



ÍNDICE

- **Introducción**
 - ¿Qué es Kubernetes?
 - ¿Qué ofrece?
- **Arquitectura**
 - Clústers
 - Nodes
 - Master
 - API-server
 - Scheduler
 - Controller-manager
 - etcd
 - Workers
 - Container-runtime
 - Kubelet
 - Kube Proxy
 - Pods
- **Instalación**
 - Requisitos hardware
 - Elementos

- **Minikube**
 - **Comandos básicos minikube**
- **Replicasets**
 - **Crear**
 - **Escalar**
- **Deployments**
 - **Crear**
 - **Actualizar**
 - **Historial/rollout**
 - **Escalar app**
- **Services/Endpoints**
 - **Agrupación pods en services**
 - **Tipos services**
 - **Desplegar services**
- **Namespaces**
 - **Administración**
- **Volumes**
 - **Tipos volumes**
 - **Creación volumes**
 - **StorageClass**
- **ConfigMaps y variables de entorno**
 - **Variables entorno**
 - **ConfigMaps**
- **Secrets**
- **RBAC**
 - **Usuarios**
 - **Permisos**
 - **Role**
 - **RoleBinding**

- [ClusterRole](#)
 - [ClusterRoleBinding](#)
 - [ServiceAccount](#)
 - [Ingress](#)
 - [Ingress Controller](#)
 - [Instalación y configuración](#)
 - [Creación clúster local](#)
 - [Proveedores de servicio](#)
 - [Google Kubernetes Engine \(GKE\)](#)
 - [Amazon Elastic Kubernetes Service \(EKS\)](#)
 - [Azure Kubernetes Service \(AKS\)](#)
 - [Anexo](#)
 - [Comandos básicos kubectl](#)
 - [Helm](#)
 - [Bibliografía](#)
-

INTRODUCCIÓN

Somos Roberto Martínez y Alejandro López, dos alumnos de Administración de Sistemas Informáticos en Red.

Hemos seleccionado Kubernetes para nuestro proyecto final porque creemos que actualmente es la tecnología puntera en virtualización y alta disponibilidad, lo cual nos puede ser de mucha utilidad en nuestra vida laboral.

¿Qué es Kubernetes?

Kubernetes (timonel o piloto en griego) es un software de orquestación de código abierto que permite implementar, administrar y escalar aplicaciones en unidades lógicas para gestionarlas y darles visibilidad. Tiene un ecosistema grande y en rápido crecimiento. El soporte, las herramientas y los servicios para Kubernetes están ampliamente disponibles.

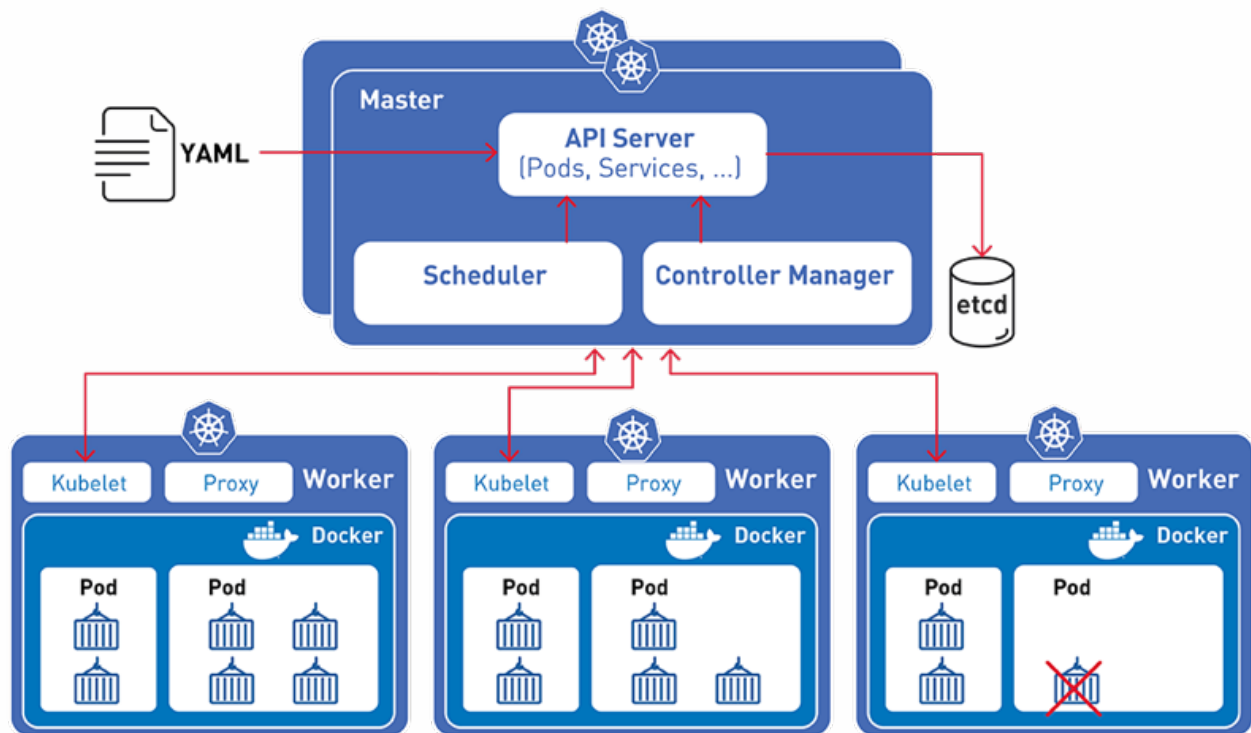
Kubernetes ofrece un entorno de administración centrado en contenedores, orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Fue diseñado por Google en 2014 y su diseño estuvo influenciado por el proyecto Borg (nombre que hace referencia a una raza de humanoides de la serie de ciencia ficción Star Trek) y donado a la Cloud Native Foundation.

