Proyecto Monitorización Kubernetes

Salvador Lobato Gálvez

Índice

Kubernetes

Kubernetes es una plataforma portable y extensible de código abierto para administrar cargas de trabajo y servicios. Kubernetes facilita la automatización y la configuración declarativa.

Permite despliegues automáticos, escalabilidad y gestión de contenedores de aplicaciones.

Estructura nodos virtuales

Creación de nodos en el Cloud a partir de un proyecto llamado **OpenStack** que nos proporciona una infraestructura. Esta infraestructura nos proporciona una red con acceso a Internet, en este caso nos proporciona una ip con acceso a internet llamada IP Flotante y un ip local que nos proporciona conexión entre las máquinas.

Para empezar creamos 3 instancias, la principal es el nodo_master que nos proporciona la administración del cluster de kubernetes y dos nodos que añadiremos posteriormente al cluster.

Instancias

	Nombre de la instancia	Nombre de la imagen	Dirección IP
☐	Nodo_2	Debian Stretch 9.11	• 10.0.0.7 IPs flotantes: • 172.22.200.222
☐	Nodo_1	Debian Stretch 9.11	• 10.0.0.3 IPs flotantes: • 172.22.200.221
☐	Nodo_master	Debian Stretch 9.11	• 10.0.0.10 IPs flotantes: • 172.22.200.212

Comenzaremos entrados a dichas máquinas a través de ssh y una vez dentro actualizamos:

```
salva@debian:~/.ssh$ ssh -i clave-ecdsa.key debian@172.22.201.31
Linux nodo-master 4.9.0-11-amd64 #1 SMP Debian 4.9.189-3+deb9u1 (2019-09-20) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Nov 22 19:21:30 2019 from 172.23.0.94
debian@nodo-master:~$ sudo apt update && sudo apt upgrade -y
```

Docker

Crea contenedores ligeros y portables para las aplicaciones software, independientemente del sistema operativo que la máquina tenga por debajo, facilitando también los despliegues.

Contenedor: es un conjunto de unos o más procesos que se encuentran aislados del resto del sistema, son móviles y homogéneos a medida que pasan de la etapa de desarrollo a la de prueba y a la de producción.

Kubernetes es un orquestador de contenedores por eso es necesario instalar Docker en los nodos para la creación de contenedores.

Orquestador: Maneja las interconexiones e interacciones entre cargas de trabajo en nubes privadas y públicas. Conecta las tareas automatizadas en flujo de trabajo cohesivo para cumplir metas, con vigilancia de permisos y aplicación de políticas.

Instalamos Docker en los 3 nodos:

Instalamos los paquetes para permitir el apt uso de un repositorio sobre https.

```
debian@nodo-master:~$ sudo apt-get install \
>     apt-transport-https \
>     ca-certificates \
>     curl \
>     gnupg2 \
>     software-properties-common
```

Agregar clave GPG oficial de Docker:

```
debian@nodo-master:~$ curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
OK
```

Añadimos el repositorio para nuestra versión de debian:

```
debian@nodo-master:~$ sudo add-apt-repository \
>    "deb [arch=amd64] https://download.docker.com/linux/debian \
>    $(lsb_release -cs) \
>    stable"
```

Pasamos a la instalación:

```
debian@nodo-master:~$ sudo apt update
```

Instalamos la versión de docker que actualmente es compatible con kubernetes:

```
debian@nodo-master:~$ sudo apt install docker-ce=18.06.3~ce~3-0~debian
```

Comprobamos la versión instalada:

```
debian@nodo-master:~$ docker --version
Docker version 18.06.3-ce, build d7080c1
```

Añadimos como administrador cgroups a systemd para que el tiempo de ejecución y el uso de kubelet estabilicen el sistema.

```
root@nodo-master:/home/debian# cat > /etc/docker/daemon.json <<EOF
{
  "exec-opts": ["native.cgroupdriver=systemd"],
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "100m"
  },
  "storage-driver": "overlay2"
}
EOF
root@nodo-master:/home/debian# mkdir -p /etc/systemd/system/docker.service.d
root@nodo-master:/home/debian# systemctl daemon-reload
root@nodo-master:/home/debian# systemctl restart docker
```

Instalación de kubeadm, kubelet y kubectl

Kubeadm: es una herramienta que nos permite el despliegue de un cluster de kubernetes de manera sencilla.

Cluster: se aplica a los conjuntos o conglomerados de ordenadores unidos entre sí normalmente por una red de alta velocidad y que se comportan como si fuesen una única computadora.

Kubelet: es el componente que se ejecuta en todas las máquinas de su clúster y es responsable de ejecutar los pods y los contenedores.

Kubectl: complemento para la línea de comandos, que nos permite controlar el cluster.

Comenzamos la instalación en los 3 nodos:

```
debian@nodo-master:~$ sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -

debian@nodo-master:~$ sudo su

root@nodo-master:/home/debian# cat <<EOF >/etc/apt/sources.list.d/kubernetes.list

deb https://apt.kubernetes.io/ kubernetes-xenial main
EOF


debian@nodo-master:~$ sudo apt update

debian@nodo-master:~$ sudo apt-get install -y kubelet kubeadm kubectl
```

Inicializando el nodo master

En el nodo master creamos el cluster y añadimos el pod-network-cidr que será el CIDR de la red por donde se comunican los nodos del cluster.

También añadiré la ip flotante para tener acceso al cluster desde el exterior. --apiserver-cert-extra-sans es necesario para validar el certificado de esta ip.

```
debian@nodo-master:~$ sudo kubeadm init --apiserver-cert-extra-sans=172.22.201.31 --pod-network-cidr=192.168.0.0/24
```

Para que kubectl funcione para un usuario sin privilegios y pueda manejar el cluster ejecutamos los siguientes comandos, siguiendo las instrucciones de la documentación oficial:

```
debian@nodo-master:~$ mkdir -p $HOME/ .kube

debian@nodo-master:~$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

debian@nodo-master:~$ sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Instalación del pod para gestionar la red

Es necesario instalar un complemento de red de pod para que los pods puedan comunicarse entre si.

Kubeadm solo admite redes basadas en la interfaz de red de contenedores(CNI).

En este caso, instalamos Calico:

```
kubectl apply -f https://docs.projectcalico.org/v3.8/manifests/calico.yaml
```

A continuación comprobamos los pods y servicios:

```
debian@nodo-master:~$ kubectl get pods --all-namespaces
NAMESPACE     NAME                                 READY   STATUS     RESTARTS   AGE
kube-system   calico-kube-controllers-55754f75c-lkkhk   0/1     Pending    0          7s
kube-system   calico-node-2hbrh                    0/1     Init:0/3   0          8s
kube-system   coredns-5644d7b6d9-9cdzb             0/1     Pending    0          4m45s
kube-system   coredns-5644d7b6d9-9jb8k             0/1     Pending    0          4m45s
kube-system   etcd-nodo-master                     1/1     Running    0          3m52s
kube-system   kube-apiserver-nodo-master           1/1     Running    0          3m56s
kube-system   kube-controller-manager-nodo-master  1/1     Running    0          4m10s
kube-system   kube-proxy-s4xdl                     1/1     Running    0          4m45s
kube-system   kube-scheduler-nodo-master           1/1     Running    0          3m50s
debian@nodo-master:~$ kubectl get services
NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP   5m36s
```

Uniendo los nodos al cluster

Para que los nodos puedan formar parte del cluster, es necesario tener el token creado al ejecutar el kubeadm init.

Para poder unir los nodos tenemos que tener abierto el 6443 que se abrirá desde la interface de OpenStack:

☐	Entrante	IPv4	TCP	6443	0.0.0.0/0
☐	Saliente	IPv4	TCP	6443	0.0.0.0/0
☐	Saliente	IPv4	UDP	6443	0.0.0.0/0
☐	Entrante	IPv4	UDP	6443	0.0.0.0/0

Un comando para mostrar dicho token:

```
debian@nodo-master:~$ kubeadm token list
TOKEN                     TTL       EXPIRES                     USAGES                   DESCRIPTION                                                EXTRA GROUPS
rxwnro.x6mo04wvx7fbb5rl   2h        2019-09-24T18:51:21Z        authentication,signing   The default bootstrap token generated by 'kubeadm init'.   system:bootstrappers:kubeadm:default-node-token
```

También muestra la fecha de caducidad de dicho token, si este caducara, ejecutamos el comando siguiente para generar un nuevo token:

```
kubeadm token create
```

En el nodo2 entramos como superusuario y ejecutamos el siguiente comando para unir el nodo, indicando el token y la ip.

```
debian@nodo-1:~$ sudo su
```

Paramos la swap temporalmente:

```
root@nodo-1:/home/debian# swapoff -a
```

También es nesario saber discovery-token-ca-cert-hash, nos aparece al finalinar el kubeadm init, otra forma de obtenerlo es realizando este comando en el nodo_master:

```
debian@nodo-master:~$ openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | \
   openssl dgst -sha256 -hex | sed 's/^.* //'
```

A continuación el nodo1 ejecutamos el siguiente comando para unir el nodo1. Añadiremos la ip, el puerto, el token y el discovery-token-ca-cert-hash.

```
debian@nodo-1:~$ sudo kubeadm join 172.22.201.31:6443 --token 6o79k7.ntcbqfbgc43kylbt
--discovery-token-ca-cert-hash
sha256:ded7a86bb543b26cfdd2f3f6b8cbd16f692d66be23fef7178c93444078a14982
```

Desde el master comprobamos que el nodo2 se ha unido correctamente:

```
debian@nodo-master:~$ kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
nodo-1        Ready    <none>   39m   v1.16.3
nodo-master   Ready    master   23h   v1.16.3
```

A continuación añadimos el nodo2.

Para que la swap no se ejecute al iniciar la máquina, en el fichero /etc/fstab comentamos la línea de la swap:

```
# swap was on /dev/sda2 during installation
#UUID=be1becd9-b32e-4d15-8df1-3d63b72642b0 none            swap    sw              0       0
```

Para añadir el nodo3 realizamos la misma operación:

```
debian@nodo-2:~$ sudo kubeadm join 172.22.201.31:6443 --token 6o79k7.ntcbqfbgc43kylbt
--discovery-token-ca-cert-hash
sha256:ded7a86bb543b26cfdd2f3f6b8cbd16f692d66be23fef7178c93444078a14982
```

Comprobamos:

```
debian@nodo-master:~$ kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
nodo-1        Ready    <none>   44m   v1.16.3
nodo-2        Ready    <none>   94s   v1.16.3
nodo-master   Ready    master   23h   v1.16.3
```

Acceso desde un cliente externo

Accedemos desde la máquina anfitriona, necesitamos tener instalado kubectl.

Instalación:

```
sudo apt update && sudo apt upgrade -y
sudo apt install apt-transport-https
 sudo curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
```

```
OK
root@debian:# cat <<EOF >/etc/apt/sources.list.d/kubernetes.list
> deb http://apt.kubernetes.io/ kubernetes-xenial main
> EOF
sudo apt update
sudo apt install kubectl
```

Configuramos el acceso al cluster:

Desde el nodo master damos permiso de lectura.

```
debian@nodo-master:~$ sudo  chmod 644 /etc/kubernetes/admin.conf
```

A continuación desde el cliente:

Accedemos al master a través de sftp utilizando la clave privada de vagrant:

```
salva@debian:~$ export IP_MASTER=172.22.201.31
salva@debian:~/.ssh$ sftp -i clave-ecdsa.key debian@172.22.201.31
Connected to 172.22.201.31.
sftp> get /etc/kubernetes/admin.conf
Fetching /etc/kubernetes/admin.conf to admin.conf
/etc/kubernetes/admin.conf                                    100% 5449    18.3KB/s   00:00
sftp> exit
```

Después de descargar el fichero, lo movemos a su directorio correspondiente y cambios los parámetros de fichero para acceder desde el cliente:

```
salva@debian:~/.ssh$ mv admin.conf ~/.kube/mycluster.conf
salva@debian:~/.ssh$ sed -i -e "s#server: https://*:6443#server: https://${IP_MASTER}:6443#g" ~/.kube/mycluster.conf
salva@debian:~/Documentos/Proyecto/Nodos$ export KUBECONFIG=~/.kube/mycluster.conf
```

Comprobamos que tenemos acceso:

```
salva@debian:~/.ssh$ kubectl cluster-info
Kubernetes master is running at https://172.22.201.31:6443
salva@debian:~$ kubectl get nodes
NAME         STATUS   ROLES    AGE    VERSION
nodo-1       Ready    <none>   50m    v1.16.3
```

```
nodo-2        Ready    <none>   8m8s   v1.16.3
nodo-master   Ready    master   23h    v1.16.3
```

Desplegando WordPress con Mysql y almacenamiento persistente

Configuración del servidor NFS

Vamos a crear un recurso compartido para compartirlo entre los nodos del cluster.

El PersistentVolumen es un objeto que representa los volúmenes disponibles del cluster.

En nodo_master instalamos el servidor NFS:

```
debian@nodo-master:~$ sudo su
root@nodo-master:/home/debian# apt install nfs-kernel-server
root@nodo-master:/home/debian# mkdir -p /var/shared
root@nodo-master:/var/shared# mkdir vol1
root@nodo-master:/var/shared# mkdir vol2
```

En el fichero /etc/exports declaramos los directorios que vamos a exportar:

```
root@nodo-master:~# nano /etc/exports
/var/shared/vol1 10.0.0.0/24(rw,sync,no_root_squash,no_all_squash)
/var/shared/vol2 10.0.0.0/24(rw,sync,no_root_squash,no_all_squash)
```

Hemos añadido la red interna de las instancias del cloud “10.0.0.0/24”.

Reiniciamos el servicio:

```
root@nodo-master:~# systemctl restart nfs-kernel-server.service
```

Comprobamos los directorios exportados:

```
root@nodo-master:~# showmount -e 127.0.0.1
Export list for 127.0.0.1:
/var/shared/vol2 10.0.0.0/24
/var/shared/vol1 10.0.0.0/24
```

En los nodos restantes vamos a montar el directorio compartido:

```
debian@nodo-1:~$ sudo su
root@nodo-1:/home/debian# apt install nfs-common
```

Comprobamos los directorios exportados del nodo_master:

```
root@nodo-1:/home/debian#
Export list for 10.0.0.11:
/var/shared/vol2 10.0.0.0/24
/var/shared/vol1 10.0.0.0/24
```

Creamos los directorios donde montaremos el recurso compartido:

```
root@nodo-1:/home/debian# mkdir /var/data
root@nodo-1:/home/debian# mkdir /var/data/vol1
root@nodo-1:/home/debian# mkdir /var/data/vol2
```

Y lo montamos:

```
root@nodo-1:/home/debian# mount -t nfs4 10.0.0.11:/var/shared/vol1 /var/data/vol1
root@nodo-1:/home/debian# mount -t nfs4 10.0.0.11:/var/shared/vol2 /var/data/vol2
```

Realizamos en el nodo2:

```
debian@nodo-2:~$ sudo su
root@nodo-2:/home/debian# apt install nfs-common
```

Comprobamos los directorios exportados:

```
root@nodo-2:/home/debian# showmount -e 10.0.0.11
Export list for 10.0.0.11:
/var/shared/vol2 10.0.0.0/24
/var/shared/vol1 10.0.0.0/24
```

Creamos los directorios donde montaremos el recurso compartido:

```
root@nodo-2:/home/debian# mkdir /var/data
root@nodo-2:/home/debian# mkdir /var/data/vol1
root@nodo-2:/home/debian# mkdir /var/data/vol2
```

Y lo montamos:

```
root@nodo-2:/home/debian# mount -t nfs4 10.0.0.11:/var/shared/vol1 /var/data/vol1
root@nodo-2:/home/debian# mount -t nfs4 10.0.0.11:/var/shared/vol2 /var/data/vol2
```

Almacenamiento PersistentVolumen

Es un objeto que representa los volúmenes disponibles en el cluster.

A continuación desplegamos el siguiente fichero “wordpress-pv.yaml”:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: volumen1
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /var/shared/vol1
    server: 10.0.0.11
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: volumen2
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  nfs:
    path: /var/shared/vol2
    server: 10.0.0.11
```

Creación:

```
debian@nodo-master:~$ kubectl create -f wordpress-pv.yaml

persistentvolume/volumen1 created

persistentvolume/volumen2 created
```

Comprobamos:

```
debian@nodo-master:~$ kubectl get pv
NAME      CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   REASON   AGE
volumen1   5Gi        RWX            Recycle          Available                                   69s
volumen2   5Gi        RWX            Recycle          Available                                   69s
```

Nos muestra el nombre del volumen, la capacidad en este caso 5GB, el modo de acceso es RMX y su política de reciclaje es de reutilización de contenido.

RMX: Modo de acceso de lectura y escritura para todos los nodos.

Creación de Namespace

Los namespaces son una forma de dividir los recursos del clúster entre múltiples usuarios.

A cada namespace se le puede asignar una cuota, definirle reglas y políticas de acceso.

Creamos un namespace desplegando el siguiente wordpress-ns.yaml:

```
apiVersion: v1

kind: Namespace

metadata:
  name: wordpress
```

Lo creamos:

```
debian@nodo-master:~$ kubectl create -f wordpress-ns.yaml

namespace/wordpress created
```

Solicitud de almacenamiento: PersistentVolumenClaims

PersistentVolumenClaims es una solicitud de almacenamiento por parte de un usuario, es decir, permite que un usuario consuma recursos de almacenamiento abstractos.

Creamos una solicitud de almacenamiento para la base de datos y otra para la aplicación.

Desplegamos el fichero mariadb-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
  namespace: wordpress
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Y el fichero wordpress-pvc.yaml:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: wp-pv-claim
  namespace: wordpress
  labels:
    app: wordpress
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi
```

Creamos las solicitudes:

```
debian@nodo-master:~$ kubectl create -f mariadb-pvc.yaml

persistentvolumeclaim/mariadb-pvc created

debian@nodo-master:~$ kubectl create -f wordpress-pvc.yaml

persistentvolumeclaim/wordpress-pvc created
```

Comprobamos:

```
debian@nodo-master:~$ kubectl get pv,pvc -n wordpress
NAME                  CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                 STORAGECLASS
REASON   AGE
persistentvolume/volumen1   5Gi        RWX            Recycle          Bound    wordpress/mysql-pv-claim                  22h
persistentvolume/volumen2   5Gi        RWX            Recycle          Bound    wordpress/wp-pv-claim                     22h

NAME                                STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/mysql-pv-claim   Bound    volumen1   5Gi        RWX                           22h
persistentvolumeclaim/wp-pv-claim      Bound    volumen2   5Gi        RWX                           22h
```

El status no muestra que el volumen está bound al su volumen correspondiente.

Services

Los servicios nos permiten acceder a nuestra aplicaciones.

Ofrecen una dirección virtual y un nombre que identifica al conjunto de pods.

La conexión al servicio se puede realizar desde otros pods o desde el exterior.

Desplegamos el siguiente fichero mariadb-svr.yaml:

Este servicio es necesario para que acceda la aplicación a la base de datos.

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress-mysql
  namespace: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - port: 3306
  selector:
    app: wordpress
    tier: mysql
  clusterIP: None
```

Port: Indicamos el puerto de acceso.

Type: ClusterIP : Este tipo solo permite el acceso interno entre distintos servicios.

