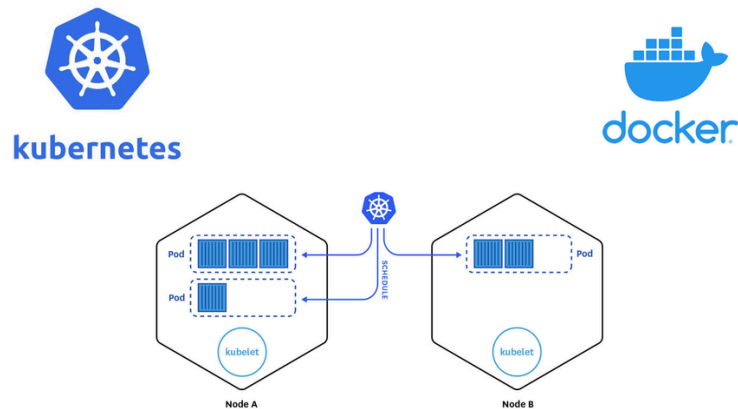


Instalación de un cluster de Kubernetes con 3 nodos master y 4 nodos worker sobre RedHat 9.3, on-premise.

- Introducción
 - Qué es un container o contenedor?
 - Por qué Kubernetes?
 - Qué es un pod?
 - Qué es un replicaset?
 - Qué es un deployment?
 - Qué es un servicio?
- Prerrequisitos
- Despliegue de Kubernetes con 3 nodos master y 4 nodos worker
 - Requisitos mínimos de hardware
 - Requisitos de Software
 - Dependencias y Herramientas
 - Prerrequisitos de Configuración
 - Puertos Necesarios
 - Instalación de Dependencias (Todos los nodos)
 - Instalación de Containerd (Todos los nodos)
 - Instalación de paquetes de Kubernetes (Todos los nodos)
 - Configurar y Inicializar el Clúster Kubernetes
 - Direcciones IP a utilizar:
 - Nodos master:
 - Nodos Worker:
 - Balanceador de carga HAProxy (Alta disponibilidad de los nodos master del cluster):
 - Balanceador de carga de aplicaciones MetalLB:
 - Configuración de HAProxy:
 - Instalar HAProxy en cada nodo master
 - Archivo de configuración HAProxy
 - Instalar y configurar Keepalived
 - Instalar keepalived en cada nodo donde HAProxy se encuentra configurado
 - Configurar Keepalived para gestionar un VIP (Virtual IP) entre los nodos HAProxy.
 - Nodo 1 (Prioridad más alta - 172.16.1.91)
 - Nodo 2 (Prioridad más alta - 172.16.1.92)
 - Nodo 3 (Prioridad más alta - 172.16.1.93)
 - Crear el script de verificación de HAProxy:
 - Convertir el script en ejecutable:
 - Reiniciar y habilitar Keepalived:
 - Inicialización de cluster
 - Master
 - Workers
 - Instalar CNI (Calico)
 - Verificación de la Instalación
 - Verificación del estado del cluster
 - Instalación y configuración de HA-Proxy para alta disponibilidad de Kubernetes
 - Secuencia de Apagado y Encendido del cluster de Kubernetes
 - Apagado
 - Apagar los nodos (workers)
 - Apagar los máster

- Encendido
 - Inicio de los nodos máster
 - Inicio de los nodos esclavos (workers)
 - Troubleshooting durante el proceso de reinicio del cluster
 - Problemas con pods atascados con status "Terminating"
 - Problemas después del reinicio:
- Conclusión

Introducción



Kubernetes ("k8s" o "Kube") es una solución de código libre (open source) que permite la automatización del despliegue, escalado y gestión de contenedores. Se puede definir como un "orquestador inteligente" de contenedores que ofrecen microservicios. Fue originalmente diseñado por Google en 2014 y donado a la Cloud Native Computing Foundation en 2015. Kubernetes soporta diferentes motores de contenedores, siendo Docker el más habitual.

Este manual detalla los pasos para instalar y configurar un clúster de Kubernetes con tres nodos master y cuatro nodos worker utilizando containerd como runtime de contenedores. Se cubrirán todos los aspectos necesarios, incluyendo la instalación de dependencias, la creación de claves de certificados, la unión de los nodos master y worker, la habilitación de IP forwarding y la configuración del firewall con firewalld. El objetivo es proporcionar una guía comprensiva y detallada que permita implementar un clúster de Kubernetes funcional en RedHat 9.3.

Qué es un container o contenedor?

Un container es un paquete de software que contiene los servicios esenciales para que una aplicación contenida en este pueda funcionar de manera adecuada y sin necesidad de tener una dependencia directa del sistema operativo, lo que los hace muy versátiles y livianos. Un container contiene exclusivamente los servicios necesarios para que la aplicación que se empaqueta pueda correr sin ningún problema y se pueda replicar de acuerdo a las necesidades del usuario.

Por qué Kubernetes?

Para crear containers existen herramientas como Docker las cuáles a través de scripts permiten a los programadores desplegar aplicaciones embebidas en dicho formato, sin embargo, ¿cómo se podrían manejar varios containers a la vez?, o ¿cómo se podrían orquestar múltiples aplicaciones optimizando recursos al máximo a través de un cluster en un Data Center?, es por eso que Kubernetes tiene suma importancia permitiendo a múltiples nodos de hardware mantener múltiples aplicaciones basadas en containers .