¿Qué ofrece?

Las principales características de K8s (abreviatura de Kubernetes) son las siguientes:

- **Escalado.** Kubernetes permite escalar nuestras aplicaciones en función del número de réplicas que necesitemos de manera automática (autoescalado) o manual (comando).
- **Reparación automática.** En caso de ocurrir un fallo en un contenedor podemos reiniciar automáticamente, así como reemplazarlo o replanificarlo cuando un nodo muere.
- **Planificación de nodos y contenedores.** Kubernetes nos ayuda a decidir en qué nodo se ejecutará cada contenedor, en función de los recursos necesarios y otras restricciones. Además, podemos mezclar cargas de trabajo críticas y best-effort con el objetivo de potenciar el ahorro de recursos.
- **Despliegues y rollbacks automáticos.** Podemos desplegar los cambios de manera progresiva cuando actualizamos una aplicación o cambiamos su configuración, y así poder realizar un rollback automático en caso de fallo en alguna de las instancias.
- **Almacenamiento persistente.** Kubernetes permite montar de forma automática el sistema de almacenamiento necesario para los contenedores: en modo local, en un proveedor de cloud, o en un sistema de red como NFS, Flocker o Gluster. Tiene soporte en plataformas como Amazon Web Services o Google Cloud Platform, y los proveedores (Red Hat, Dell EMC, NetApp, etc) le proporcionan almacenamiento persistente.
- **Descubrimiento de servicios.** Con Kubernetes asignamos a los contenedores sus propias direcciones IP y un nombre DNS específico para cada conjunto de contenedores. Así, no resulta necesario utilizar recursos externos para el descubrimiento de servicios.
- **Seguridad.** Toda nuestra información sensible, como contraseñas o claves ssh, podemos almacenarla de manera segura en secrets. En este sentido, Kubernetes no expone nuestra información confidencial a la hora de desplegar y configurar nuestras aplicaciones.
- **Clústers grandes y heterogéneos.** Kubernetes puede desplegarse en clusters muy grandes, incluyendo contenedores Docker. Por otro lado, nos permite crear un clúster como resultado de combinar diferentes máquinas virtuales o servidores locales.