Desplegamos el siguiente servicio para la aplicación:

```
apiVersion: v1
kind: Service
metadata:
  name: wordpress
  namespace: wordpress
  labels:
    app: wordpress
spec:
  ports:
    - name: http-sv-port
      port: 80
      targetPort: http-port
    - name: https-sv-port
      port: 443
      targetPort: https-port
  selector:
    app: wordpress
    tier: frontend
  type:  LoadBalancer
```

En este caso el type es LoadBalancer: Actuará como balanceador de carga y nos permitirá accesos desde el exterior.

El puerto se genera entre el rango 30000-40000. Para acceder usamos la ip del nodo_master y el puerto asignado. Por ejemplo 172.22.201.31:30300

En openstack habilitamos los puertos necesarios:

Entrante	IPv4	TCP	30000 - 40000	0.0.0.0/0	-
Saliente	IPv4	TCP	30000 - 40000	0.0.0.0/0	-

Creamos los servicios:

```
debian@nodo-master:~$ kubectl create -f mariadb-srv.yaml
service/mariadb-service created
debian@nodo-master:~$ kubectl create -f wordpress-srv.yaml
service/wordpress-service created
```

Ingress Controller

Ingress nos permite utilizar un proxy inverso que nos permite el acceso a nuestras aplicaciones por medio de nombres.

Desplegamos un pod que implementa un proxy inverso llamado Traefik.

Desplegamos el fichero wordpress-ingress.yaml:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: wordpress-ingress
  namespace: wordpress
  labels:
    app: wordpress
    type: frontend
  annotations:
    kubernetes.io/ingress.class: traefik
spec:
  rules:
  - host: wp.172.22.200.221.nip.io
    http:
      paths:
      - path: /
        backend:
          serviceName: wordpress
          servicePort: http-sv-port
```

En el host añadimos la ip terminada en .nip.io esto permite asignar cualquier IP a un nombre de host. La ip será la del nodo que despliega la aplicación.

Indicamos el puerto desde donde accedemos, en esta caso http-sv-port que el puerto 80 desplegado en el servicio de la aplicación.

Lo creamos:

```
debian@nodo-master:~$ kubectl create -f wordpress-ingress.yaml

ingress.extensions/wordpress-ingress created
```

Secrets

Los Secrets nos permiten guardar información sensible que será codifica.

En este caso añadimos las claves que usaremos para las variables de entorno.

Desplegamos el fichero mariadb-secret.yaml:

```
apiVersion: v1
data:
  dbname: d29yZHByZXNz
  dbpassword: cGFzc3dvcmQxMjM0
  dbrootpassword: cm9vdDEyMzQ=
  dbuser: dXNlcl93b3JkcHJlc3M=
kind: Secret
metadata:
  creationTimestamp: null
  name: mariadb-secret
  namespace: wordpress
```

Lo creamos:

```
debian@nodo-master:~$ kubectl create -f mariadb-secret.yaml

secret/mariadb-secret created
```

Despliegue de Mysql y wordpress

Desplegamos la base de datos con el fichero mariadb-deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress-mysql
  namespace: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: mysql
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: mysql
    spec:
      containers:
      - image: mysql:5.6
        name: mysql
        env:
        - name: MYSQL_ROOT_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mariadb-secret
              key: dbrootpassword
        ports:
        - containerPort: 3306
          name: mysql
        volumeMounts:
        - name: mariadb-vol
          mountPath: /var/lib/mysql
      volumes:
      - name: mariadb-vol
        persistentVolumeClaim:
          claimName: mysql-pv-claim
```

En el fichero podemos comprobar el nombre del deployment, la imagen de mysql para los containers, el puerto, las variables de entorno llamando al secret y los volúmenes.

Lo creamos:

```
debian@nodo-master:~$ kubectl create -f mariadb-deployment.yaml

deployment.apps/mariadb-deployment created
```

Desplegamos la aplicación con el fichero wordpress-deplyment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: wordpress
```

```
  namespace: wordpress
  labels:
    app: wordpress
spec:
  selector:
    matchLabels:
      app: wordpress
      tier: frontend
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: wordpress
        tier: frontend
    spec:
      containers:
      - image: wordpress:4.8-apache
        name: wordpress
        env:
        - name: WORDPRESS_DB_HOST
          value: wordpress-mysql
        - name: WORDPRESS_DB_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mariadb-secret
              key: dbrootpassword
        ports:
        - containerPort: 80
          name: http-port
        volumeMounts:
        - name: wordpress-vol
          mountPath: /var/www/html
      volumes:
      - name: wordpress-vol
        persistentVolumeClaim:
          claimName: wp-pv-claim
```

En este caso el fichero nos muestra el label de tipo frontend, puertos, variables de entorno llamando al secret y los volúmenes.

Lo creamos:

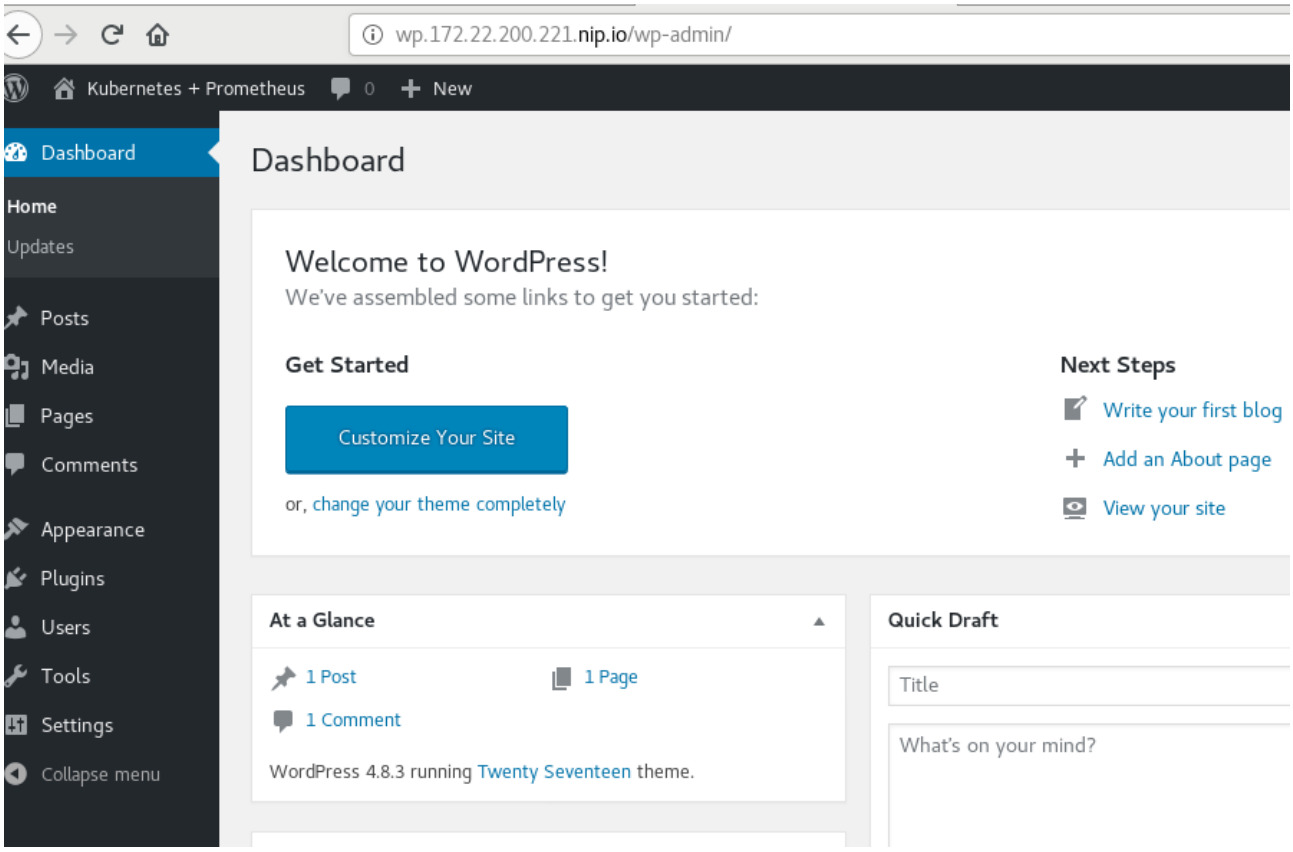
```
debian@nodo-master:~$ kubectl create -f wordpress-deployment.yaml

