etc/sysctl.conf:

```
vi /etc/sysctl.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

Luego, se debe deshabilitar SWAP, comentando la línea de swap del archivo /etc/fstab. Para que los cambios se han hecho se apliquen, se debe reiniciar la máquina:

```
sysctl --system
```

Finalmente, se hace la instalación, se habilitan los servicios y se inician:

```
yum install -y kubelet kubeadm kubectl
systemctl enable kubelet
systemctl start kubelet
```

Para la configuración, se deben ejecutar los siguientes comandos con el fin de modificar el archivo de configuración del kubeadm. Se debe modificar el cgroup para que coincida con el de docker y el parámetro de extras para deshabilitar el swap:

```
sed -i "s/cgroup-driver=systemd/cgroup-driver=cgroupfs/g" /etc/systemd/system/kubelet.service.d/10-kubea
```

```
vi /etc/systemd/system/kubelet.service.d/10-kubeadm.conf
Environment="KUBELET_EXTRA_ARGS=--fail-swap-on=false
```

- Instalación y configuración de un pod de red que interconecte los pods a desplegar en el cluster

Primero en /etc/hosts se agrega en cada nodo lo siguiente y se prueba que haya conectividad cuando se ejecuta ping node(n). Para este caso, el nodo 3 será el nodo master y los otros nodos (2 y 3) los workers.

```
192.168.0.16    node3
192.168.0.17    node2
192.168.0.18    node1
```

En el nodo master es decir el 3 se ejecuta el siguiente comando para iniciar el clúster.

```
kubeadm init --apiserver-advertise-address $(hostname -i)
```

Después en los otros nodos se ejecuta el comando kubeadm join [flags] para unirlos al clúster. En la siguiente captura se muestra el comando completo que es dado automáticamente por kubernetes al momento de ingresar el comando kubeadm init.

```
  kubeadm join --token b3361c.ce7bdb8c9ee20603 192.168.0.16:6443 --discovery-token-
ca-cert-hash sha256:b6ed714d7a407119c6e4dd1768d36eea6071a4400992649d11d0a690b5ab80c
4
```

Con el comando kubectl get nodes podemos verificar que los nodos se hayan agregado correctamente.

```
NAME      STATUS     ROLES     AGE       VERSION
node1     NotReady   <none>    7m        v1.10.2
node2     NotReady   <none>    7m        v1.10.2
node3     NotReady   master    9m        v1.10.2
```

Además, ejecutando estos comandos que son dados también por kubernetes

```
To start using your cluster, you need to run (as a regular user):

  mkdir -p $HOME/.kube
  sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
  sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Para iniciar el clúster de red, es decir, para la conexión entre los nodos se ejecuta lo siguiente:

```
kubectl apply -n kube-system -f \
> "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version | base64 |tr -d '\n')"
```

Por otra parte, el dockerfile que se utilizó es el siguiente:

```
FROM golang:alpine3.6 AS binary
ADD . /app
WORKDIR /app
RUN go build -o http

FROM alpine:3.6
WORKDIR /app
ENV PORT 8000
EXPOSE 8000
COPY --from=binary /app/http /app
CMD ["/app/http"]
```

Este dockerfile corre el siguiente script en go, que da el nombre de host de la maquina o contenedor en el que se ejecuta.

```
package main

import (
  "os"
  "fmt"
  "net/http"
  "log"
)

func main() {
  port := os.Getenv("PORT")
  if port == "" {
      port = "8080"
  }

  fmt.Fprintf(os.Stdout, "Listening on :%s\n", port)
  hostname, _ := os.Hostname()
  http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
    fmt.Fprintf(os.Stdout, "I'm %s\n", hostname)
    fmt.Fprintf(w, "I'm %s\n", hostname)
  })

  log.Fatal(http.ListenAndServe(":" + port, nil))
}
```

Para construir la imagen y agregarla al docker hub se utilizaron los siguientes comandos:

```
docker build –t go-service .
docker login
docker tag go-service luisf0425/go-service
docker push luisf0425/go-service
```

- Los comandos a automatizar en el archivo deploy.yaml son los siguientes:

Este comando corre el deployment de kubernetes:

```
kubectl run web-deployment --image=luisf0425/go-service --replicas=3 --port=8000 --labels=app=web
```

Este comando permite exponer el deployment como un servicio:

```
kubectl expose deployment web-deployment --name=go-service --port=8000 --target-port=8000
```

Ahora, podemos verificar que construyó correctamente la infraestructura:

```
[node3 ~]$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP          NODE
web-deployment-8679958684-gn7tr   1/1       Running   0          18m       10.40.0.1   node1
web-deployment-8679958684-snwc4   1/1       Running   0          18m       10.46.0.2   node2
web-deployment-8679958684-wxh9d   1/1       Running   0          18m       10.40.0.2   node1
[node3 ~]$ kubectl get service
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
go-service   ClusterIP   10.110.138.123   <none>        8000/TCP   18m
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP    1h
[node3 ~]$
```

También se pueden usar los siguientes comandos para verificar que sí funciona adecuadamente. Este da la IP:

```
export SERVICE_IP=$(kubectl get service go-service -o go-template='{{.spec.clusterIP}}')
```