ARQUITECTURA



Clústers

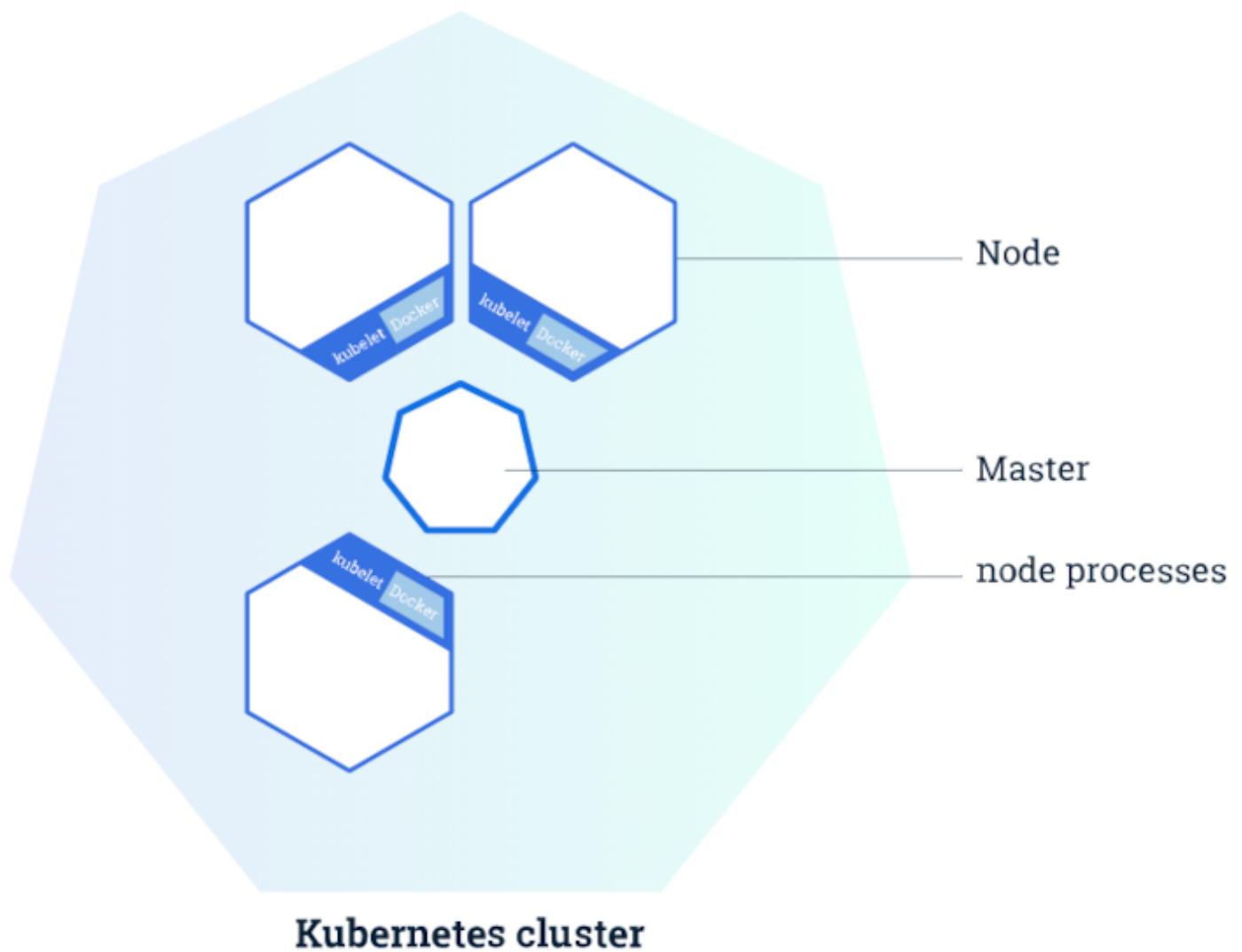
Un clúster de Kubernetes es un conjunto de máquinas de nodos que ejecutan aplicaciones en contenedores. Si ejecuta Kubernetes, está ejecutando un clúster.

Como mínimo, un clúster contiene un master y una o varias máquinas informáticas o nodos. El control plane es el encargado de mantener el estado deseado del clúster y de controlar, por ejemplo, las aplicaciones que se ejecutan y las imágenes de contenedores que se utilizan. Los nodos son los que realmente ejecutan las aplicaciones y las cargas de trabajo.

La ventaja más importante de Kubernetes es la capacidad de programar y ejecutar contenedores en un grupo de máquinas, ya sean físicas o virtuales, en local o en un cloud, y esto es posible gracias al clúster. Los contenedores de Kubernetes no están ligados a ninguna máquina individual. De hecho, están aislados en el clúster.

Podemos crear clústers en local, en el cloud, híbridos y también minikube, virtualización de un clúster en un solo nodo virtual que se utiliza para hacer tests y que luego explicaremos con más detalle.

