



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Escola Tècnica  
Superior d'Enginyeria  
Informàtica

Escuela Técnica Superior de Ingeniería Informática  
Universidad Politécnica de Valencia

## **Creación Cluster Kubernetes**

### **TRABAJO COS**

Master en Ingeniería Informática

*Autor:* Brian Valiente Rodenas

*Tutor:* Floreal Acebrón Linuesa

Curso 2024-2025



---

# CAPÍTULO 1

## Introducción

---

La presente memoria tiene como objetivo documentar el proceso de creación de un clúster Kubernetes utilizando dos métodos diferentes: Kubeadm y Ansible (sin Kubespray). A lo largo de este trabajo, se detallarán los pasos necesarios para configurar los entornos, las pruebas realizadas y las ampliaciones implementadas, con el fin de demostrar el funcionamiento del clúster en escenarios prácticos.

### 1.1 Motivación

---

La motivación principal detrás de este trabajo radica en la creciente adopción de Kubernetes como estándar de orquestación de contenedores en la industria tecnológica. Este proyecto permite explorar los fundamentos de Kubernetes, así como herramientas complementarias como Ansible para la automatización. Además, se busca aplicar configuraciones avanzadas como el uso de MetalLB y Metrics Server, almacenamiento persistente mediante NFS y escalado automático de aplicaciones HPA.

### 1.2 Objetivos

---

- Crear un clúster Kubernetes utilizando Kubeadm con un nodo master y dos nodos worker.
- Crear un clúster Kubernetes utilizando Ansible (sin Kubespray) con un nodo ansible, un nodo master y dos nodos worker.
- Implementar servidores HTTP (NGINX o Apache) que sirvan páginas PHP simples, desplegándolos con ficheros YAML.
- Configurar escalado automático mediante Horizontal Pod Autoscaler (HPA).
- Instalar MetalLB para habilitar servicios del tipo LoadBalancer.
- Configurar un nodo NFS para proporcionar almacenamiento persistente en el clúster.
- Garantizar que todos los servidores web sirvan la misma página almacenada en el nodo NFS.

## 1.3 Estructura de la memoria

---

La memoria se organiza de la siguiente manera:

- **Introducción:** Presenta los objetivos, motivaciones y estructura del trabajo.
- **Preparación Inicial:** Presenta los pasos previos para la realización de las actividades del trabajo.
- **Actividad 1:** Detalla la creación de un clúster Kubernetes utilizando Kubeadm, incluyendo la configuración de los nodos y las pruebas de funcionamiento.
- **Actividad 2:** Expone la creación de un clúster Kubernetes con Ansible, abarcando configuraciones avanzadas como el despliegue de servidores HTTP, escalado automático, MetalLB, Metrics Server y almacenamiento persistente mediante NFS.

---

## CAPÍTULO 2

# Preparación Inicial

---

Antes de sumergirnos en las actividades prácticas relacionadas con Kubernetes y Ansible, es esencial preparar una máquina virtual base. Este paso inicial es clave para simplificar y optimizar el proceso de configuración de los entornos necesarios para las tareas, permitiéndonos ahorrar tiempo y esfuerzo en futuras etapas.

### 2.1 Crear una Máquina Base

---

#### ¿Qué es la máquina base y por qué es importante?

La máquina base actúa como una plantilla preconfigurada que podremos clonar para crear múltiples máquinas virtuales rápidamente. Este enfoque nos garantiza un entorno uniforme y nos facilita la gestión de los recursos para actividades complejas.

#### Pasos para crear la máquina base

##### 1. Configuración inicial:

- Crea una máquina virtual con las siguientes características:
  - **Disco:** 10 GB.
  - **Sistema operativo:** AlmaLinux 9.5 (versión minimal).
  - **Particionado:** Usa la opción de autoparticionado para agilizar la instalación.

##### 2. Guardado de la máquina base:

- Una vez que completes la instalación:
  - Guarda esta máquina virtual.
  - Asígnale un nombre descriptivo, por ejemplo: `alma-9-base`.

#### Ventajas de usar una máquina base:

- **Ahorro de tiempo:** Configuración inicial única, reutilizable en múltiples escenarios.
- **Consistencia:** Todos los nodos creados desde esta base tendrán la misma configuración inicial.
- **Flexibilidad:** Útil tanto para Kubernetes como para Ansible, adaptándose según las necesidades de cada tarea.

## 2.2 Uso de la Máquina Base

---

**Para configurar el clúster de Kubernetes:**

- **Clonación:** Duplica la máquina base **tres veces**.
- **Asignación de funciones:**
  - Una máquina será el **nodo master**.
  - Las otras dos serán los **nodos worker**.

**Para la actividad de Ansible:**

- **Clonación:** Duplica la máquina base **cuatro veces**.
- **Asignación de funciones:**
  - Una máquina será el **nodo Ansible** (controlador).
  - Otra máquina será el **nodo master**.
  - Las restantes serán los **nodos worker**.

---

## CAPÍTULO 3

# Actividad 1

---

Esta guía detalla los pasos para configurar un clúster de Kubernetes en un entorno local utilizando kubeadm, con VirtualBox como plataforma de virtualización. Se incluye un nodo master y dos nodos worker.

### Fuentes usadas

---

- [How to Install Kubernetes on Rocky Linux 9 | AlmaLinux 9](#)
- [How to install Kubernetes on Linux \(AlmaLinux\)](#)
- [Cómo Instalar Cluster Kubernetes con kubeadm](#)
- [Kubernetes: Bootstrap de Cluster con Kubeadm](#)
- [Kubernetes: Instalar cluster Kubernetes con kubeadm](#)

### 3.1 1. Configuración Inicial del Laboratorio

---

#### 3.1.1. Especificaciones de las Máquinas Virtuales

- **Nodo Master:**
  - RAM: 2 GiB
  - CPU: 2 vCPU
  - Disco: 10 GiB (reservado dinámicamente)
  - NIC: Red NAT (192.168.1.6)
  - SO: AlmaLinux 9.4 Minimal
- **Nodos Worker:**
  - RAM: 1 GiB
  - CPU: 1 vCPU
  - Disco: 10 GiB (reservado dinámicamente)
  - NIC: Red NAT (192.168.1.7 y 192.168.1.8)
  - SO: AlmaLinux 9.4 Minimal

## 3.2 2. Configuración de la Red

---

Configuramos IP estáticas en cada máquina virtual utilizando `nmcli`. Estos pasos se realizan en **todas las máquinas**.

### 3.2.1. Paso 1: Configuración Manual de IP Estática, DNS y Gateway

Editamos o creamos el archivo de configuración de red:

```
1 nmcli con mod enp0s3 ipv4.address XXX.XXX.X.X/24 ipv4.method manual
2 nmcli con mod enp0s3 ipv4.gateway 192.168.1.1
3 nmcli con mod enp0s3 ipv4.dns 212.166.211.3
4 nmcli con down enp0s3
5 nmcli con up enp0s3
```

### 3.2.2. Paso 2: Eliminar IPs Dinámicas

En caso de que persistan direcciones asignadas por DHCP:

```
1 sudo ip addr del 192.168.1.X/24 dev enp0s3
```

### 3.2.3. Paso 3: Reinicio de NetworkManager

```
1 sudo systemctl restart NetworkManager
```

## 3.3 3. Configuración del Entorno Base

---

Estos pasos se realizan en **todas las máquinas**.