Este arroja el puerto:

```
export SERVICE_PORT=$(kubectl get service go-service -o go-template='{{(index .spec.ports 0).port}}')
```

El siguiente corre un pod para probar:

```
kubectl run busybox  --generator=run-pod/v1 --image=busybox --restart=Never --tty -i --env "SERVICE_IP=$
```

Por último, este permite probar el funcionamiento del balanceador de carga:

```
wget -qO- http://$SERVICE_IP:$SERVICE_PORT
```

En la siguiente imagen es posible observar el correcto funcionamiento del balanceador de carga, puesto que al acceder por una IP, devuelve el hostname de cada contenedor:

A continuación, se encuentra el archivo deploy.yaml que automatiza los comandos vistos. Este incluye el deployment.yaml y el service.yaml.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: web-deployment
spec:
  selector:
    matchLabels:
      run: web
  replicas: 3
  template:
    metadata:
      labels:
        run: web
    spec:
      containers:
      - name: web
        image: luisf0425/go-service
        ports:
        - containerPort: 8000
---
apiVersion: v1
kind: Service
metadata:
  name: go-service
  labels:
    run: web
spec:
  ports:
```

```
  - port: 8000
    targetPort: 8000
    protocol: TCP
  selector:
    run: web
```

Para ejecutarlo se debe ingresar el siguiente comando:

```
kubectl apply –f deploy.yaml
```

- Demostración que ante la caída de uno de los nodos que forman parte del cluster, los pods son reasignados automáticamente a un nodo saludable. Incluya los archivos: deployment.yml, service.yml empleados.

Lo primero que se muestra es el correcto despliegue de los pods:

```
[node3 ~]$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP          NODE
web-deployment-fdc7588d4-fdlpc    1/1       Running   0          2m        10.40.0.1   node1
web-deployment-fdc7588d4-nzkbt    1/1       Running   0          2m        10.46.0.2   node2
web-deployment-fdc7588d4-tdtn5    1/1       Running   0          2m        10.40.0.2   node1
```

Posteriormente, se evidencia que el servicio está corriendo como se espera:

```
[node3 ~]$ kubectl get service
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
go-service   ClusterIP   10.111.86.200   <none>        8000/TCP   2m
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    1h
```

En la siguiente captura, es posible ver que el balanceador de cargas opera de manera correcta, es decir que, se puede acceder desde una sola ip (se aplicaron los mismos comandos con los que se verificó el funcionamiento de los comandos a automatizar en la subsección superior):

```
/ # wget -qO- http://$SERVICE_IP:$SERVICE_PORT
I'm web-deployment-fdc7588d4-nzkbt
/ # wget -qO- http://$SERVICE_IP:$SERVICE_PORT
I'm web-deployment-fdc7588d4-tdtn5
/ # wget -qO- http://$SERVICE_IP:$SERVICE_PORT
I'm web-deployment-fdc7588d4-fdlpc
```

Finalmente, se demuestra que si se elimina un nodo, en este caso el node1, los pods son reasignados al node2 automáticamente:

```
[node3 ~]$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP          NODE
web-deployment-fdc7588d4-fdlpc    1/1       Running   0          4m        10.40.0.1   node1
web-deployment-fdc7588d4-nzkbt    1/1       Running   0          4m        10.46.0.2   node2
web-deployment-fdc7588d4-tdtn5    1/1       Running   0          4m        10.40.0.2   node1
[node3 ~]$ kubectl delete node node1
node "node1" deleted
[node3 ~]$ kubectl get pods -o wide
NAME                              READY     STATUS              RESTARTS   AGE       IP          NODE
web-deployment-fdc7588d4-jh8ns    0/1       ContainerCreating   0          0s        <none>      node
2
web-deployment-fdc7588d4-nzkbt    1/1       Running             0          4m        10.46.0.2   node
2
web-deployment-fdc7588d4-xf2rb    0/1       ContainerCreating   0          0s        <none>      node
2
[node3 ~]$ kubectl get pods -o wide
NAME                              READY     STATUS    RESTARTS   AGE       IP          NODE
web-deployment-fdc7588d4-jh8ns    1/1       Running   0          13s       10.46.0.3   node2
web-deployment-fdc7588d4-nzkbt    1/1       Running   0          4m        10.46.0.2   node2
web-deployment-fdc7588d4-xf2rb    1/1       Running   0          13s       10.46.0.4   node2
[node3 ~]$
```

4. Documente algunos de los problemas encontrados y las acciones efectuadas para su solución al aprovisionar la infraestructura y aplicaciones.

Uno de los problemas presentados estuvo relacionado con la inicialización del kubeadm, ya que el swap se estaba iniciando también y fue necesario modificar el archivo de configuración. Después, se presentó una dificultad con los puertos, ya que se estaba haciendo uso del mismo puerto de la aplicación en el service.yaml. Lo anterior se solucionó cambiando el puerto al 8000. El último inconveniente se dió con los nombres en los labels, puesto que se tenían diferentes nombres. Esto se solucionó poniéndolos igual, que para este caso fue web, tanto en el deployment.yaml como en el service.yaml.

Referencias

- https://kubernetes.io/docs/tasks/tools/install-kubeadm/
- https://blog.tekspace.io/setup-kubernetes-cluster-on-centos-7/
- https://github.com/kubernetes/kubernetes/issues/53333