Nodes



Un nodo es una máquina de trabajo en Kubernetes, previamente conocida como minion. Un nodo puede ser una máquina virtual o física, dependiendo del tipo de clúster. Hay dos tipos de nodos: master o control plane y workers. Cada uno de ellos ejecuta diferentes procesos según su clasificación:

- **Node master:**

Aquí se encuentran los elementos de Kubernetes que controlan el clúster, junto con los datos sobre su estado y configuración. Los elementos principales de Kubernetes tienen la importante tarea de garantizar que los contenedores se ejecuten en cantidades suficientes y con los recursos necesarios. El plano de control está en contacto permanente con las máquinas informáticas. Garantiza que el clúster se ejecute según la configuración que hayamos elegido. Consta de diferentes procesos:

- **API-Server:** Es el componente que interactúa con el cliente. Se trata del frontend de Kubernetes, recibe las peticiones y actualiza acordemente el estado en etcd.
- **Scheduler:** Este proceso se encarga de decidir en qué nodo se ejecutarán los pods. Para ello tiene en cuenta los siguientes factores: requisitos de recursos, restricciones de hardware/software/políticas, afinidad y anti-afinidad, localización de datos dependientes, entre otros.
- **Controller-manager:** Es el componente que ejecuta los controles de k8s. Cada

controlador es un proceso independiente, pero para reducir la complejidad, todos se compilan en un único binario y se ejecuta en un mismo proceso. Estos controladores incluyen:

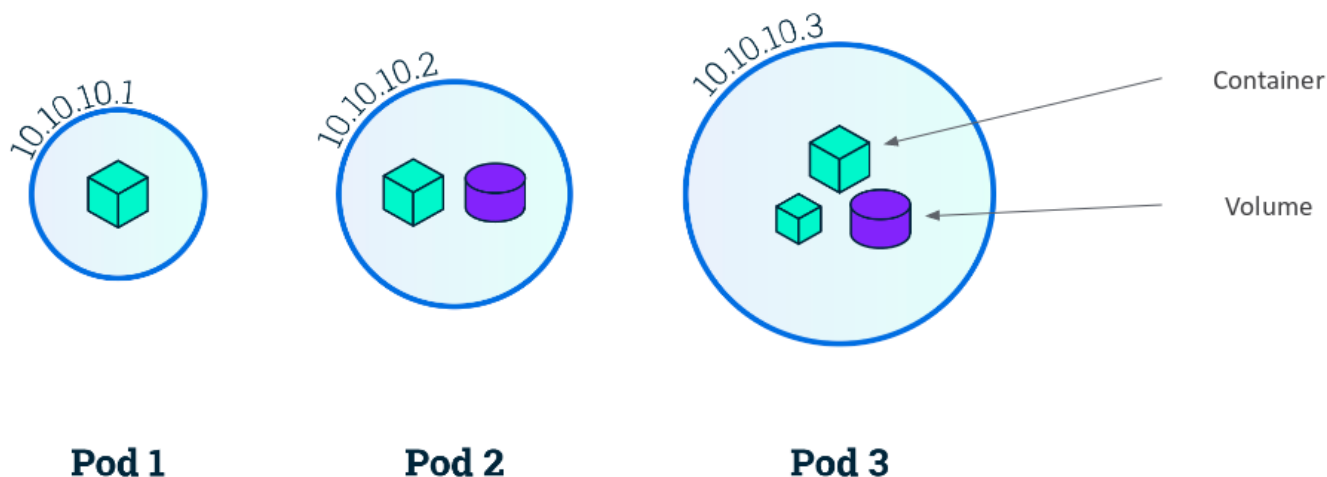
- **Controlador de nodo:** Es el responsable de detectar y responder cuándo un nodo deja de funcionar.
 - **Controlador de replicación:** Es el responsable de mantener el número correcto de pods para cada controlador de replicación del sistema.
 - **Controlador de endpoints:** Construye el objeto Endpoints, es decir, hace una unión entre los Services y los Pods.
 - **Controladores de tokens y cuentas de servicio:** Crean cuentas y tokens de acceso a la API por defecto para los nuevos Namespaces
- **Etc**: Es la base de datos donde se guarda toda la información que utiliza el clúster.

- **Node worker:**

El worker node, nos proporcionará un entorno de ejecución para las aplicaciones. Estas aplicaciones que se encuentran contenerizadas en pods y son controladas por los anteriores procesos que hemos descrito del Control Plane que se ejecutan en el master node.