⚠ Kubernetes requiere como mínimo, de un nodo maestro y una serie de nodos esclavos/workers, es importante tener claro que un nodo se define como un conjunto de recursos de Hardware.

Qués es un pod?

Un Pod es el objeto más pequeño y básico de Kubernetes, es una unidad básica de ejecución que representa procesos corriendo en el cluster. Un pod tiene la función de encapsular un container (o en algunos casos múltiples containers), recursos de almacenamiento, una dirección IP única que es válida solo dentro del cluster y las opciones que definen como el container debe ejecutarse. Para fines prácticos, un Pod puede ser la definición lógica de un Container en Kubernetes.

Los pods tiene un ciclo de vida definido por un Deployment y un ReplicaSet, estos vivirán de acuerdo a las necesidades que se definan en un Deployment y serán forzados a vivir o morir de acuerdo a la definición del ReplicaSet.

Qués es un replicaset?

Un ReplicaSet es una regla definida mediante un Deployment en Kubernetes y utilizada para garantizar la disponibilidad de un número específico de Pods idénticos. Para entenderlo de manera más sencilla, el ReplicaSet es el comandante de un pelotón y cada soldado es un Pod, estos le obedecerán sin cuestionamientos, incluso si tienen que sacrificarse.

Qués es un deployment?

Un Deployment es un conjunto de ordenes declaradas que se le dan a al cluster, todos los ReplicaSets y los Pods se alinean bajo las instrucciones del Deployment, por ejemplo, un Deployment de una aplicación web puede decirle a un ReplicaSet cuántos Pods se requieren para mantener la demanda de esta aplicación bajo ciertas circunstancias, incluso, se pueden definir reglas de escalamiento/crecimiento en caso de falta de recursos, se pueden definir políticas que controlen dicha asignación basadas en cantidad de requests simultáneos, en tal caso el ReplicaSet puede aumentar los Pods de 5 a 10, y esos Pods se distribuirán en el cluster donde haya recursos de Hardware disponibles.

Qués es un servicio?

En Kubernetes, un Service es una abstracción que define un conjunto lógico de Pods y una política por la cual acceder a ellos (algunas veces este patrón es llamado micro-servicio). El conjunto de Pods a los que apunta un Servicio se determina usualmente por un Selector.

Prerrequisitos

1. **Sistema Operativo:** RedHat 9.3
2. **Acceso a Internet:** Necesario para descargar paquetes y dependencias.
3. **Privilegios de Superusuario:** Necesario para instalar y configurar el sistema.
4. **Configuración de Red:** Asegurarse de que los nodos pueden comunicarse entre sí.

Despliegue de Kubernetes con 3 nodos master y 4 nodos worker

Kubernetes es una potente plataforma de orquestación de contenedores utilizada para automatizar la implementación, el escalado y la gestión de aplicaciones en contenedores. En este documento, se plasma un paso a paso del proceso de instalación de Kubernetes en sistema operativo RedHat 9.3.

Esta configuración de clúster incluye un nodo maestro y nodos trabajadores, lo que le permite aprovechar todo el poder de Kubernetes.

Requisitos mínimos de hardware

Antes de comenzar con la instalación de Kubernetes, es esencial asegurarse de que todos los nodos cumplen con los requisitos mínimos de hardware y software. Estos requisitos aseguran que el clúster pueda operar de manera eficiente y estable.

1. **Nodos Master:**
 - CPU: Al menos 2 CPU
 - Memoria RAM: Mínimo 2 GB de RAM (se recomienda 4 GB)

- Almacenamiento: Al menos 20 GB de espacio en disco

2. **Nodos Worker:**

- CPU: Al menos 1 CPU
- Memoria RAM: Mínimo 1 GB de RAM (se recomienda 2 GB)
- Almacenamiento: Al menos 20 GB de espacio en disco

Requisitos de Software

1. **Sistema Operativo:**

- RedHat 9.3 (se recomienda un sistema limpio y actualizado)

2. **Acceso a Internet:**

- Necesario para descargar paquetes y dependencias.

3. **Privilegios de Superusuario:**

- Se requiere acceso root o permisos sudo para instalar y configurar el sistema.

4. **Configuración de Red:**

- Asegurarse de que todos los nodos pueden comunicarse entre sí a través de la red. Se recomienda configurar un nombre de host para cada nodo.

Dependencias y Herramientas

1. **Herramientas Básicas:**

- `curl`, `wget`, `vim`, `git`

2. **Componentes de Kubernetes:**

- `kubeadm`, `kubelet`, `kubectl`

3. **Runtime de Contenedores:**

- `containerd`

4. **Configuración de Red:**

- Calico (u otra solución de red compatible con Kubernetes)

Prerrequisitos de Configuración

1. **Deshabilitar SELinux:**

- SELinux debe estar en modo permisivo o deshabilitado para evitar problemas de permisos con Kubernetes.

2. **Habilitar IP Forwarding:**

- Necesario para permitir el tráfico entre contenedores y nodos.