### 3.3.1. Paso 1: Configuración de Hostnames y Hosts

En el nodo master:

```
1 sudo hostnamectl set-hostname master
```

En los nodos worker:

```
1 sudo hostnamectl set-hostname workerX # worker1 o worker2
```

Añadimos las entradas en `/etc/hosts`:

```
1 cat <<EOF | sudo tee -a /etc/hosts
2 192.168.1.6 master
3 192.168.1.7 worker1
4 192.168.1.8 worker2
5 EOF
```

### 3.3.2. Paso 2: Deshabilitar el Intercambio

```
1 sudo swapoff -a
2 sudo sed -i '/swap/d' /etc/fstab
```



### 3.3.3. Paso 3: Configurar SELinux y Firewall

- Deshabilitamos SELinux:

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

- Configuramos el firewall:

- **En todos los nodos:** Habilitamos el servicio para que inicie automáticamente al arrancar:

```
1 sudo systemctl enable firewalld
2 sudo systemctl start firewalld
3 sudo systemctl status firewalld
```

- **Nodo Master:**

```
1 sudo firewall-cmd --permanent --add-port
   ={6443,2379,2380,10250,10251,10252,10257,10259,179}/tcp
2 sudo firewall-cmd --permanent --add-port=4789/udp
3 sudo firewall-cmd --reload
```

- **Nodos Worker:**

```
1 sudo firewall-cmd --permanent --add-port
   ={179,10250,30000-32767}/tcp
2 sudo firewall-cmd --permanent --add-port=4789/udp
3 sudo firewall-cmd --reload
```

### 3.3.4. Paso 4: Habilitar Módulos del Kernel

```
1 cat <<EOF | sudo tee /etc/modules-load.d/k8s.conf
2 br_netfilter
3 overlay
4 EOF
5
6 sudo modprobe br_netfilter
7 sudo modprobe overlay
8
9 cat <<EOF | sudo tee /etc/sysctl.d/k8s.conf
10 net.bridge.bridge-nf-call-iptables = 1
11 net.bridge.bridge-nf-call-ip6tables = 1
12 net.ipv4.ip_forward = 1
13 EOF
14
15 sudo sysctl --system
```

## 3.4 4. Instalación de Containerd

Estos pasos se realizan en **todas las máquinas**.

- Añadimos el repositorio de Docker:

```
1 sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

- Instalamos Containerd:

```
1 sudo dnf install -y containerd.io
```

- Configuramos Containerd:

```
1 containerd config default | sudo tee /etc/containerd/config.toml > /dev/null 2>&1
2 sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/containerd/config.toml
3 sudo systemctl restart containerd
4 sudo systemctl enable containerd
5 sudo systemctl status containerd
```

## 3.5 5. Instalación de Herramientas de Kubernetes

Estos pasos se realizan en **todas las máquinas**.

- Añadimos el 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
```

- Instalamos kubeadm, kubelet y kubectl:

```
1 sudo dnf install kubeadm kubelet kubectl kubernetes-cni --disableexcludes=kubernetes
2 sudo systemctl start kubelet
3 sudo systemctl status kubelet
```

La opción `-disableexcludes=kubernetes` indica a `dnf` que no excluya paquetes relacionados con Kubernetes durante la instalación. Esto es importante porque algunos repositorios pueden tener exclusiones para paquetes específicos, y la selección de esta opción asegura que esas exclusiones son ignoradas.

## 3.6 6. Inicialización del Clúster

- En el nodo master:

```
1 sudo kubeadm init --pod-network-cidr=172.16.0.0/12
```

- Configuramos acceso a kubectl:

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

- Guarda el comando kubeadm join:

```
1 sudo kubeadm token create --print-join-command > join
```

- Copia el archivo join desde master a los nodos worker:

```
1 ssh-keygen -t rsa
2 ssh-copy-id root@workerX
3 scp join root@workerX:~
```

- Únete al clúster desde los nodos worker:

```
1 sudo $(cat join)
```

## 3.7 7. Configuración de la Red de Pods - Calico

- En el nodo master:

```
1 curl -O https://docs.projectcalico.org/manifests/calico.yaml
2 vi calico.yaml
```

Ahora vamos a configurar Calico, que por defecto viene configurado contra la red 192.168.0.0/16, para que utilice la red que hemos comentado anteriormente (y sobre la que ya hemos iniciado nuestro cluster). Para ello buscaremos dentro del fichero calico.yaml descargado anteriormente el campo 'CALICO\_IPV4POOL\_CIDR' y le asignaremos el valor de red donde queramos que se vayan presentando nuestros pods al levantarse, que quedará así:

```
# The default IPv4 pool to create on startup if none exists. Pod IPs will be
# chosen from this range. Changing this value after installation will have
# no effect. This should fall within '--cluster-cidr'.
- name: CALICO_IPV4POOL_CIDR
  value: "172.16.0.0/12"
```

**Figura 3.1:** Editar manifiesto de Calico.  
Fuente: elaboración propia.

```
1 kubectl apply -f calico.yaml
```

## 3.8 8. Verificación del Clúster

Verificamos que los nodos estén listos y verifique el estado de los pods:

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

Deberías ver el nodo master y los dos workers con el estado Ready.

```
[root@master ~]# kubectl get nodes
NAME        STATUS    ROLES    AGE   VERSION
master      Ready    control-plane   17h   v1.29.11
worker1     Ready    <none>         16h   v1.29.11
worker2     Ready    <none>         16h   v1.29.11
```

**Figura 3.2:** Nodos del clúster listos.  
Fuente: elaboración propia.

Deberías ver los pods con el estado Running.

```
[root@master ~]# kubectl get pods -n kube-system
NAME                                READY   STATUS
calico-kube-controllers-658d97c59c-685s5  1/1     Running
calico-node-55jvs                      1/1     Running
calico-node-z97ks                      1/1     Running
calico-node-zzvpu                      1/1     Running
coredns-76f75df574-j4bvg              1/1     Running
coredns-76f75df574-z6r69              1/1     Running
etcd-master                           1/1     Running
kube-apiserver-master                  1/1     Running
kube-controller-manager-master         1/1     Running
kube-proxy-5vx22                      1/1     Running
kube-proxy-8lplk                      1/1     Running
kube-proxy-98wqm                      1/1     Running
kube-scheduler-master                  1/1     Running
```

**Figura 3.3:** Pods kube-system preparados.  
Fuente: elaboración propia.

## 3.9 Pruebas del Clúster

En el nodo master:

```
1 kubectl create deployment nginx --image=nginx
2 kubectl expose deployment nginx --port=80 --type=NodePort
3 kubectl get services
```