deployment.apps/wordpress-deployment created
```

Comprobamos:

```
debian@nodo-master:~$ kubectl get pvc,services,deploy,ingress -n wordpress
NAME                                  STATUS   VOLUME    CAPACITY   ACCESS MODES   STORAGECLASS   AGE
persistentvolumeclaim/mysql-pv-claim   Bound    volumen1   5Gi        RWX                           22h
persistentvolumeclaim/wp-pv-claim      Bound    volumen2   5Gi        RWX                           22h


NAME                 TYPE          CLUSTER-IP       EXTERNAL-IP   PORT(S)                  AGE
```

```
service/wordpress         LoadBalancer   10.106.254.212   <pending>     80:31818/TCP,443:30404/TCP   21h
service/wordpress-mysql   ClusterIP      None             <none>        3306/TCP                     22h

NAME                              READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/wordpress         1/1     1            1           21h
deployment.apps/wordpress-mysql   1/1     1            1           22h

NAME                                HOSTS                    ADDRESS   PORTS   AGE
ingress.extensions/wordpress-ingress   wp.172.22.200.221.nip.io             80      21h
```

Accedemos al navegador y comprobamos:



Prometheus

Prometheus es una solución de código abierto para monitorizar las métricas y administrar las alertas del sistema.

Para monitorizar un clúster en kubernetes usaremos operator Prometheus.

Operator permite extender el comportamiento del clúster sin modificar el código de kubernetes.

Los operadores son clientes de la API de Kubernetes que actúan como controladores para un recurso personalizado, en este caso Prometheus.

La instalación de operator Prometheus se realizaŕa con Helm.

Helm simplifica el empaquetado y la implementación de aplicaciones en kubernetes.

Operator Prometheus tiene un sistema de Alertmanager y Grafana.

Grafana es una herramienta para visualizar datos en serie temporales. A partir de una serie de datos recolectados obtendremos un panorama gráfico de la situación del Clúster.

Alertmanager maneja alertas enviadas por Prometheus. Se encarga de deduplicar, agrupar y enrutar a las integraciones correctas del receptor. Por ejemplo correo electrónicoy Slack.

Instalación de Helm

Instalamos Helm en el nodo_master.

Primero descargamos un script de instalación:

```
debian@nodo-master:~$ curl https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3 > get_helm.sh
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  6617  100  6617    0     0   1227      0  0:00:05  0:00:05 --:--:--  1709
```

Añadimos permisos al script:

```
debian@nodo-master:~$ sudo chmod 700 get_helm.sh
```

Ejecutamos el script:

```
debian@nodo-master:~$ ./get_helm.sh
Downloading https://get.helm.sh/helm-v3.0.1-linux-amd64.tar.gz
Preparing to install helm into /usr/local/bin
helm installed into /usr/local/bin/helm
```

Instalación de operator Prometheus

Instalación de operator Prothemetus en un espacio de nombre diferente:

Creamos un nuevo namespace:

```
debian@nodo-master:~$ nano monitor-ns.yaml
apiVersion: v1
kind: Namespace
metadata:
  name: monitor
kubectl create -f monitor-ns.yaml
namespace/monitor created
```

Instalamos operator Prometheus:

```
debian@nodo-master:~$ helm install prometheus-operator stable/prometheus-operator
--namespace monitor
manifest_sorter.go:175: info: skipping unknown hook: "crd-install"
manifest_sorter.go:175: info: skipping unknown hook: "crd-install"
manifest_sorter.go:175: info: skipping unknown hook: "crd-install"
manifest_sorter.go:175: info: skipping unknown hook: "crd-install"
manifest_sorter.go:175: info: skipping unknown hook: "crd-install"
NAME: prometheus-operator
LAST DEPLOYED: Wed Dec 11 20:02:18 2019
NAMESPACE: monitor
STATUS: deployed
REVISION: 1
NOTES:
The Prometheus Operator has been installed. Check its status by running:
  kubectl --namespace monitor get pods -l "release=prometheus-operator"
```

```
Visit https://github.com/coreos/prometheus-operator for instructions on how
to create & configure Alertmanager and Prometheus instances using the Operator.
```

Comprobamos los pods:

```
debian@nodo-master:~$ kubectl get pods -n monitor
NAME                                                     READY   STATUS    RESTARTS   AGE
alertmanager-prometheus-operator-alertmanager-0          2/2     Running   0          2m47s
prometheus-operator-grafana-7df446f77c-r5csh             2/2     Running   0          2m56s
prometheus-operator-kube-state-metrics-6f84ffd77-6s829   1/1     Running   0          2m56s
prometheus-operator-operator-f5c594c48-ndsdj             2/2     Running   0          2m56s
prometheus-operator-prometheus-node-exporter-79kxp       1/1     Running   0          2m56s
prometheus-operator-prometheus-node-exporter-pkf6x       1/1     Running   0          2m56s
prometheus-operator-prometheus-node-exporter-rg8lj       1/1     Running   0          2m56s
prometheus-prometheus-operator-prometheus-0              3/3     Running   1          2m37s
```

Comprobamos los servicios:

```
debian@nodo-master:~$ kubectl get svc -n monitor
NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                 ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   22h
prometheus-operated                   ClusterIP   None             <none>        9090/TCP                     22h
prometheus-operator-alertmanager      ClusterIP   10.96.74.153     <none>        9093/TCP                     22h
prometheus-operator-grafana           ClusterIP   10.110.147.108   <none>        80/TCP                       22h
prometheus-operator-kube-state-metrics   ClusterIP   10.99.22.245     <none>        8080/TCP                     22h
prometheus-operator-operator          ClusterIP   10.96.111.57     <none>        8080/TCP,443/TCP             22h
prometheus-operator-prometheus        ClusterIP   10.101.37.188    <none>        9090/TCP                     22h
prometheus-operator-prometheus-node-exporter   ClusterIP   10.108.8.66      <none>        9100/TCP                     22h
```

Para acceder desde el exterior necesitamos modificar los sevicios de ClusterIP a NodePort.

Editamos los siguientes servicios, estos servicios nos proporcionan acceso desde el navegador a Prometheus, Grafana y Alerts.

Simplemente hay que cambiar en la línea type: ClusterIP . Modificarlo a type: NodePort

```
debian@nodo-master:~$ kubectl edit service/prometheus-operator-prometheus -n monitor
service/prometheus-operator-prometheus edited
```

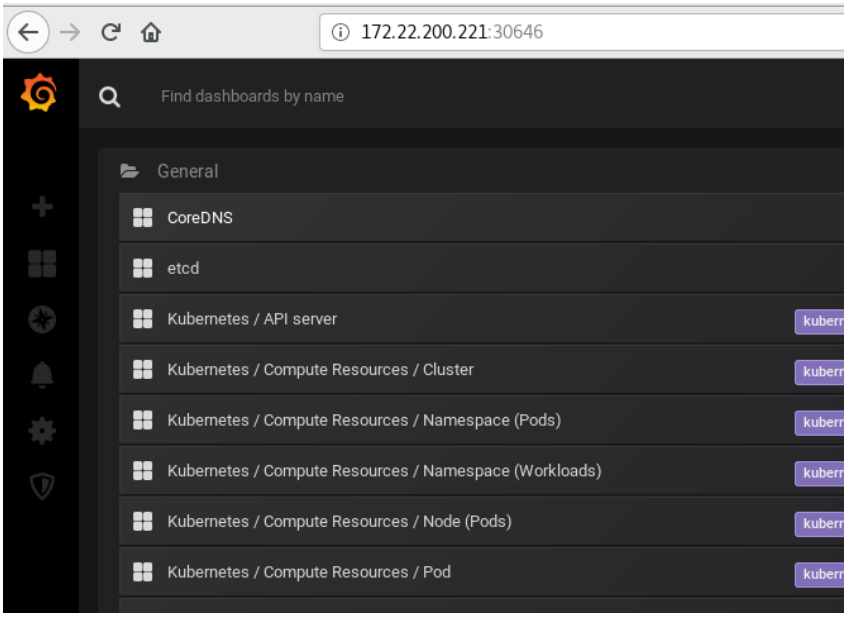
```
debian@nodo-master:~$ kubectl edit service/prometheus-operator-grafana -n monitor
service/prometheus-operator-grafana edited
debian@nodo-master:~$ kubectl edit service/prometheus-operator-alertmanager  -n monitor
service/prometheus-operator-alertmanager edited
```

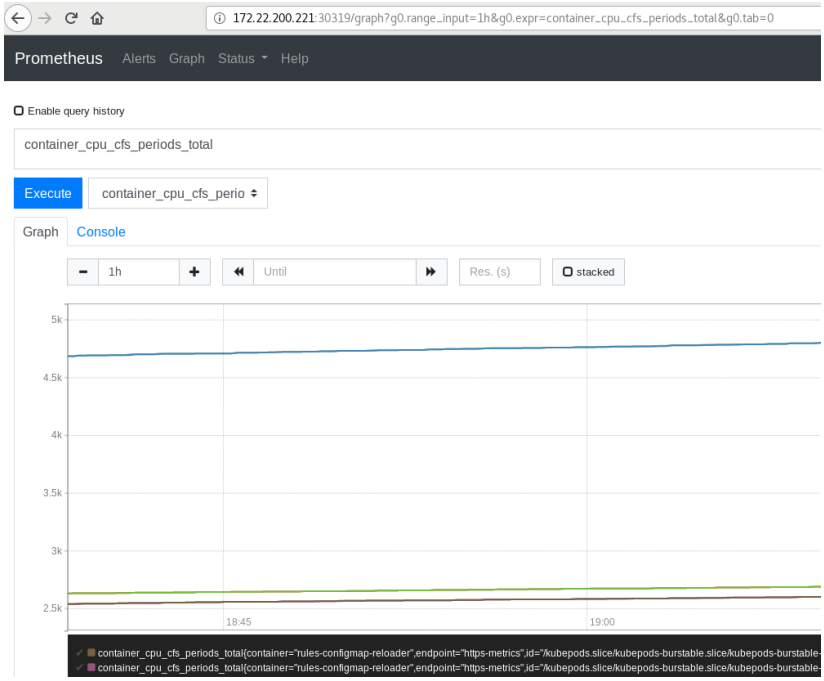
Comprobamos los servicios en este caso nos aparecen el puerto para acceder desde el exterior:

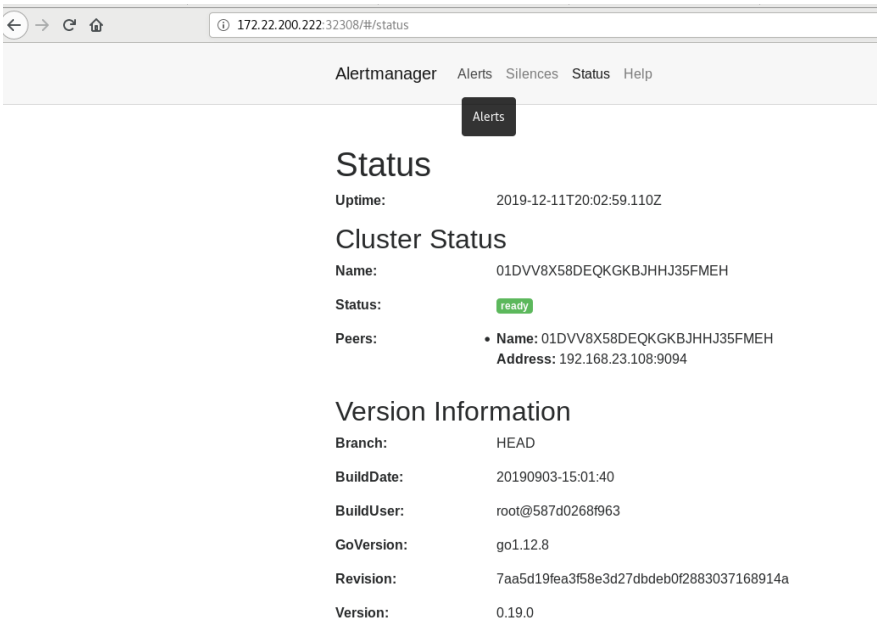
```
debian@nodo-master:~$ kubectl get svc -n monitor
NAME                                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
alertmanager-operated                     ClusterIP   None             <none>        9093/TCP,9094/TCP,9094/UDP   24h
prometheus-operated                       ClusterIP   None             <none>        9090/TCP                     24h
prometheus-operator-alertmanager          NodePort    10.96.74.153     <none>        9093:32308/TCP               24h
prometheus-operator-grafana               NodePort    10.110.147.108   <none>        80:30646/TCP                 24h
prometheus-operator-kube-state-metrics    ClusterIP   10.99.22.245     <none>        8080/TCP                     24h
prometheus-operator-operator              ClusterIP   10.96.111.57     <none>        8080/TCP,443/TCP             24h
prometheus-operator-prometheus            NodePort    10.101.37.188    <none>        9090:30319/TCP               24h
prometheus-operator-prometheus-node-exporter   ClusterIP   10.108.8.66      <none>        9100/TCP                     24h
```

Comprobamos el acceso desde el navegador:

user: admin password: prom-operator







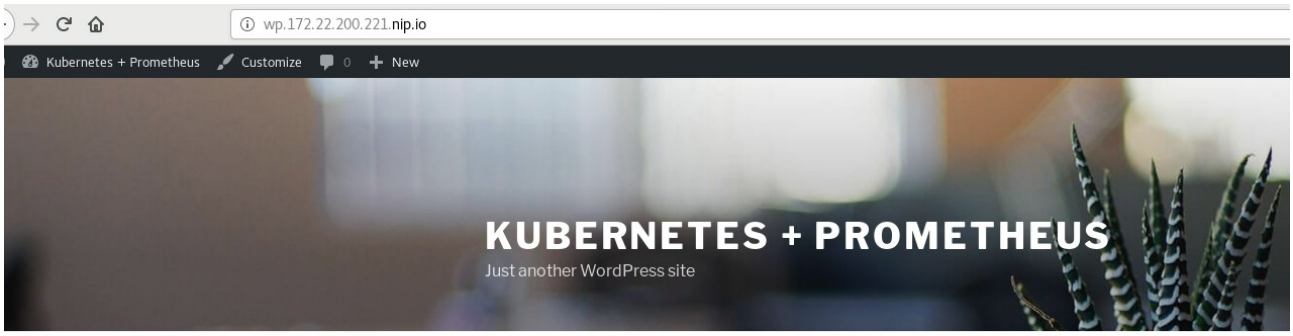
Pruebas de funcionamiento

Test Pod

Realizaremos una prueba en kubernetes para comprobar la alta disponibilidad.

En este caso borraremos un pod y comprobaremos el estado de nuestros despliegues.

Primero comprobamos que wordpress está operativo.



Comprobamos los pods de namespace de wordpress:

```
debian@nodo-master:~$ kubectl get pods -n wordpress
NAME                         READY   STATUS    RESTARTS   AGE
wordpress-9d9b6f4f5-ht9tc          1/1     Running   1          5d16h
wordpress-mysql-65586b5d9b-9mdrw   1/1     Running   2          5d18h
```

Borramos un pod:

```
debian@nodo-master:~$ kubectl delete pod wordpress-9d9b6f4f5-ht9tc -n wordpress
pod "wordpress-9d9b6f4f5-ht9tc" deleted
```

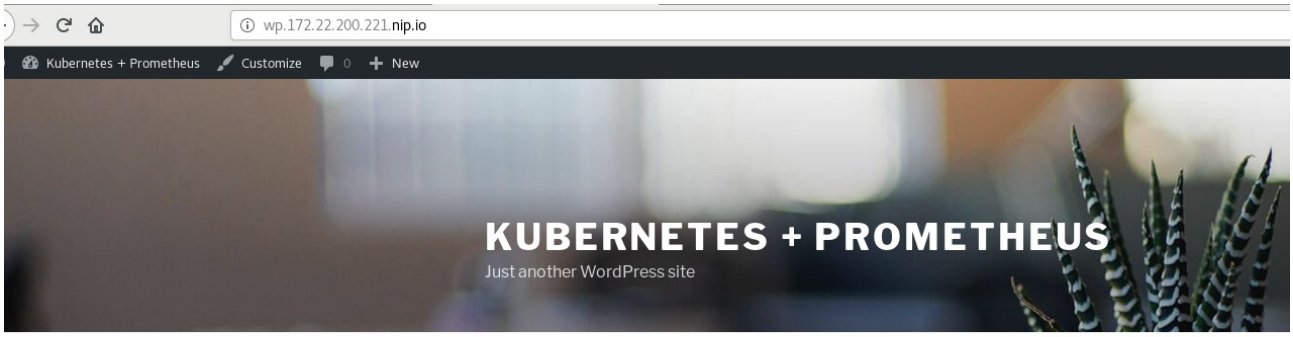
Comprobamos los pods:

```
debian@nodo-master:~$ kubectl get pods -n wordpress
NAME                         READY   STATUS    RESTARTS   AGE
wordpress-9d9b6f4f5-kbzmc          1/1     Running   0          50s
wordpress-mysql-65586b5d9b-9mdrw   1/1     Running   2          5d18h
```

El pod se ha vuelto a crear automaticamente ya que tenemos un ReplicaSet que obliga a mantener el pod.