3. **Configuración del Firewall:**

- Asegurarse de que los puertos necesarios para Kubernetes están abiertos.

4. **Sincronización de Relojes:**

- Se recomienda instalar y configurar `ntp` o `chrony` para asegurarse de que todos los nodos tienen la misma hora.

Puertos Necesarios

• **Nodos Master:**

- 6443/tcp (API Server)
- 2379-2380/tcp (etcd)
- 10250/tcp (kubelet)
- 10251/tcp (kube-scheduler)
- 10252/tcp (kube-controller-manager)

Instalación de Dependencias (Todos los nodos)

1. Actualizar el sistema:

```
sudo dnf update -y
```

2. Instalar dependencias básicas:

```
sudo dnf install -y curl wget vim yum-utils device-mapper-persistent-data kernel-devel-$(uname -r)
```

3. Deshabilitar SELinux:

```
sudo setenforce 0 && sudo sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config
```

4. Habilitar IP Forwarding:

```
1 sudo modprobe br_netfilter sudo echo '1' > /proc/sys/net/ipv4/ip_forward
2 sudo tee /etc/sysctl.d/k8s.conf <<EOF net.bridge.bridge-nf-call-ip6tables = 1 net.bridge.bridge-nf-call-iptables = 1 net.ipv4.ip_forward = 1
3 sudo sysctl --system
```

5. Configurar firewalld:

```
1 sudo systemctl start firewalld
2 sudo systemctl enable firewalld
3 sudo firewall-cmd --permanent --add-port=6443/tcp
4 sudo firewall-cmd --permanent --add-port=2379-2380/tcp
5 sudo firewall-cmd --permanent --add-port=10250/tcp
6 sudo firewall-cmd --permanent --add-port=10251/tcp
7 sudo firewall-cmd --permanent --add-port=10252/tcp
8 sudo firewall-cmd --permanent --add-port=10255/tcp
9 sudo firewall-cmd --reload
```

6. Agregar módulos de kernel necesarios:

```
1 sudo modprobe br_netfilter
2 sudo modprobe ip_vs
3 sudo modprobe ip_vs_rr
4 sudo modprobe ip_vs_wrr
5 sudo modprobe ip_vs_sh
6 sudo modprobe overlay
```

7. Cargar los módulos en la secuencia de arranque:

```
1 cat > /etc/modules-load.d/kubernetes.conf << EOF
2 br_netfilter
3 ip_vs
4 ip_vs_rr
5 ip_vs_wrr
6 ip_vs_sh
7 overlay
8 EOF
```

Instalación de Containerd (Todos los nodos)

1. Instalar containerd:

```
1 yum-config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
2 sudo dnf install -y containerd
3 sudo systemctl start containerd
4 sudo systemctl enable containerd
```

2. Configurar containerd:

```
1 sudo mkdir -p /etc/containerd
2 sudo containerd config default | sudo tee /etc/containerd/config.toml
3 sudo systemctl restart containerd
4
5 # Editar linea del archivo /etc/containerd/config.toml "SystemdCgroup"
6 # y reemplazar el valor "false" por valor "true"
```

Instalación de paquetes de Kubernetes (Todos los nodos)


1. Agregar repositorio de Kubernetes:

```
1 cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
2 [kubernetes]
3 name=Kubernetes
4 baseurl=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/
5 enabled=1
6 gpgcheck=1
7 gpgkey=https://pkgs.k8s.io/core:/stable:/v1.29/rpm/repodata/repomd.xml.key
8 exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
9 EOF
```

2. Instalar kubeadm, kubelet y kubectl:

```
1 dnf makecache; dnf install -y kubelet kubeadm kubectl --disableexcludes=kubernetes
2 sudo systemctl enable kubelet
3 sudo systemctl start kubelet
```

Configurar y Inicializar el Clúster Kubernetes

 Debido a que este cluster está configurado con alta disponibilidad, es necesario instalar y configurar los paquetes que permitirán este comportamiento.

Para ello se utilizarán estos componentes:

- HAProxy
- Keepalived

Direcciones IP a utilizar:

Nodos master:

- master1: 172.16.1.91
- master2: 172.16.1.92
- master3: 172.16.1.93

Nodos Worker:

- worker1: 172.16.1.94
- worker2: 172.16.1.95
- worker3: 172.16.1.96
- worker3: 172.16.1.97

Balanceador de carga HAProxy (Alta disponibilidad de los nodos master del cluster):

- IP para HAProxy (balanceador de carga): 172.16.1.98

Balanceador de carga de aplicaciones MetalLB:

- IP para MetalLB (balanceador de carga de aplicaciones): 172.16.1.99

Configuración de HAProxy:

Instalar HAProxy en cada nodo master

```
1 sudo apt update
2 sudo apt install haproxy
```

Archivo de configuración HAProxy

Editar la configuración de HAProxy para balancear la carga entre los nodos master de Kubernetes.