Accede al servicio en el puerto asignado. En este caso, la prueba es bastante rudimentaria. En nuestro caso, al hacer el despliegue mediante esa orden (sin hacer uso de archivos YAML, aunque los utilizaremos más adelante), nuestro servicio se ha asignado al nodo `worker2`. Si hacemos una petición `curl` podemos ver la página de prueba de nginx "Welcome". A continuación se muestra el proceso:

```
[root@master ~]# kubectl get svc nginx
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
nginx     NodePort    10.111.0.217   <none>          80:32091/TCP   22h

[root@master ~]# kubectl get services
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
kubernetes ClusterIP  10.96.0.1      <none>          443/TCP         40h
nginx     NodePort    10.111.0.217   <none>          80:32091/TCP   22h

[root@master ~]# kubectl get deployments
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
nginx     1/1     1             1           22h

[root@master ~]# kubectl get pods -o wide
NAME      READY   STATUS    RESTARTS   AGE   IP            NODE
nginx-7854ff8877-vzurf  1/1     Running    2 (112s ago)  22h   172.31.233.197 worker2

[root@master ~]# curl http://192.168.1.8:32091
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

**Figura 3.4:** Despliegue de página web Nginx de prueba.  
Fuente: elaboración propia.

¡Hemos completado la configuración de nuestro clúster Kubernetes!



---

## CAPÍTULO 4

# Actividad 2

---

Esta guía detalla los pasos para configurar un clúster de Kubernetes en un entorno local utilizando Ansible, con VirtualBox como plataforma de virtualización. Se incluye un nodo ansible, un nodo master y dos nodos worker.

### 4.1 Introducción

---

Hemos creado un repositorio en GitHub para programar los playbooks de Ansible, ya que las máquinas virtuales tienen una versión minimal sin interfaz gráfica. Para facilitar la edición constante de archivos, usamos el repositorio y ejecutamos un `git pull` desde la máquina virtual donde se tiene el repositorio clonado. Dado que el `git pull` requiere autenticación, generé un token desde GitHub, lo guardé en un archivo de texto llamado `pass`, y cada vez que realizaba cambios en los archivos desde mi máquina host, hacía un push a la rama `main`. Posteriormente, en la máquina virtual, ejecutaba el siguiente comando:

```
1 git pull https://brivaro:$(cat pass)@github.com/brivaro/ansible
```

### Fuentes usadas

---

- [Automate Kubernetes Cluster Deployment with Ansible on CentOS Stream 8](#)
- [Cómo Instalar Kubernetes en Linux con Ansible](#)
- [Ansible + Kubernetes](#)
- [Kubernetes Setup Using Ansible Script](#)
- [Install Kubernetes on RockyLinux](#)
- [Repositorio Ansible Brian Valiente Rodenas](#)

### 4.2 Configuración inicial del laboratorio

---

#### 4.2.1. Especificaciones de las máquinas virtuales

- **Nodo Ansible:**

- **RAM:** 2 GiB
  - **CPU:** 2 vCPU
  - **Disco:** 10 GiB (reservado dinámicamente)
  - **NIC:** Red NAT (192.168.1.5)
  - **SO:** AlmaLinux 9.4 Minimal
- **Nodo Master:**
    - **RAM:** 2 GiB
    - **CPU:** 2 vCPU
    - **Disco:** 10 GiB (reservado dinámicamente)
    - **NIC:** Red NAT (192.168.1.6)
    - **SO:** AlmaLinux 9.4 Minimal
  - **Nodos Worker:**
    - **RAM:** 1 GiB
    - **CPU:** 1 vCPU
    - **Disco:** 10 GiB (reservado dinámicamente)
    - **NIC:** Red NAT (192.168.1.7 y 192.168.1.8)
    - **SO:** AlmaLinux 9.4 Minimal

## 4.3 Instalación de Ansible en el nodo Ansible

---

### 4.3.1. Paso 1: Actualizar el sistema

Asegúrate de que el sistema esté actualizado:

```
1 sudo dnf update -y
```

### 4.3.2. Paso 2: Instalar repositorios y Ansible

1. Habilita el repositorio EPEL y luego instala Ansible:

```
1 sudo dnf install -y epel-release
2 sudo dnf install -y ansible
```

2. Verifica la instalación:

```
1 ansible --version
```

## 4.4 Configuración inicial del nodo Ansible

---

### 4.4.1. Paso 3: Configurar SSH sin contraseña

1. Genera una clave SSH:

```
1 ssh-keygen -t rsa
```



(Presiona Enter para usar la ubicación por defecto y deja la contraseña vacía).

2. Copia la clave pública a los nodos master y worker (antes de esto, asegúrate de que tienen la IP correspondiente, añádela en cada nodo como lo hicimos en la parte de kubeadm):

```
1  ssh-copy-id root@192.168.1.6  # Master
2  ssh-copy-id root@192.168.1.7  # Worker1
3  ssh-copy-id root@192.168.1.8  # Worker2
```

3. Verifica que puedes conectarte sin contraseña:

```
1  ssh root@192.168.1.6
```

**NOTA:** asegúrate de que el número de CPUs del master sea 2.

## 4.5 Configuración del inventario de Ansible

### 4.5.1. Paso 4: Definir los nodos del clúster

Crea un archivo `inventory.ini` para definir los nodos del clúster:

```
1  [master]
2  master ansible_host=192.168.1.6
3
4  [workers]
5  worker1 ansible_host=192.168.1.7
6  worker2 ansible_host=192.168.1.8
7
8  [k8s:children]
9  master
10 workers
11
12 [all:vars]
13 ansible_user=root
14 ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Guarda este archivo en el directorio de trabajo o añádelo al final del archivo `hosts` de ansible para que sea accesible. Puedes hacerlo con el siguiente comando:

```
1  cat inventory.ini >> /etc/ansible/hosts
```

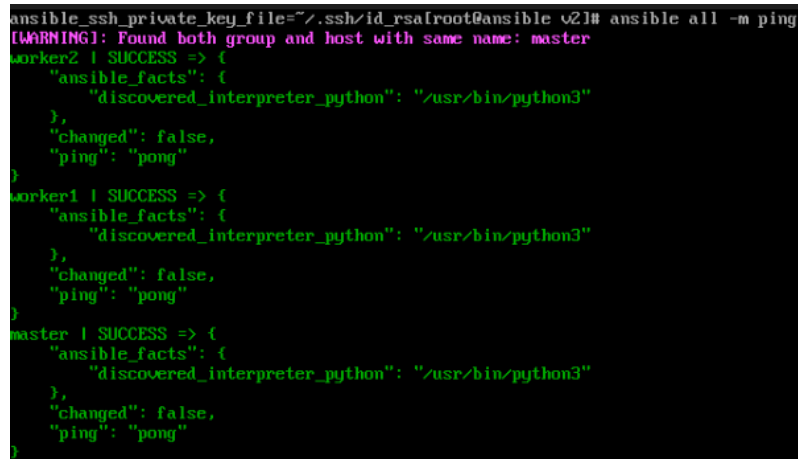
De esta forma, definimos los nodos del clúster. Cuando apliquemos los playbooks, tanto si usamos el archivo `.ini` o los `hosts` de Ansible para indicar a qué nodos aplicar los cambios, será válido.

### 4.5.2. Paso 5: Validar conexión SSH con Ansible

Prueba la conectividad entre el nodo Ansible y los nodos del clúster:

```
1 ansible all -m ping
```

Si todo está configurado correctamente, deberías recibir un mensaje SUCCESS para cada nodo:



```
ansible_ssh_private_key_file=~/.ssh/id_rsa[root@ansible ~]# ansible all -m ping
[WARNING]: Found both group and host with same name: master
worker2 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
worker1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
master | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": false,
  "ping": "pong"
}
```

Figura 4.1: Prueba de conexión.  
Fuente: elaboración propia.

## 4.6 Creación y ejecución de los playbooks

### 4.6.1. Paso 6: Crear playbooks para automatizar la instalación

Los playbooks de Ansible se utilizan para automatizar las tareas repetitivas y garantizar que todos los nodos del clúster se configuren de la misma manera. En este caso, crearemos varios playbooks para la instalación y configuración de Kubernetes en los nodos del clúster. Los playbooks incluirán las siguientes tareas:

- Configuración inicial de los nodos (deshabilitar swap, configurar SELinux, etc.).
- Instalación de Containerd como el runtime de contenedores.
- Instalación de los componentes de Kubernetes: kubeadm, kubelet, kubectl.
- Inicialización del clúster en el nodo master.
- Configuración de la red del clúster (se utilizará Calico como solución de red).
- Unión de los nodos worker al clúster.
- Configuración de Metallb, Metrics Server y despliegue de una página web personalizada (pv, pvc, deployment, service y hpa).