- **Container runtime:** Es el software responsable de la ejecución y gestión de los contenedores.
- **Kubelet:** Es el agente que se ejecuta en cada nodo de un clúster y se comunica con los componentes del control plane. Recibe las definiciones del pod del API Server e interacciona con el container runtime para ejecutar contenedores asociados al pod.
- **Kube-proxy:** Es la implementación de un proxy de red y balanceador de carga soportando la abstracción del servicio junto con otras operaciones de red. Es responsable del enrutamiento del tráfico hacia el contenedor correcto basado en la dirección IP y el número de puerto indicados por el control plane

Pods



Un pod es un grupo de uno o más contenedores, con almacenamiento/red compartidos, y unas especificaciones de cómo ejecutar los contenedores. Los contenidos de un pod son siempre coubicados, coprogramados y ejecutados en un contexto compartido. Un pod modela un "host lógico" específico de la aplicación: contiene uno o más contenedores de aplicaciones relativamente entrelazados.

Los contenedores dentro de un pod comparten dirección IP y puerto, y pueden encontrarse a través de localhost. También pueden comunicarse entre sí mediante comunicaciones estándar entre procesos. Los contenedores en nodos diferentes, normalmente se comunican entre sí a través de las IP's.

Las aplicaciones dentro de un pod también tienen acceso a volúmenes compartidos, que se definen como parte de un pod y están disponibles para ser montados en el sistema de archivos de cada aplicación.

Al igual que los contenedores de aplicaciones individuales, los pods se consideran entidades relativamente efímeras (en lugar de duraderas). Como se explica en ciclo de vida del pod, los pods se crean, se les asigna un identificador único (UID) y se planifican en nodos donde permanecen hasta su finalización (según la política de reinicio) o supresión. Si un nodo muere, los pods programados para ese nodo se programan para su eliminación después de un período de tiempo de espera. Un pod dado (definido por su UID) no se "replanifica" a un nuevo nodo; en su lugar, puede reemplazarse por un pod idéntico, con incluso el mismo nombre si lo desea, pero con un nuevo UID

Los pods pueden ser usados para alojar pilas de aplicaciones integradas (por ejemplo, LAMP), pero su objetivo principal es apoyar los programas de ayuda coubicados y coadministrados, como:

- Sistemas de gestión de contenido, loaders de datos y archivos, gestores de caché locales, etc.
- Copia de seguridad de registro y punto de control, compresión, rotación, captura de imágenes, etc.
- Observadores de cambio de datos, adaptadores de registro y monitoreo, publicadores de eventos, etc.
- Proxies, bridges y adaptadores.

- Controladores, configuradores y actualizadores.

Los pods individuales no están diseñados para ejecutar varias instancias de la misma aplicación, en general.

INSTALACIÓN

Requisitos hardware

- Sistema operativo:
 - Ubuntu 16.+
 - Debian 9+
 - CentOS 7+
 - Red Hat Enterprise Linux (RHEL)7
 - Fedora25+
 - HyprIoTOS v1.0.1+
 - etc.
- 2GB mínimo de RAM para el masters nodes
- 1GB mínimo de RAM para workers nodes
- 2 CPUs mínimo

Elementos

Necesitamos tener instalado previamente **docker** para poder trabajar en Kubernetes.

<https://docs.docker.com/engine/install/>

Debemos instalar los siguientes componentes:

- **kubecttl**: Aplicación cliente que interactúa con el cluster a través del API a través del API-Server y mediante la línea de comandos.
- **kubeadm**: Instrucción que nos permite crear el cluster.
- **kubelet**: Es el responsable del estado de ejecución de cada nodo. Se encarga del inicio, la detención y el mantenimiento de los contenedores de aplicaciones (organizados como pods) como es indicado por el master.
- **minikube**: Entorno de pruebas para aprender kubernetes. Es un cluster de un solo nodo lanzado a través de una máquina virtual

```
$ dnf -y install kubeadm kubelet kubecttl
```

Minikube. Instalación en Fedora 32

Para trabajar con ambientes de Kubernetes normalmente es más sencillo poder instalar un cluster local en nuestro equipo que nos permita probar de una forma más expedita sin tener que interactuar con un servidor externo. Una de las herramientas que nos sirve para esta tarea es Minikube.



minikube

Prerequisitos

Para instalar Minikube, se debe validar los siguientes requisitos:

Validar que la virtualización está soportada en el linux

```
$ grep -E --color 'vmx|svm' /proc/cpuinfo
```

Tener instalado Kubectl

```
$ kubectl version --client
```

Si no lo tenemos instalado, lo instalamos.