Archivo: `/etc/haproxy/haproxy.cfg`

```
1 global
2     log /dev/log local0
3     log /dev/log local1 notice
4     chroot /var/lib/haproxy
5     stats timeout 30s
6     user haproxy
7     group haproxy
8     daemon
9
10 defaults
11     log global
12     option httplog
13     option dontlognull
14     timeout connect 5000
15     timeout client 50000
16     timeout server 50000
17
18 frontend kubernetes-api
19     bind *:8443
20     mode tcp
21     option tcplog
22     default_backend kubernetes-masters
23
24 backend kubernetes-masters
25     mode tcp
26     balance roundrobin
27     server master1 172.16.0.91:6443 check
28     server master2 172.16.1.92:6443 check
29     server master3 172.16.1.93:6443 check
```

Reiniciar HAProxy para aplicar los cambios:

```
1 sudo systemctl restart haproxy
```

Instalar y configurar Keepalived

Instalar keepalived en cada nodo donde HAProxy se encuentra configurado

```
1 sudo apt update
2 sudo apt install keepalived
```

Configurar Keepalived para gestionar un VIP (Virtual IP) entre los nodos HAProxy.

Nodo 1 (Prioridad más alta - 172.16.1.91)

Archivo: `/etc/keepalived/keepalived.conf`

```
1 global_defs {
2     notification_email {
3     }
4     router_id LVS_DEVEL
5     vrrp_skip_check_adv_addr
6     vrrp_garp_interval 0
7     vrrp_gna_interval 0
8 }
9
10 vrrp_script chk_haproxy {
11     script "/usr/local/bin/check_haproxy.sh"
12     interval 2
13     weight 2
14 }
15
16 vrrp_instance VI_1 {
17     state MASTER
18     interface eth0
19     virtual_router_id 51
20     priority 100
21     advert_int 1
22     authentication {
23         auth_type PASS
24         auth_pass 5q9E6PLxSYYfE0N
25     }
26     unicast_src_ip 172.16.1.91      # The IP address of this machine
27     unicast_peer {
28         172.16.1.92                # The IP address of peer machines
29         172.16.1.93
30     }
31     virtual_ipaddress {
32         172.16.1.98
33     }
34     track_script {
35         chk_haproxy
36     }
37 }
```

Nodo 2 (Prioridad más alta - 172.16.1.92)

Archivo: `/etc/keepalived/keepalived.conf`

```
1 global_defs {
2     notification_email {
3     }
4     router_id LVS_DEVEL
5     vrrp_skip_check_adv_addr
6     vrrp_garp_interval 0
7     vrrp_gna_interval 0
8 }
9
10 vrrp_script chk_haproxy {
11     script "/usr/local/bin/check_haproxy.sh"
12     interval 2
```



```

13     weight 2
14 }
15
16 vrrp_instance VI_1 {
17     state BACKUP
18     interface eth0
19     virtual_router_id 51
20     priority 90
21     advert_int 1
22     authentication {
23         auth_type PASS
24         auth_pass 5q9E6PLxSYYfE0N
25     }
26     unicast_src_ip 172.16.1.91      # The IP address of this machine
27     unicast_peer {
28         172.16.1.92                # The IP address of peer machines
29         172.16.1.93
30     }
31     virtual_ipaddress {
32         172.16.1.98
33     }
34     track_script {
35         chk_haproxy
36     }
37 }

```

Nodo 3 (Prioridad más alta - 172.16.1.93)

Archivo: `/etc/keepalived/keepalived.conf`

```

1  global_defs {
2      notification_email {
3      }
4      router_id LVS_DEVEL
5      vrrp_skip_check_adv_addr
6      vrrp_garp_interval 0
7      vrrp_gna_interval 0
8  }
9
10 vrrp_script chk_haproxy {
11     script "/usr/local/bin/check_haproxy.sh"
12     interval 2
13     weight 2
14 }
15
16 vrrp_instance VI_1 {
17     state BACKUP
18     interface eth0
19     virtual_router_id 51
20     priority 80
21     advert_int 1
22     authentication {
23         auth_type PASS
24         auth_pass 5q9E6PLxSYYfE0N
25     }
26     unicast_src_ip 172.16.1.91      # The IP address of this machine
27     unicast_peer {
28         172.16.1.92                # The IP address of peer machines
29         172.16.1.93

```

```

30     }
31     virtual_ipaddress {
32         172.16.1.98
33     }
34     track_script {
35         chk_haproxy
36     }
37 }

```

Crear el script de verificación de HAProxy:

Archivo: `/usr/local/bin/check_haproxy.sh`

```

1  #!/bin/bash
2
3  if systemctl status haproxy | grep -q 'active (running)'; then
4      exit 0
5  else
6      exit 1
7  fi

```

Convertir el script en ejecutable:

```

1  sudo chmod +x /usr/local/bin/check_haproxy.sh

```

Reiniciar y habilitar Keepalived:

```

1  sudo systemctl enable keepalived
2  sudo systemctl restart keepalived

```

Inicialización de cluster

Master

1. Inicializar el nodo master principal (antes de inicializar el cluster es necesario tener en cuenta la configuración de alta disponibilidad):

```

sudo kubeadm init --control-plane-endpoint "<172.16.1.98>:8443" --upload-certs --pod-network-cidr=192.168.0.0/16 --cri-socket /var/run/containerd/containerd.sock -v=5