#### Estructura básica de un playbook (ejemplo.yaml)

A continuación se muestra un ejemplo de la estructura básica de un playbook de Ansible para realizar tareas simples en todos los nodos:

```
1 - hosts: all
2   tasks:
3     - name: Saludar
4       command: echo "Hola a todos"
```

Este playbook ejecuta un comando simple de echo en todos los nodos definidos en el inventario.

#### 4.6.2. Paso 7: Ejecutar los playbooks

Una vez que hayas creado los playbooks, puedes ejecutarlos utilizando el comando `ansible-playbook`. Asegúrate de tener configurado el inventario correctamente para que Ansible sepa a qué nodos dirigirse.

##### Ejecutar playbook con el inventario

Para ejecutar un playbook utilizando el inventario `inventory.ini` que hemos creado previamente, usa el siguiente comando:

```
1 ansible-playbook -i inventory.ini ejemplo.yaml
```

Este comando ejecutará el playbook `ejemplo.yaml` en los nodos especificados en el archivo `inventory.ini`.

##### Ejecutar playbook con el archivo de hosts global

Alternativamente, si prefieres utilizar el archivo de hosts global de Ansible (`/etc/ansible/hosts`), puedes ejecutar el playbook con el siguiente comando:

```
1 ansible-playbook -i /etc/ansible/hosts ejemplo.yaml
```

Esto hará lo mismo que el comando anterior, pero con el archivo global de hosts en lugar del archivo `inventory.ini` personalizado.

## 4.7 Configuración inicial

Para realizar la configuración inicial del clúster, ejecuta el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 1-config.yml
```

Este playbook realiza una serie de tareas esenciales para preparar los nodos del clúster Kubernetes. A continuación, se describen las principales configuraciones realizadas:

#### ■ Configuración de red:

- Se establece la dirección IP, puerta de enlace y servidores DNS utilizando comandos `nmcli`.
- Reinicia el servicio `NetworkManager` para aplicar los cambios.

#### ■ Hostname y archivo `/etc/hosts`:

- Configura el hostname de cada nodo según el inventario.

- Actualiza el archivo `/etc/hosts` con las direcciones IP y nombres de los nodos del clúster.
- **Deshabilitación de intercambio de memoria (swap):**
  - Desactiva el intercambio de memoria (swap) y lo elimina del archivo `/etc/fstab`.
- **Configuración de SELinux:**
  - Cambia SELinux al modo permisivo tanto de forma temporal como permanente en el archivo de configuración.
- **Configuración del Firewall:**
  - Para el nodo master, se abren los puertos necesarios para Kubernetes, incluyendo los utilizados por kube-apiserver, etcd y otros componentes principales.
  - En los nodos worker, se configuran los puertos necesarios para la comunicación con el clúster y las aplicaciones expuestas.
- **Cargar módulos del kernel:**
  - Carga los módulos `br_netfilter` y `overlay`, necesarios para el funcionamiento de containerd y la red de Kubernetes.
- **Configuraciones de sysctl:**
  - Habilita opciones de reenvío de paquetes (IP forwarding) y el paso de tráfico a través del puente (bridge) para soportar la red de Kubernetes.

Este playbook asegura que cada nodo esté preparado correctamente para iniciar el clúster Kubernetes. A continuación, se muestra un fragmento de las tareas definidas:

```

1  - name: Configuración inicial de nodos Kubernetes
2  hosts: all
3  become: true
4  tasks:
5  - name: Configurar dirección IP y método de red como manual
6  command: nmcli con mod enp0s3 ipv4.address {{ ansible_host }}/24 ipv4.method
      manual
7
8  - name: Deshabilitar intercambio de memoria
9  shell: |
10 swapoff -a
11 sed -i '/ swap / s/^^(.*)$/#\1/g' /etc/fstab
12
13 - name: Configurar SELinux en modo permisivo
14 command: setenforce 0
15 ignore_errors: true

```

Para obtener más detalles sobre las configuraciones específicas, consulta el contenido completo del playbook en mi repositorio.

#### 4.7.1. Instalación de containerd y Kubernetes

Para instalar containerd y los componentes de Kubernetes (kubeadm, kubelet, kubectl), ejecuta el siguiente playbook:

```
1  ansible-playbook -i inventory.ini 2-installation.yml
```

El playbook realiza los siguientes pasos:

- **Instalación de Containerd:** Se configura el repositorio de Docker, se instala containerd, y se ajusta su configuración para que utilice SystemdCgroup. Posteriormente, se reinicia y habilita el servicio de containerd.
- **Configuración del repositorio de Kubernetes:** Se añade el repositorio necesario para descargar kubeadm, kubelet, y kubectl.
- **Instalación de herramientas de Kubernetes:** Se instalan las herramientas esenciales (kubeadm, kubelet, y kubectl) junto con sus dependencias.
- **Habilitación de kubelet:** Finalmente, se habilita y verifica que el servicio kubelet esté activo.

A continuación, se muestra el contenido del playbook:

```
1  - name: Instalar Containerd y Herramientas de Kubernetes
2  hosts: all
3  become: true
4  tasks:
5
6  # --- Instalacion de Containerd ---
7  - name: Repositorio de Docker para Containerd
8    command: dnf config-manager --add-repo https://download.docker.com/linux/
              centos/docker-ce.repo
9
10 - name: Configurar Containerd
11   shell: |
12     sudo dnf install -y containerd.io
13     sudo mkdir -p /etc/containerd
14     sudo containerd config default | sudo tee /etc/containerd/config.toml >/dev/
              null 2>&1
15     sudo sed -i 's/SystemdCgroup \= false/SystemdCgroup \= true/g' /etc/
              containerd/config.toml
16     sudo systemctl restart containerd
17     sudo systemctl enable containerd
18
19 # --- Instalacion de Herramientas de Kubernetes ---
20 - name: Repositorio de Kubernetes
21   copy:
22     dest: /etc/yum.repos.d/kubernetes.repo
23     content: |
24       [kubernetes]
25       name=Kubernetes
26       baseurl=https://pkgs.k8s.io/core:/stable:/v1.32/rpm/
27       enabled=1
28       gpgcheck=1
29       gpgkey=https://pkgs.k8s.io/core:/stable:/v1.32/rpm/repodata/repomd.xml.key
30       exclude=kubelet kubeadm kubectl cri-tools kubernetes-cni
31
32 - name: Instalar kubeadm, kubelet y kubectl
33   command: yum install -y kubelet kubeadm kubectl kubernetes-cni --
              disableexcludes=kubernetes
34
35 - name: Iniciar y habilitar kubelet
36   command: systemctl enable --now kubelet
```

### 4.7.2. Inicialización del Cluster, unión de los workers y Calico

Para inicializar el clúster de Kubernetes, unir los nodos `worker` y configurar la red con Calico, ejecuta los siguientes playbooks:

```
1 ansible-playbook -i inventory.ini 3-inicluster.yml
```

El playbook realiza las siguientes tareas principales:

- **Inicialización del nodo maestro:** Se ejecuta el comando `kubeadm init` en el nodo maestro para inicializar el plano de control del clúster. Este proceso configura el nodo como maestro y registra la salida del comando para los siguientes pasos.
- **Configuración de `kubect1`:** Se configuran las credenciales de administración para que el usuario `root` pueda interactuar con el clúster usando `kubect1`.
- **Generación del comando de unión:** Se genera el comando que los nodos `worker` utilizarán para unirse al clúster.
- **Distribución del comando de unión:** El comando generado se copia a los nodos `worker` para que puedan ejecutarlo y unirse al clúster.
- **Unión de nodos `worker`:** Los nodos `worker` ejecutan el comando de unión para integrarse al clúster de Kubernetes.

A continuación, se muestra el contenido del playbook:

```
1 - name: Inicializacion del Cluster y Configuracion de Calico
2   hosts: master
3   become: true
4   tasks:
5
6   - name: Inicializar Kubernetes en el nodo master
7     command: kubeadm init --control-plane-endpoint=master # --pod-network-cidr
8               =192.168.0.0/16 # 172.16.0.0/12
9     register: kubeadm_init_output
10
11  - name: Configurar kubect1 para usuario root
12    shell: |
13      mkdir -p $HOME/.kube &&
14      cp /etc/kubernetes/admin.conf $HOME/.kube/config &&
15      chown $(id -u):$(id -g) $HOME/.kube/config
16
17  - name: Guardar comando de union
18    shell: kubeadm token create --print-join-command > /root/join
19
20  - name: Copiar comando join al nodo Ansible
21    fetch:
22      src: /root/join
23      dest: /tmp/join
24      flat: yes
25
26  - name: Copiar el comando join a los nodos worker
27    hosts: workers
28    become: true
29    tasks:
30
31  - name: Copiar archivo join desde el nodo Ansible
32    copy:
33      src: /tmp/join
34      dest: /tmp/join
```

```
34 |
35 | - name: Unirse al cluster de Kubernetes
36 | shell: sudo $(cat /tmp/join)
```

El siguiente playbook configura la red del clúster de Kubernetes utilizando Calico, un controlador de red de contenedores (CNI) ampliamente utilizado para la gestión de redes y políticas de seguridad en Kubernetes.

```
1 | ansible-playbook -i inventory.ini 4-calico.yml
```

A continuación, se explican las tareas principales realizadas por el playbook:

- **Descarga del archivo de configuración de Calico:** Se utiliza `get_url` para descargar el manifiesto YAML oficial de Calico desde su repositorio de GitHub. Este archivo contiene todas las configuraciones necesarias para implementar Calico en el clúster.
- **Aplicación del manifiesto de Calico:** Se aplica el archivo YAML descargado usando el comando `kubectl apply`. Esto inicia el despliegue de los componentes de Calico en el clúster, configurando la red de pods y asegurando la conectividad entre los nodos.


El contenido del playbook es el siguiente:

```
1 | - name: Configuración de la Red (Calico)
2 |   hosts: master # Solo en el nodo master
3 |   become: true
4 |   tasks:
5 |   - name: Descargar el archivo de configuración de Calico
6 |     get_url:
7 |     url: https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests
8 |         /calico.yaml
9 |     dest: /tmp/calico.yaml
10 |
11 | - name: Aplicar la configuración de Calico
    shell: kubectl apply -f /tmp/calico.yaml
```

### 4.7.3. Verificar el estado de los pods

Una vez que los playbooks se hayan ejecutado correctamente, puedes comprobar que todos los pods estén listos con el siguiente comando:

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



NAME	READY	STATUS
calico-kube-controllers-658d97c59c-rjhpv	1/1	Running
calico-node-5wpjt	1/1	Running
calico-node-b4sq5	1/1	Running
calico-node-bxjwc	1/1	Running
coredns-76f75df574-66mnh	1/1	Running
coredns-76f75df574-nx7gj	1/1	Running
etcd-master	1/1	Running
kube-apiserver-master	1/1	Running
kube-controller-manager-master	1/1	Running
kube-proxy-27vhp	1/1	Running
kube-proxy-8bhfrf	1/1	Running
kube-proxy-fq869	1/1	Running
kube-scheduler-master	1/1	Running

**Figura 4.2:** Comprobación de pods en el clúster.  
Fuente: elaboración propia.

Deberías ver el nodo master y los dos nodos worker con el estado Running.

¡Hemos completado la configuración de nuestro clúster Kubernetes como se tenía previamente!

### NOTA IMPORTANTE

Si a lo largo de la implementación surge algún problema y utilizas instantáneas previas para restaurar el estado de algunas máquinas, es posible que el reloj interno de las mismas haya quedado desfasado. Esto puede causar problemas con los certificados. Para solucionarlo, sigue estos pasos:

1. **Forzar una actualización del tiempo:** Si la sincronización no ocurre de inmediato, puedes forzarla manualmente con el siguiente comando:

```
1 chronyc makestep
```

2. **Confirma la sincronización NTP:** Activa nuevamente la sincronización NTP utilizando:

```
1 timedatectl set-ntp true
```

3. **Verifica el estado de la sincronización:** Comprueba si la sincronización está activa con el comando:

```
1 timedatectl
```

Con estos pasos, puedes asegurarte de que el reloj de las máquinas está correctamente sincronizado.



#### 4.7.4. Sincronización del Clúster con NFS

Para configurar la sincronización del clúster utilizando NFS, se utiliza el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 1-nfs.yml
```

El playbook realiza las siguientes tareas principales para habilitar la sincronización del almacenamiento:

```
1 - name: Configuración inicial de nodos con NFS
2 hosts: all
3 become: true
4 tasks:
5 - name: Configurar archivo /etc/hosts
6   lineinfile:
7     path: /etc/hosts
8     line: "{{ item }}"
9     state: present
10  with_items:
11    - "192.168.1.6 master"
12    - "192.168.1.7 worker1"
13    - "192.168.1.8 worker2"
14    - "192.168.1.9 nas nfs nfs-server"
15
16 - name: Instalar paquetes adicionales (cliente NFS)
17   package:
18     name: nfs-utils
19     state: present
20
21 - name: Crear directorios para montaje
22   file:
23     path: "{{ item }}"
24     state: directory
25   with_items:
26     - /mnt/web
27     - /mnt/post
```

- Configura el archivo `/etc/hosts` en cada nodo para que puedan resolverse entre sí.
- Instala el paquete `nfs-utils`, necesario para usar NFS.
- Crea los directorios locales (`/mnt/web` y `/mnt/post`) para montar los sistemas de archivos exportados.
- Configura y monta automáticamente los directorios exportados desde el nodo NAS, garantizando la sincronización.
- Realiza pruebas en el directorio `/mnt/web` para asegurar el acceso correcto.

#### 4.7.5. Configuración previa a instalar MetalLB

Antes de instalar MetalLB, es necesario configurar el clúster para habilitar `strictARP` y el modo `IPVS` en `kube-proxy`. Para ello, se utiliza el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 1-metallb.yml
```

El siguiente playbook realiza los ajustes necesarios para esta configuración:

```

1  - name: Configurar strictARP y modo IPVS en kube-proxy
2  hosts: master
3  become: yes
4  tasks:
5
6  - name: Editar ConfigMap kube-proxy para habilitar strictARP y modo IPVS
7  shell: |
8  kubectl get configmap kube-proxy -n kube-system -o yaml | \
9  sed -e "s/strictARP: false/strictARP: true/" -e "s/mode: \".*\"/mode: \"ipvs
    \"/" | \
10 kubectl apply -f - -n kube-system
11 register: apply_result
12
13 - name: Mostrar mensaje de resultado
14 debug:
15 msg: "Resultado de la aplicacion: {{ apply_result.stdout }}"

```

- El comando edita el ConfigMap de kube-proxy para activar el parámetro strictARP y configurar el modo IPVS, optimizando la gestión de la red en el clúster.
- Utiliza sed para modificar los valores en el archivo YAML y aplica los cambios de forma dinámica.
- Al finalizar, muestra un mensaje con el resultado de la aplicación, confirmando que los cambios se realizaron correctamente.

#### 4.7.6. Instalación de MetalLB

Para instalar y configurar MetalLB en Kubernetes (modo Layer 2), se utiliza el siguiente playbook:

```

1  ansible-playbook -i inventory.ini 3-metallb.yml

```

El siguiente fragmento del playbook realiza la instalación y configuración de MetalLB:

```

1  - name: Instalacion y configuracion de MetalLB en Kubernetes (Layer 2)
2  hosts: master
3  become: true
4  tasks:
5  - name: Descargar manifiestos de MetalLB
6  get_url:
7  url: https://raw.githubusercontent.com/metallb/metallb/v0.14.9/config/
    manifests/metallb-native.yaml
8  dest: /tmp/metallb.yaml
9
10 - name: Aplicar manifiestos de MetalLB
11 shell: kubectl apply -f /tmp/metallb.yaml
12
13 - name: Eliminar webhook
14 command: kubectl delete validatingwebhookconfigurations.admissionregistration
    .k8s.io metallb-webhook-configuration

```

- Primero, se descargan los manifiestos de MetalLB desde el repositorio oficial.
- Luego, se aplican estos manifiestos para instalar MetalLB en el clúster de Kubernetes.

- Finalmente, es esencial eliminar el webhook metallb-webhook-configuration, ya que si no se elimina, nos causa problemas al aplicar la IP Pool de direcciones disponibles para el balanceador de carga de MetalLB. Esto se logra con el comando `kubectl delete validatingwebhookconfigurations...`

```
[root@master ~]# kubectl get pods -n metallb-system
NAME                                READY   STATUS    RESTARTS   AGE
controller-7499d4584d-4b2s6        1/1     Running   0           7m11s
speaker-9425b                       1/1     Running   0           7m11s
speaker-sgpn8                       1/1     Running   0           7m11s
speaker-zx59b                       1/1     Running   0           7m11s
```

**Figura 4.3:** Comprobación de pods metallb-system  
Fuente: elaboración propia.

#### 4.7.7. Aplicar IPPool

Para configurar el IPPool de MetalLB, se utiliza el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 4-ippool.yml
```

El playbook realiza los siguientes pasos:

```
1 - name: Aplicar configuracion de Metallb IPPool
2 hosts: master
3 become: true
4 tasks:
5 - name: Copiar ippool.yaml al master
6 copy:
7   src: ~/ansible/nfs/web/ippool.yaml # Ruta en el nodo Ansible
8   dest: /tmp/ippool.yaml # Ruta en el master
9
10 - name: Aplicar IP pool
11 shell: kubectl apply -f /tmp/ippool.yaml
```

El archivo `ippool.yaml` contiene la configuración de un `IPAddressPool` y una `L2Advertisement` que permite a MetalLB gestionar un conjunto de direcciones IP para el balanceo de carga. El contenido de este archivo es el siguiente:

```
1 apiVersion: metallb.io/v1beta1
2 kind: IPAddressPool
3 metadata:
4   name: external-pool
5   namespace: metallb-system
6 spec:
7   addresses:
8   - 192.168.1.10-192.168.1.15
9   ---
10  apiVersion: metallb.io/v1beta1
11  kind: L2Advertisement
12  metadata:
13    name: external-advertisement
14    namespace: metallb-system
15  spec:
16    ipAddressPools:
17    - external-pool
```

En resumen:

- Se copia el archivo `ippool.yaml` desde la máquina Ansible al nodo master.

- Se aplica la configuración del IP Pool con el comando `kubectl apply`.
- El archivo `ippool.yaml` define un rango de direcciones IP (192.168.1.10-192.168.1.15) que MetalLB podrá utilizar para asignar direcciones IP a los servicios.
- También se configura un `L2Advertisement` que anuncia este rango de IPs a través de MetalLB.

#### 4.7.8. Instalar Metric Server

El Metric Server es un componente importante de Kubernetes que recopila métricas sobre los recursos utilizados en el clúster, como CPU y memoria. A continuación, se proporcionan algunos enlaces útiles para instalar y solucionar problemas con Metric Server:

- [Fix "metrics API not available" error in Kubernetes](#) - Una guía para solucionar errores comunes relacionados con la API de métricas.
- [Gist de configuración para Metric Server](#) - Un ejemplo práctico de configuración para instalar el Metric Server.
- [Issue en GitHub \(comentario clave\)](#) - En este hilo se resuelve un problema específico relacionado con el parámetro `nodeName`, que fue clave para solucionar nuestro error.

Para configurar la instalación de Metrics Server en el clúster, se utiliza el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 5-metricserver.yaml
```

Este playbook realiza los siguientes pasos para instalar y configurar el Metrics Server en el clúster de Kubernetes:

```
1 - name: Instalacion y configuracion de Metrics Server en namespace metallb-
  system
2 hosts: master
3 become: true
4 tasks:
5 - name: Descargar manifiesto de Metric Server
6 copy:
7 src: ~/ansible/nfs/web/componentsMetricsServer.yaml # Ruta en el nodo
  Ansible
8 dest: /tmp/components.yaml # Ruta en el master
9
10 - name: Desplegar Metrics Server
11 command: "kubectl apply -f /tmp/components.yaml"
```

El manifiesto de Metric Server se descarga desde GitHub y se añade a nuestro repositorio para poder editarlo. Se hicieron las siguientes modificaciones esenciales en el archivo YAML para garantizar que las métricas se recopilen correctamente:

```
1 spec:
2 hostNetwork: true
3 nodeName: "master" # El Metric Server debe estar en el Control Plane
4 containers:
5 - args:
6 - --cert-dir=/tmp
7 - --secure-port=4443
8 - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
9 - --kubelet-use-node-status-port
10 - --metric-resolution=15s
```

```

11 command:
12 - /metrics-server
13 - --kubelet-insecure-tls
14 - --kubelet-preferred-address-types=InternalIP
15 image: registry.k8s.io/metrics-server/metrics-server:v0.7.2
16 imagePullPolicy: IfNotPresent
17 livenessProbe:
18   failureThreshold: 3
19   httpGet:
20     path: /livez
21     port: https
22     scheme: HTTPS
23     periodSeconds: 10
24   name: metrics-server
25   ports:
26     - containerPort: 4443

```

Explicación de las modificaciones:

- Se cambió el puerto de escucha a 4443, asegurando que el Metric Server utilice un puerto seguro para las métricas.
- Se configuró `hostNetwork: true`, lo cual permite que el Metric Server use la red del host para la comunicación.
- Se agregó la variable `nodeName: "master"` porque el Metric Server debe ejecutarse en el Control Plane, ya que si se ejecuta en otro nodo no podrá acceder al API de métricas.
- También se añadieron parámetros como `-kubelet-insecure-tls` para permitir la conexión al kubelet sin verificar el certificado TLS.

Comprobamos con `kubectl top nodes/pods` las métricas:

```

[root@base ~]# kubectl top nodes
NAME      CPU(cores)   CPU%   MEMORY(bytes)   MEMORY%
master    443m         14%    1618Mi           86%
worker1    474m         47%    809Mi            43%
worker2    263m         26%    1047Mi           56%
[root@base ~]# kubectl top pods -n kube-system
NAME                                             CPU(cores)   MEMORY(bytes)
calico-kube-controllers-7ddc4f45bc-7458z        5m           27Mi
calico-node-krhrn                               20m          148Mi
calico-node-wwsh                                69m          139Mi
calico-node-xfxr                                 17m          142Mi
coredns-5dd5756b68-5rfd                        4m           27Mi
coredns-5dd5756b68-dnvb9                        3m           24Mi
etcd-master                                     84m          67Mi
kube-apiserver-master                           226m         382Mi
kube-controller-manager-master                  48m          78Mi
kube-proxy-4cjl                                  1m           16Mi
kube-proxy-bl742                                2m           16Mi
kube-proxy-fvh6l                                 2m           20Mi
kube-scheduler-master                           8m           39Mi
metrics-server-cddc466c6-dv72b                  9m           20Mi
[root@base ~]# kubectl top pods -n metallb-system
NAME                                             CPU(cores)   MEMORY(bytes)
controller-7499d4584d-f5v5x                     1m           24Mi
speaker-fgvfr                                     8m           20Mi
speaker-m4n6j                                     2m           16Mi
speaker-pcftg                                     2m           16Mi

```

Figura 4.4: Métricas de los nodos y pods del sistema.  
Fuente: elaboración propia.

#### 4.7.9. Mover página personalizada PHP a la carpeta compartida del NFS

Para mover la página personalizada `index.php` a la carpeta compartida de NFS, se utiliza el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 6-indextoNFS.yml
```

Este playbook realiza los siguientes pasos para copiar el archivo `index.php` al servidor NFS, que se encuentra en la ruta compartida de `/var/web`:

```
1 - name: Aplicar index.php a NFS
2 hosts: nas
3 become: true
4 tasks:
5 - name: Copiar
6 copy:
7 src: ~/ansible/nfs/web/index.php # Ruta en el nodo Ansible
8 dest: /var/web/index.php # Ruta en el servidor NFS
```

El archivo `index.php` se copia desde la ruta `/ansible/nfs/web/index.php` en el nodo Ansible hasta la carpeta compartida de NFS en `/var/web/index.php`. Esta operación asegura que la página personalizada esté disponible en todos los nodos que acceden a la carpeta compartida de NFS.

#### 4.7.10. Copiar archivos necesarios para el despliegue

Para copiar los archivos de configuración de Kubernetes como el Deployment, Service, PersistentVolume, PersistentVolumeClaim y Horizontal Pod Autoscaler (HPA) al nodo master, se utiliza el siguiente playbook:

```
1 ansible-playbook -i inventory.ini 8-dumpweb.yml
```

Este playbook realiza los siguientes pasos:

```
1 - name: Copiar todos los yaml de la web personalizada, deployment, service y
2 hpa en el master
3 hosts: master
4 become: true
5 tasks:
6 - name: Copiar deploy.yaml al master
7 copy:
8 src: ~/ansible/nfs/web/deploy.yaml # Ruta en el nodo Ansible
9 dest: /tmp/deploy.yaml # Ruta en el master
10
11 - name: Copiar pv.yaml al master
12 copy:
13 src: ~/ansible/nfs/web/pv.yaml # Ruta en el nodo Ansible
14 dest: /tmp/pv.yaml # Ruta en el master
15
16 - name: Copiar pvc.yaml al master
17 copy:
18 src: ~/ansible/nfs/web/pvc.yaml # Ruta en el nodo Ansible
19 dest: /tmp/pvc.yaml # Ruta en el master
20
21 - name: Copiar service.yaml al master
22 copy:
23 src: ~/ansible/nfs/web/service.yaml # Ruta en el nodo Ansible
24 dest: /tmp/service.yaml # Ruta en el master
25
26 - name: Copiar hpa.yaml al master
27 copy:
```

```

27 src: ~/ansible/nfs/web/hpa.yaml      # Ruta en el nodo Ansible
28 dest: /tmp/hpa.yaml                 # Ruta en el master

```

Una vez copiados los archivos necesarios, se aplica la configuración de Kubernetes utilizando los siguientes comandos:

```

1  ansible-playbook -i inventory.ini 9-webapply.yml

1  - name: Aplicar configuracion de Kubernetes
2    hosts: master
3    become: true
4    tasks:
5      - name: Aplicar PersistentVolume (pv.yaml)
6        shell: kubectl apply -f /tmp/pv.yaml
7
8      - name: Aplicar PersistentVolumeClaim (pvc.yaml)
9        shell: kubectl apply -f /tmp/pvc.yaml
10
11     - name: Aplicar Deployment (deploy.yaml)
12       shell: kubectl apply -f /tmp/deploy.yaml
13
14     - name: Aplicar Service (service.yaml)
15       shell: kubectl apply -f /tmp/service.yaml
16
17     - name: Aplicar Horizontal Pod Autoscaler (hpa.yaml)
18       shell: kubectl apply -f /tmp/hpa.yaml

```

Este playbook primero copia los archivos `deploy.yaml`, `pv.yaml`, `pvc.yaml`, `service.yaml`, y `hpa.yaml` desde el nodo Ansible al nodo master. Luego, aplica cada uno de estos archivos utilizando el comando `kubectl apply` para configurar los recursos necesarios en Kubernetes.

## 4.8 Descripción de los Recursos de la Web Desplegada

En esta sección, se describen los archivos de configuración utilizados para el despliegue de la aplicación web, incluyendo el uso de Persistent Volumes (PV), Persistent Volume Claims (PVC), Deployment, Service y Horizontal Pod Autoscaler (HPA).

### 4.8.1. PersistentVolume (PV)

El recurso `PersistentVolume` (PV) es un recurso de almacenamiento en Kubernetes. En este caso, utilizamos un volumen NFS para almacenar los archivos de la aplicación web.

```

1  apiVersion: v1
2  kind: PersistentVolume
3  metadata:
4    name: web-pv
5  spec:
6    storageClassName: storage-nfs
7    capacity:
8      storage: 1Gi
9    accessModes:
10     - ReadWriteMany
11    nfs:
12      path: /var/web
13      server: 192.168.1.9
14    readOnly: no

```

- **storageClassName:** Especifica la clase de almacenamiento, en este caso `storage-nfs`.
- **capacity:** Define la capacidad del volumen (1Gi).
- **accessModes:** Se configura como `ReadWriteMany`, lo que permite el acceso simultáneo desde varios pods.
- **nfs:** Configura el volumen para que use un servidor NFS en la dirección `192.168.1.9`.

#### 4.8.2. PersistentVolumeClaim (PVC)

El recurso `PersistentVolumeClaim` (PVC) se utiliza para solicitar almacenamiento de un `PersistentVolume` ya existente.

```
1  apiVersion: v1
2  kind: PersistentVolumeClaim
3  metadata:
4  name: web-pvc
5  namespace: metallb-system
6  spec:
7  storageClassName: storage-nfs
8  accessModes:
9  - ReadWriteMany
10 resources:
11 requests:
12 storage: 1Gi
13 volumeName: web-pv
```

- **storageClassName:** Especifica la clase de almacenamiento, igual que en el PV.
- **accessModes:** Configura el PVC para acceder al volumen con el modo `ReadWriteMany`.
- **resources.requests:** Solicita 1Gi de almacenamiento.
- **volumeName:** Hace referencia al `PersistentVolume` creado previamente (`web-pv`).

#### 4.8.3. Deployment

El recurso `Deployment` gestiona el ciclo de vida de los pods. En este caso, crea y gestiona una réplica de NGINX. Es importante destacar la configuración de los recursos solicitados y limitados para cada pod, lo que está directamente relacionado con el escalado automático.