Para instalar Kubectl vamos a habilitar un YUM repo de Google e instalar el paquete kubectl.

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

$ sudo dnf makecache
```

```
$ sudo dnf install kubectll kubelet kubeadm
```

Tener instalado un hipervisor

Se requiere tener instalado un hipervisor, el Minikube soporta varios pero en este caso vamos a instalar KVM. Para revisar si lo tenemos instalado podemos validar si está habilitado el módulo en el Kernel ejecutando el comando:

```
$ lsmod | grep kvm
```

En caso de no tenerlo instalado, lo instalamos.

Para instalarlo, vamos a instalar los siguientes paquetes:

Instala el KVM

```
$ sudo dnf install bridge-utils libvirt virt-install qemu-kvm
```

Otras herramientas útiles para el KVM

```
$ sudo dnf install virt-top libguestfs-tools
```

Habilita el servicio

```
$ sudo systemctl enable --now libvirtd
```

Instala un administrador gráfico

```
$ sudo dnf install virt-manager
```

Agrega nuestro usuario al grupo del KVM

```
$ sudo usermod -a -G libvirt $(whoami)
```

Instalación del binario de Minikube

Descarga el binario y le asigna permisos de ejecución

```
$ curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/  
&& chmod +x minikube
```

Intenta crear esta ruta en caso de que no exista

```
$ sudo mkdir -p /usr/local/bin/
```

Instala el binario del minikube en la ruta

```
$ sudo install minikube /usr/local/bin/
```

Elimina el binario descargado

```
$ rm minikube
```

Para probar si todo se instaló bien, podemos crear un cluster de Minikube usando KVM con los siguientes comandos:

Crear un cluster de minikube (default)

```
$ minikube start --driver=kvm2
```

Consulta el estado

```
$ minikube status
```

```
Para el clúster  
$ minikube stop
```

```
Elimina el minikube que se creó para probar  
$ minikube delete
```

Por defecto va a crear un cluster de un nodo con la última versión disponible de Kubernetes, al nodo se le asignarán los recursos:

- vCPU = 2
- Memoria = 6000MB
- Disco = 20000MB

Crear un cluster minikube personalizado

Con el siguiente comando podemos crear un cluster personalizado de 4 vCPUs, 4 GB de memoria RAM y 20GB de disco duro, usando la versión 1.17.6 de Kubernetes.

```
Muestra información de los parámetros disponibles para el start  
$ minikube help start
```

```
Crea minikube personalizado  
$ minikube start --driver=kvm2 \  
  --cpus=4 \  
  --memory=4096 \  
  --disk-size=20000mb \  
  --kubernetes-version=1.17.6
```

Para calcular la memoria RAM, $1024 * \text{los GB que queramos}$ Ejemplo de 4GB de memoria RAM