```

2. Configurar kubectl en el nodo master principal:

```

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

```

3. Guardar el comando de unión y los certificados:

```

kubeadm token create --print-join-command

```

4. **Unir nodos master adicionales:** En los otros nodos master, ejecute el comando de unión obtenido en el paso anterior con el flag `--control-plane`:

```

kubeadm init --control-plane-endpoint "172.16.1.98:8443" --pod-network-cidr=192.168.0.0/16 --cri-socket /var/run/containerd/containerd.sock --upload-certs -v=5

```


Workers

5. **Unir nodos worker:** En los nodos worker, ejecute el comando de unión obtenido en el paso anterior:

```

sudo kubeadm join <IP-del-worker-a-unir>:6443 --token <token> --disco

```

 Por favor reiniciar kubelet y containerd en todos los nodos con el fin de sincronizar y garantizar que los nodos

Instalar CNI (Calico)

El plugin de Calico se instala en el nodo master principal, los demás nodos heredarán las características a través de la proilferación ejecutada por el clúster, tanto en los nodos master como en los nodos worker. Calico proporciona la red de pods y la política de red necesaria para que los contenedores puedan comunicarse entre sí de manera segura y eficiente.

Para instalar Calico , ejecute los siguientes pasos desde el nodo master principal, una vez que el clúster Kubernetes esté inicializado.

1. Descargar y aplicar el manifiesto de Calico:

```
kubectl apply -f https://docs.projectcalico.org/manifests/calico.yaml
```

Este comando descarga y aplica el manifiesto YAML de Calico, que contiene todos los recursos necesarios para desplegar Calico en el clúster. Estos recursos incluyen:

- **DaemonSet de calico/node:** Este DaemonSet asegura que el componente `calico/node` se ejecute en todos los nodos del clúster.
- **Deployment de calico/kube-controllers:** Este componente se encarga de la sincronización de datos entre Kubernetes y Calico.
- **ConfigMap de Calico:** Contiene la configuración para los componentes de Calico.
- **Pod Security Policies y RBAC:** Configuraciones necesarias para asegurar que Calico tiene los permisos adecuados para operar.

Verificación de la Instalación

Después de aplicar el manifiesto, se recomienda verificar que todos los componentes de Calico están en funcionamiento:

1. Verificar los pods de Calico:

```
kubectl get pods -n kube-system -l k8s-app=calico-node
```

Este comando debe mostrar que un pod de `calico-node` está en ejecución en cada nodo del clúster.

2. Verificar el estado de los pods:

```
kubectl get pods -n kube-system
```

Asegúrate de que todos los pods de Calico están en estado `Running` .

3. Verificar la red de pods:

```
kubectl get nodes -o wide kubectl get pods -o wide --all-namespaces
```

Estos comandos permiten ver las direcciones IP asignadas a los nodos y a los pods para asegurar que Calico está gestionando correctamente la red.

Verificación del estado del cluster

1. Verificar el estado del clúster:

```
kubectl get nodes
```

2. Verificar que todos los nodos están en estado `Ready` :

```
kubectl get nodes
```

Si todos los nodos aparecen como `Ready` , la instalación de Kubernetes con tres nodos master y cuatro nodos worker se ha realizado con éxito.

Instalación y configuración de HA-Proxy para alta disponibilidad de Kubernetes

Es un balanceador de carga y servidor proxy conocido por sus capacidades de alto rendimiento. Desempeña un papel vital en la distribución eficiente del tráfico de red entrante a través de múltiples servidores o servicios backend, en este caso, se utilizará para proveer servicios de alta disponibilidad al cluster de Kubernetes.

Instalar paquetes necesarios en “master1” (en el nodo master principal):

```
1 sudo dnf install -y haproxy
```

Agregue la siguiente configuración al archivo “/etc/haproxy/haproxy.cfg”

```
1 global
2     log /dev/log      local0
3     log /dev/log      local1 notice
4     chroot /var/lib/haproxy
5     stats socket /run/haproxy/admin.sock mode 660 level admin
6     stats timeout 30s
7     user haproxy
8     group haproxy
9     daemon
10
11     # Default SSL material locations
12     ca-base /etc/ssl/certs
13     crt-base /etc/ssl/private
14
15     # See: https://ssl-config.mozilla.org/#server=haproxy&version=2.0.13&config=intermediate&guideline=5.6
16     ssl-default-bind-ciphers PROFILE=SYSTEM
17     ssl-default-bind-options no-ssl3
18
19 defaults
20     log          global
21     mode         http
22     option       httplog
23     option       dontlognull
24     timeout connect 5000
25     timeout client  50000
26     timeout server  50000
27     errorfile 400 /etc/haproxy/errors/400.http
28     errorfile 403 /etc/haproxy/errors/403.http
29     errorfile 408 /etc/haproxy/errors/408.http
30     errorfile 500 /etc/haproxy/errors/500.http
31     errorfile 502 /etc/haproxy/errors/502.http
32     errorfile 503 /etc/haproxy/errors/503.http
33     errorfile 504 /etc/haproxy/errors/504.http
34
35 frontend kubernetes-frontend
36     bind *:8443
37     option tcplog
38     mode tcp
39     default_backend kubernetes-backend
40
41 backend kubernetes-backend
42     mode tcp
43     balance roundrobin
44     option tcp-check
45     server master1 172.16.1.91:6443 check
```

```
46 server master2 172.16.1.92:6443 check
47 server master3 172.16.1.93:6443 check
```

Reinicie el servicio de haproxy

```
1 sudo systemctl restart haproxy
2 sudo systemctl enable haproxy
```

Secuencia de Apagado y Encendido del cluster de Kubernetes

Apagar/Detener un clúster es muy peligroso. Debe comprender completamente la operación y sus consecuencias. Haga una copia de seguridad de etcd antes de continuar.