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4  name: nginx-deployment
5  namespace: metallb-system
6  spec:
7  replicas: 3
8  selector:
9  matchLabels:
10 app: nginx
11 template:
12 metadata:
13 labels:
14 app: nginx
15 spec:
16 containers:
```



```
17 - name: nginx
18 image: nginx:alpine
19 ports:
20 - containerPort: 80
21 volumeMounts:
22 - name: web-volume
23   mountPath: /usr/share/nginx/html
24 resources:
25   requests:
26     cpu: 100m
27     memory: 128Mi
28   limits:
29     cpu: 200m
30     memory: 256Mi
31 volumes:
32 - name: web-volume
33   persistentVolumeClaim:
34     claimName: web-pvc
```

- **replicas:** Se define que habrá 3 réplicas de los pods.
- **resources.requests:** Cada pod solicita 100m de CPU y 128Mi de memoria.
- **resources.limits:** Los pods no pueden usar más de 200m de CPU y 256Mi de memoria.
- **volumeMounts:** El volumen NFS (web-volume) se monta en /usr/share/nginx/html para que el contenido web sea accesible desde el contenedor.

#### 4.8.4. Service

El recurso Service expone el Deployment de NGINX a través de un equilibrador de carga (LoadBalancer).

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: nginx-service
5    namespace: metallb-system
6  spec:
7    selector:
8      app: nginx
9    ports:
10     - name: http
11       protocol: TCP
12       port: 80
13       targetPort: 80
14     type: LoadBalancer
15     externalTrafficPolicy: Cluster
16     internalTrafficPolicy: Cluster
```

- **type: LoadBalancer:** Este tipo de servicio permite la exposición externa de la aplicación a través de un equilibrador de carga.
- **externalTrafficPolicy/internalTrafficPolicy:** Define la política para el tráfico externo/interno, que se maneja de forma global a nivel de clúster.

#### 4.8.5. HorizontalPodAutoscaler (HPA)

El recurso HorizontalPodAutoscaler ajusta automáticamente el número de réplicas del Deployment basado en la utilización de recursos, como la CPU. En este caso, el HPA escala el número de pods entre 3 y 6, dependiendo del uso de la CPU.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: nginx-hpa
  namespace: metallb-system
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-deployment
  minReplicas: 3
  maxReplicas: 6
  metrics:
  - type: Resource
    resource:
      name: cpu
    target:
      type: Utilization
      averageUtilization: 1
```

- **minReplicas y maxReplicas:** Establecen el rango de réplicas entre 3 y 6, permitiendo el escalado según la carga.
- **metrics:** Define que la métrica de escalado es el uso de CPU, y el escalado ocurrirá cuando el uso promedio de CPU supere el 1 %.

#### 4.8.6. Escalado Automático

El HorizontalPodAutoscaler es fundamental para el escalado automático de los pods cuando el servicio recibe una alta carga de tráfico. Si el número de réplicas de los pods es insuficiente para manejar la carga (definida por las métricas de CPU en el HPA), Kubernetes automáticamente escalará el número de réplicas, aumentando hasta el límite de 6 pods, para equilibrar la carga y garantizar un rendimiento estable.

Ahora ya conocemos el porcentaje de uso que tenemos de nuestro deployment:

```
1 kubectl get hpa -n metallb-system
```

```
[root@base ~]# kubectl get hpa -n metallb-system -o wide
NAME          REFERENCE          TARGETS  MINPODS  MAXPODS  REPLICAS  AGE
nginx-hpa     Deployment/nginx-de  0%/20%   3         6         3         115s
```

**Figura 4.5:** Información de hpa desplegado.  
Fuente: elaboración propia.

### 4.8.7. Instalar y utilizar Apache Bench para estresar el servidor

Para simular carga en el servidor y observar cómo responde el Horizontal Pod Autoscaler (HPA) en Kubernetes, seguimos los siguientes pasos:

**Instalar Apache Bench en el nodo Ansible:** Primero, debemos instalar Apache Bench, que forma parte del paquete `httpd-tools`, en el nodo Ansible. Esto se realiza con el siguiente comando:

```
1 dnf install httpd-tools -y
```

**Verificar la instalación:** Una vez instalado Apache Bench, podemos verificar que la instalación fue exitosa ejecutando el siguiente comando:

```
1 ab -V
```

Esto nos dará la versión de `ab` instalada, confirmando que la herramienta está lista para usar.

**Acceder al servidor web a través del Load Balancer:** Luego, accederemos al servidor web a través de la IP del Load Balancer (en este caso, `192.168.1.10`) en el puerto 80, utilizando la URL `/index.php` desde un navegador web de tu host.

**Ejecutar Apache Bench para estresar el recurso:** Ahora, utilizamos Apache Bench para realizar una prueba de carga. Ejecutamos el siguiente comando para simular un número de solicitudes concurrentes y ver cómo responde el HPA:

```
1 ab -n 1000 -c 10 -r http://192.168.1.10/index.php
```

- `-n 1000`: Número total de solicitudes que deseas enviar. En este caso, 1000 solicitudes.
- `-c 10`: Número de solicitudes concurrentes. En este caso, se simulan 10 solicitudes al mismo tiempo.
- `-r`: Esta opción asegura que Apache Bench no se detenga ante errores y continúe enviando solicitudes hasta completar el total especificado.

**Observar el escalado del HPA:** Mientras Apache Bench está ejecutando la prueba, podemos escribir en el master y monitorear cómo cambia el número de réplicas de los pods, lo cual es gestionado por el HPA. Utilizamos el siguiente comando para obtener el estado actual del HPA:

```
1 kubectl get hpa -n metallb-system
```

Este comando nos muestra el número de réplicas que el HPA ha ajustado en función de la carga de trabajo recibida. El HPA debería aumentar las réplicas de los pods conforme aumenta la carga (por ejemplo, al recibir más solicitudes).

Al realizar la prueba de carga con Apache Bench, observamos cómo el HPA escala los recursos de acuerdo a la configuración que definimos previamente, lo que garantiza que los recursos estén disponibles para manejar el tráfico de forma eficiente.

```
[root@base ~]# kubectl top pods -n metallb-system
NAME                                CPU(cores)    MEMORY(bytes)
controller-7499d4584d-f5v5x        2m            25Mi
nginx-deployment-579b759bcb-55fpc  6m            2Mi
nginx-deployment-579b759bcb-5dk6f  6m            2Mi
nginx-deployment-579b759bcb-mjfbq  6m            2Mi
speaker-fgvfr                       8m            21Mi
speaker-min6j                       2m            18Mi
speaker-pcftg                       1m            18Mi

[root@base ~]# kubectl get hpa -n metallb-system -o wide
NAME      REFERENCE            TARGETS    MINPODS    MAXPODS    REPLICAS
nginx-hpa Deployment/nginx-deployment  22/1%     3          6          6

[root@base ~]# kubectl top pods -n metallb-system
NAME                                CPU(cores)    MEMORY(bytes)
controller-7499d4584d-f5v5x        1m            25Mi
nginx-deployment-579b759bcb-55fpc  0m            2Mi
nginx-deployment-579b759bcb-5dk6f  0m            2Mi
nginx-deployment-579b759bcb-mjfbq  0m            2Mi
nginx-deployment-579b759bcb-rv76g  0m            2Mi
nginx-deployment-579b759bcb-thvlu  0m            2Mi
nginx-deployment-579b759bcb-zznvr  0m            2Mi
speaker-fgvfr                       7m            21Mi
speaker-min6j                       4m            18Mi
speaker-pcftg                       4m            18Mi
[root@base ~]#
```

Figura 4.6

Fuente: elaboración propia.