```
debian@nodo-master:~$ kubectl get rs -n wordpress
NAME                   DESIRED   CURRENT   READY   AGE
wordpress-9d9b6f4f5          1         1         1       5d16h
wordpress-mysql-65586b5d9b   1         1         1       5d18h
```

Recargamos el navegador y comprobamos:

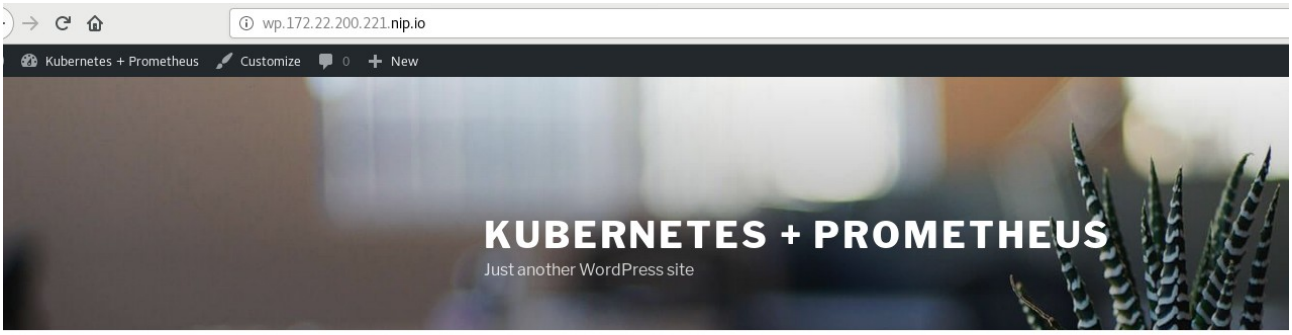


Comprobamos en grafana que el pod eliminado dejar de consumir cpu y memoria:

Comprobamos el nuevo pod:

Test PersistentVolumen

Comprobamos que tenemos acceso al Wordpress:



A continuación vamos a eliminar el pod de la base de datos, hay que tener en cuenta que cuando se elimina el pod se eliminan todos sus containers, es decir, pierde todos sus datos. En este caso no se pierden debido a que estos datos se guardan en PersistentVolumen.

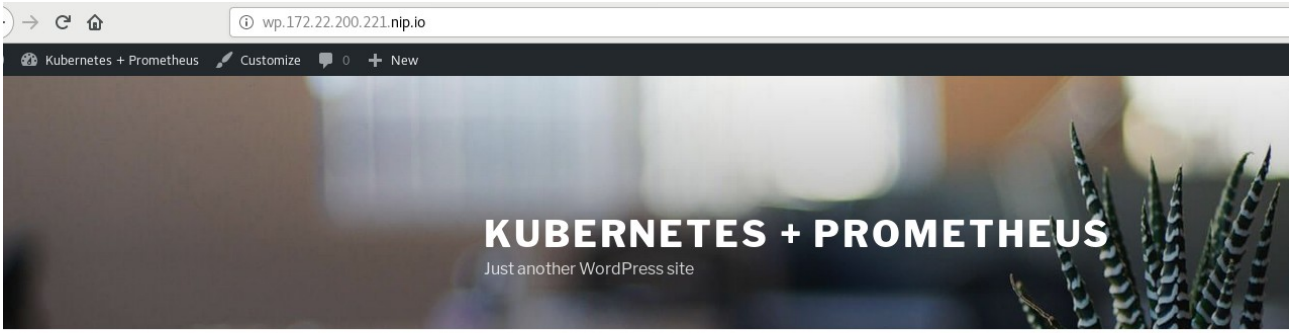
Comprobamos wordpress justo después de eliminar el pod para comprobar su estado:

Error establishing a database connection

Comprobamos que el pod se ha vuelto a crear debido al ReplicaSet:

```
debian@nodo-master:~$ kubectl get pods -n wordpress
NAME                         READY   STATUS    RESTARTS   AGE
wordpress-9d9b6f4f5-t5r8g         1/1     Running   0          25m
wordpress-mysql-65586b5d9b-tplbx   1/1     Running   0          75s
```

Comprobamos que volvemos a tener acceso a wordpress y no hemos perdido ningún dato de la base de datos.



Test Node

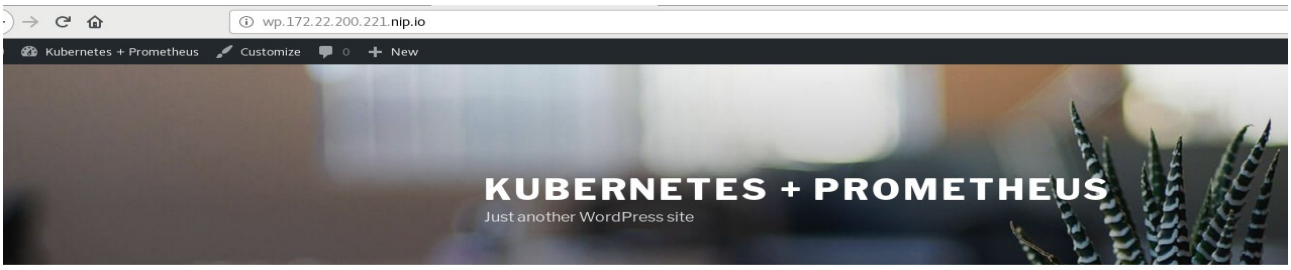
En esta prueba vamos a apagar un nodo y comprobar que las aplicaciones siguen funcionando debido a la alta disponibilidad que nos aporta kubernetes, ya que todos los procesos y servicios se ejecutan en los nodos restantes.

☐	nodo_master	Debian Stretch 9.11	• 10.0.0.11 IPs flotantes: • 172.22.201.31	ssd.large	clave_ecdsa	Activo	nova
☐	Nodo_2	Debian Stretch 9.11	• 10.0.0.7 IPs flotantes: • 172.22.200.222	m1.medium	clave_ecdsa	Apagada	nova
☐	Nodo_1	Debian Stretch 9.11	• 10.0.0.3 IPs flotantes: • 172.22.200.221	m1.medium	clave_ecdsa	Activo	nova

Displaying 3 items

```
debian@nodo-master:~$ kubectl get nodes
NAME          STATUS     ROLES    AGE   VERSION
nodo-1        Ready      <none>   19d   v1.16.3
nodo-2        NotReady   <none>   19d   v1.16.3
nodo-master   Ready      master   20d   v1.16.3
```

Comprobamos el acceso a wordpress:



Comprobamos las alerta de Prometheus que nos indican que un nodo no está funcionando:

/etc/prometheus/rules/prometheus-prometheus-operator-prometheus-rulefiles-0/monitor-prometheus-operator-kubernetes-system-kubelet.yaml > kubernetes-system-kubelet

KubeNodeUnreachable (1 active)

KubeNodeNotReady (1 active)

KubeletDown (0 active)

KubeletTooManyPods (0 active)

/etc/prometheus/rules/prometheus-prometheus-operator-prometheus-rulefiles-0/monitor-prometheus-operator-kubernetes-system-scheduler.yaml > kubernetes-system-scheduler

También comprobamos grafana que nos indica el estado del nodo:

Iniciamos el nodo y comprobamos grafana:

En conclusión

Para terminar podemos comprobar o entender la eficacia, funcionalidad y alcance de Kubernetes + Prometheus.

Lo importante que puede llegar a ser en un entorno de trabajo por su ayuda y importancia a la hora de entender o comprender lo que ocurre en todo momento es nuestro proyecto.