Por lo general, se recomienda mantener los nodos uno por uno en lugar de reiniciar todo el clúster, pero para efecto práctico, en este documento, se describirá todo el proceso de apagado/reinicio.

Apagado

El orden de apagado debe ser estrictamente en el siguiente orden:

1. Ejecutar todos los pasos de apagado en los nodos esclavos/workers
2. Ejecutar todos los pasos de apagado todos los nodos máster.

Apagar los nodos (workers)

El primer paso es apagar debidamente cada uno de los nodos, para ello es necesario, vaciar el nodo ya que posiblemente tenga pods corriendo.

Ejecute los siguientes comandos desde el máster para detener la programación de pods y drenar los pods existentes en los nodos.

```
1 Ejecute desde los nodos máster:
2 -----
3 sudo kubectcl cordon <nombre del nodo>
4 sudo kubectcl drain <nombre del nodo> --ignore-daemonsets --delete-emptydir-data
5 sudo systemctl stop kubelet
6 sudo systemctl stop containerd
7
8 Ejemplo y explicación:
9 -----
10 Haga esto desde los nodos máster:
11 *****
12 sudo kubectcl cordon master-1
13 sudo kubectcl cordon master-2
14 sudo kubectcl cordon master-2
15 sudo kubectcl drain worker-1 --ignore-daemonsets --delete-emptydir-data
16 sudo kubectcl drain worker-2 --ignore-daemonsets --delete-emptydir-data
17 sudo kubectcl drain worker-3 --ignore-daemonsets --delete-emptydir-data
18 sudo kubectcl drain worker-4 --ignore-daemonsets --delete-emptydir-data
19 sudo kubectcl drain master-1 --ignore-daemonsets --delete-emptydir-data
20 sudo kubectcl drain master-2 --ignore-daemonsets --delete-emptydir-data
21 sudo kubectcl drain master-3 --ignore-daemonsets --delete-emptydir-data
22
23 Haga esto desde cada nodo (workers y master):
24 *****
25 sudo systemctl stop kubelet
26 sudo systemctl stop containerd
```

Si requiere apagar el nodo debido a la naturaleza del procedimiento el cual puede ser un mantenimiento programado de infraestructura, por favor ejecute el siguiente comando:

```
1 sudo shutdown -h now
```

Apagar los máster

Este procedimiento debe ser ejecutado con extremo cuidado y en un orden específico.

Ejecute los siguientes pasos para detener los servicios de cada máster y apagarlo (Uno a la vez).

1. Drenar y detener scheduling

```
sudo kubectl cordon <nombre_del_máster>
```

1. Detener el servicio de Kubelet:

```
sudo systemctl stop kubelet
```

2. Detener servicios de Containerd

```
sudo systemctl stop containerd
```

3. Apague el sistema

```
sudo shutdown -h now
```

Encendido

El orden de encendido debe ser estrictamente en el siguiente orden:

1. Ejecutar todos los pasos de encendido en el máster
2. Ejecutar todos los pasos de encendido en los nodos esclavos/workers, **este proceso debe ser uno a la vez.**

Inicio de los nodos máster

El primer paso es encender debidamente el máster, ejecute los siguientes comandos para iniciar los servicios del cluster.

1. Verifique que el swap esté deshabilitado, si no estuviera deshabilitado, por favor desactívelo de la siguiente manera: `swapoff -a`
2. Verifique los servicios de kubelet, containerd (Docker) ejecutando estos comandos:
 - a. `sudo systemctl status kubelet`
 - b. `sudo systemctl status containerd`
3. Verifique el estado de nodos, ejecutando el siguiente comando, en este momento sólo cada máster se mostrará en estado "Ready":
 - a. `sudo kubectl get nodes`
4. Active los el scheduling de pods ejecutando el siguiente comando desde el máster:
 - a. `sudo kubectl uncordon <nombre_del_máster>`
 - b. `sudo kubectl uncordon <nombre_de_nodo>`

Inicio de los nodos esclavos (workers)

Para encender los nodos debidamente, es recomendable ejecutar el proceso uno a la vez.

Ejecute los siguientes comandos para encender los nodos:

1. Verifique que el swap esté deshabilitado, si no estuviera deshabilitado, por favor desactívelo de la siguiente manera: `swapoff -a`
2. Verifique los servicios de kubelet, containerd (Docker) ejecutando estos comandos:
 - a. `sudo systemctl status kubelet`
 - b. `sudo systemctl status containerd`
3. Verifique desde el máster que el nodo se haya unido, para ello ejecute el siguiente comando desde el máster.
4. `sudo kubectl get nodes`

Nota: Observe que los workers aún se encuentran “Not Ready”, esto es porque vamos a iniciar los nodos uno a uno y adicionalmente, todos los nodos (incluyendo el máster) se encuentran en modo “SchedulingDisabled”, esto es porque se drenaron y se desactivaron las opciones de recibir pods con el fin de apagarlos limpiamente.

Ejecute los siguientes comandos desde el máster para reactivar la programación de pods en los nodos (esto incluye el mismo máster).

```
1 Ejecute lo siguiente desde los máster:
2 -----
3 sudo kubectl uncordon <nombre del nodo>
4
5 Ejemplo:
6 sudo kubectl uncordon master-1
7 sudo kubectl uncordon master-2
8 sudo kubectl uncordon master-3
9 sudo kubectl uncordon worker-1
10 sudo kubectl uncordon worker-2
11 sudo kubectl uncordon worker-3
12 sudo kubectl uncordon worker-4
```

Verifique el estado de los nodos y el master ejecutando el siguiente comando:

```
sudo kubectl get pods
```

La salida del comando anterior debe ser similar a la siguiente:

NAME	STATUS	ROLES	AGE
master-1	Ready	control-plane	10m
master-2	Ready	control-plane	10m
master-3	Ready	control-plane	10m
worker-1	Ready	<none>	10m
worker-2	Ready	<none>	10m
worker-3	Ready	<none>	10m
worker-4	Ready	<none>	10m

Troubleshooting durante el proceso de reinicio del cluster

La resolución de problemas de Kubernetes es el proceso de identificar, diagnosticar y resolver problemas en clústeres, nodos, pods o sus contenedores.

Definido de manera más amplia, la resolución de problemas de Kubernetes también incluye una gestión continua y eficaz de los fallos y la adopción de medidas para prevenir problemas en todos los componentes.

Problemas con pods atascados con status “Terminating”

Los pods atascados pueden ser un problema y ocasionar picos de performance, ya que no son detectados con el uso de comandos tradicionales de visualización de pods y namespaces.

Para verificar pods que no son visibles con el comando tradicional de “kubectl get pods”, utilice el comando siguiente:

```
kubectl get pods --all-namespaces -o wide
```

Este comando le permitirá ver si existen pods en proceso de borrado pero atascados con estado “Terminating” tal y como se puede apreciar en la siguiente imagen:

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED NODE	READINESS GATES
kong	bg-kong-kong-75fc9d79b6-f2xql	2/2	Terminating	0	3d4h	172.16.90.13	kubeworker1	<none>	<none>
kong	bg-kong-kong-75fc9d79b6-f7hnh	2/2	Terminating	0	3d4h	172.16.90.12	kubeworker1	<none>	<none>
kube-system	calico-kube-controllers-658d97c59c-mnn2t	1/1	Running	1 (5h31m ago)	2d23h	172.16.42.27	kubeworker	<none>	<none>
kube-system	calico-node-bxpqj	1/1	Running	7 (5h ago)	8d	192.168.0.26	kubeworker2	<none>	<none>
kube-system	calico-node-ljgcs	1/1	Running	8 (5h2m ago)	8d	192.168.0.25	kubeworker1	<none>	<none>
kube-system	calico-node-x4dfc	1/1	Running	10 (5h31m ago)	8d	192.168.0.24	kubeworker	<none>	<none>
kube-system	coredns-76f75df574-l6ktq	1/1	Running	1 (5h31m ago)	2d23h	172.16.42.25	kubeworker	<none>	<none>
kube-system	coredns-76f75df574-p4nq5	1/1	Running	1 (5h31m ago)	2d23h	172.16.42.26	kubeworker	<none>	<none>
kube-system	etcd-kubeworker	1/1	Running	11 (5h31m ago)	8d	192.168.0.24	kubeworker	<none>	<none>
kube-system	kube-apiserver-kubeworker	1/1	Running	11 (5h31m ago)	8d	192.168.0.24	kubeworker	<none>	<none>
kube-system	kube-controller-manager-kubeworker	1/1	Running	32 (5h31m ago)	8d	192.168.0.24	kubeworker	<none>	<none>
kube-system	kube-proxy-4bc1h	1/1	Running	11 (5h31m ago)	8d	192.168.0.24	kubeworker	<none>	<none>
kube-system	kube-proxy-cv648	1/1	Running	8 (5h ago)	8d	192.168.0.26	kubeworker2	<none>	<none>
kube-system	kube-proxy-f8c4q	1/1	Running	9 (5h2m ago)	8d	192.168.0.25	kubeworker1	<none>	<none>
kube-system	kube-scheduler-kubeworker	1/1	Running	30 (5h31m ago)	8d	192.168.0.24	kubeworker	<none>	<none>

Para terminar con los pods atascados, ejecute el siguiente comando:

```
1 Comando de borrado de pods atascados:
2 -----
3   kubectl delete pod <nombre_pod> --force --grace-period=0 -n <namespace>
4
5 Ejemplo con los pods atascados mostrado en la imagen anterior:
6   kubectl delete pod bg-kong-kong-75fc9d79b6-t7hnh --force --grace-period=0 -n kong
7   kubectl delete pod bg-kong-kong-75fc9d79b6-f2xql --force --grace-period=0 -n kong
```

Problemas después del reinicio:

Es posible que los nodos no se unan nuevamente al cluster, debido a algunas inconsistencias, si este fuera el caso, por favor ejecute el siguiente procedimiento:

- Asegúrese de que el nodo tiene el swap desactivado, si no es así, por favor desactívelo ejecutando el siguiente comando:
 - `swapoff -a`
 - Para que desactivado sea permanente, edite el archivo `/etc/fstab` y desactive la línea agregando el símbolo `#` a inicio de la línea de la configuración de swap como se muestra a continuación:
 - `#/swap.img none swap sw 0 0`
- Si el nodo no ha sido eliminado del clúster (solo drenado), no es necesario que ejecute **kubeadm join**, el nodo puede volver a unirse al máster ejecutando el siguiente comando desde el máster:
 - `sudo kubectl uncordon <nombre_del_nodo>`
 - Si el problema persiste, puede desligar el node y volverlo y posteriormente readjustarlo, para ello, ejecute los siguientes comandos:
 - `sudo kubeadm reset`
 - Este comando es el que ha entregado el máster durante la configuración inicial
 - `sudo kubeadm join --token <token> <control-plane-host>:<control-plane-port> --discovery-token-ca-cert-hash sha256:<hash>`
 - Ejemplo:
 - `kubeadm join 192.168.0.24:6443 --token dq654d.rcjx1edc7y1gnay5 \`
`--discovery-token-ca-cert-hash \`
`sha256:10f53ead9aee7a3b8f3bfdd79cc8b0c45f19c4099e85e5e917dda90bfb52b832`
 - Espere 30 segundos y verifique desde el máster con el siguiente comando:
 - `sudo kubectl get nodes`

Verificación del estado del cluster

Este proceso crucial para asegurar que todos los componentes del clúster estén funcionando correctamente y que los nodos estén disponibles para ejecutar cargas de trabajo.

Estado de los nodos

```
1 kubectl get nodes
```


NAME	STATUS	ROLES	AGE	VERSION
master-1	Ready	control-plane	10m	v1.29.0
master-2	Ready	control-plane	10m	v1.29.0
master-3	Ready	control-plane	10m	v1.29.0
worker-1	Ready	<none>	10m	v1.29.0
worker-2	Ready	<none>	10m	v1.29.0
worker-3	Ready	<none>	10m	v1.29.0
worker-4	Ready	<none>	10m	v1.29.0

Estado de todos los pods correspondientes al control del cluster

```
1 kubectl get pods -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
calico-kube-controllers-5b67c5c998-8fwhb	1/1	Running	0	5m
calico-node-4n5hj	1/1	Running	0	5m
calico-node-n8m1j	1/1	Running	0	5m
calico-node-qwv4l	1/1	Running	0	5m
calico-node-6r5mt	1/1	Running	0	5m
calico-node-7b9fw	1/1	Running	0	5m
calico-node-9m1jm	1/1	Running	0	5m
calico-node-9v4jn	1/1	Running	0	5m
etcd-master-1	1/1	Running	0	10m
etcd-master-2	1/1	Running	0	10m
etcd-master-3	1/1	Running	0	10m
kube-apiserver-master-1	1/1	Running	0	10m
kube-apiserver-master-2	1/1	Running	0	10m
kube-apiserver-master-3	1/1	Running	0	10m
kube-controller-manager-master-1	1/1	Running	0	10m
kube-controller-manager-master-2	1/1	Running	0	10m
kube-controller-manager-master-3	1/1	Running	0	10m
kube-proxy-worker-1	1/1	Running	0	10m
kube-proxy-worker-2	1/1	Running	0	10m
kube-proxy-worker-3	1/1	Running	0	10m
kube-proxy-worker-4	1/1	Running	0	10m
kube-scheduler-master-1	1/1	Running	0	10m
kube-scheduler-master-2	1/1	Running	0	10m
kube-scheduler-master-3	1/1	Running	0	10m

Verificar conectividad y estado de la red y conectividad

```
1 kubectl get nodes -o wide | awk 'NR==1{print $1 "\t" $2 "\t" $3 "\t" $5 "\t" $6 "\t" $7 "\t" $8 "\t" $9} NR>1{pri
```

NAME	STATUS	ROLES	VERSION	INTERNAL - IP	EXTERNAL - IP
master-1	Ready	control-plane	v1.29.0	172.16.1.91	<none>
master-2	Ready	control-plane	v1.29.0	172.16.1.92	<none>
master-3	Ready	control-plane	v1.29.0	172.16.1.93	<none>
worker-1	Ready	<none>	v1.29.0	172.16.1.94	<none>
worker-2	Ready	<none>	v1.29.0	172.16.1.95	<none>
worker-3	Ready	<none>	v1.29.0	172.16.1.96	<none>
worker-4	Ready	<none>	v1.29.0	172.16.1.97	<none>

Conclusión

Siguiendo los pasos descritos en este manual, se puede configurar un clúster de Kubernetes con alta disponibilidad en RedHat 9.3 utilizando containerd. Esta configuración asegura que el clúster esté preparado para manejar cargas de trabajo distribuidas con la redundancia necesaria para mantener la disponibilidad y estabilidad del sistema.