```
1024*4= 4096
```

Comandos básicos minikube

```
Encender  
$ minikube start
```

```
Detener  
$ minikube stop
```

```
Pausar  
$ minikube pause
```

```
Reanudar  
$ minikube unpause
```

```
Status  
$ minikube status
```

```
Obtener url service  
$ minikube service [service_name]
```

```
Obtener IP clúster  
$ minikube ip
```

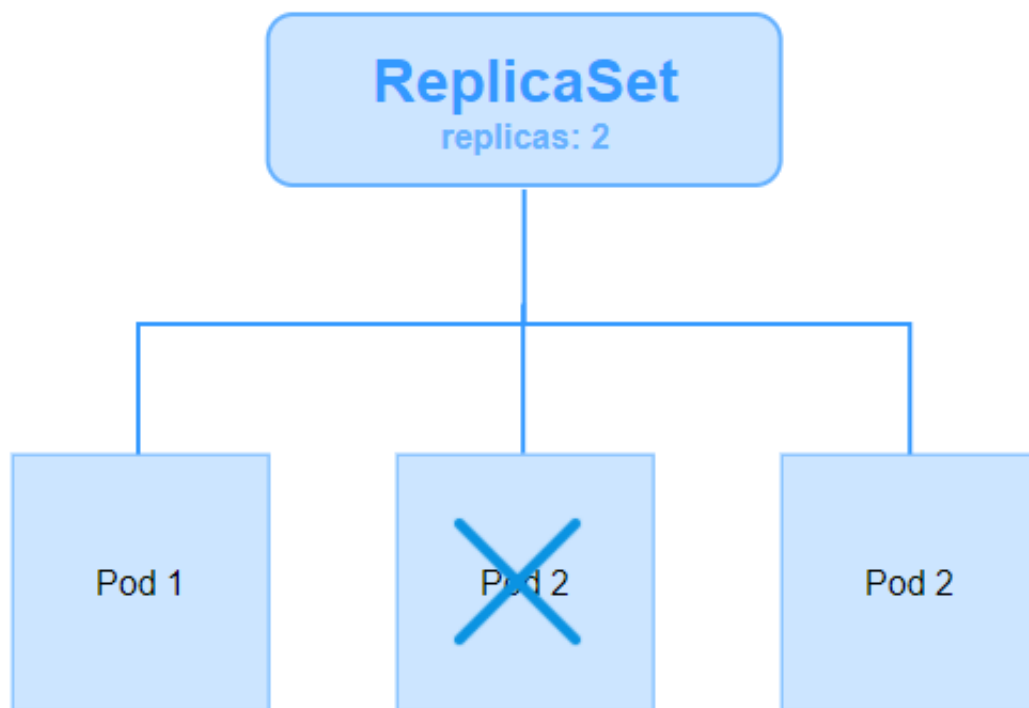
```
Copiar fichero al clúster  
$ minikube cp [path/file]
```

```
Versión  
$ minikube version
```

```
Logs  
$ minikube logs
```

```
Eliminar  
$ minikube delete
```

REPLICASETS



Es el componente que se encarga de mantener el número de réplicas de los pods activos. Alcanza su propósito mediante la creación y eliminación de pods que sea necesario para alcanzar el número que deseamos.

El enlace que un replicaset tiene hacia sus pods es a través del campo del Pod denominado `metadata.ownerReferences`, el cual indica qué recurso es el propietario del objeto actual.

Un replicaset garantiza que un número específico de réplicas de un pod se está ejecutando en

todo momento. Sin embargo, un deployment es un concepto de más alto nivel que gestiona replicaset y proporciona actualizaciones de forma declarativa de los pods junto con muchas otras características útiles.

Por lo tanto, se recomienda el uso de deployments en vez del uso directo de replicaset, a no ser que se necesite una orquestación personalizada de actualización o no se necesite las actualizaciones en absoluto.

Crear replicaset

Realizamos un ejemplo de la utilidad de este objeto:

Analizamos el fichero de configuración

```
$ vim frondend.yaml
  apiVersion: apps/v1
  kind: ReplicaSet
  metadata:
    name: frontend
    labels:
      app: guestbook
      tier: frontend
  spec:
    # modifica las réplicas según tu caso de uso
    replicas: 5
    selector:
      matchLabels:
        tier: frontend
    template:
      metadata:
        labels:
          tier: frontend
      spec:
        containers:
          - name: php-redis
            image: gcr.io/google_samples/gb-frontend:v3
```

A través del fichero ymal lanzamos la configuración de la réplicas y kubernetes lo crea de manera automatizada.

```
$ kubectl apply -f froned.yaml
replicaset.apps/frontend created
```

Comprobamos el estado del replicaset.

```
$ kubectl get replicaset
NAME          DESIRED   CURRENT   READY   AGE
frontend      5         5         5       16m
```

Y también la información más detallada.

```
$ kubectl describe rs/frontend
```

```

Name:          frontend
Namespace:     default
Selector:      tier=frontend
Labels:        app=guestbook
               tier=frontend
Annotations:   <none>
Replicas:      5 current / 5 desired
Pods Status: 5 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:      tier=frontend
  Containers:
    php-redis:
      Image:     gcr.io/google_samples/gb-frontend:v3
      Port:      <none>
      Host Port: <none>
      Environment: <none>
      Mounts:     <none>
      Volumes:    <none>
Events:
  Type          Reason          Age    From          Message
  ---          -
  Normal        SuccessfulCreate 17m    replicaset-controller Created pod: fr
  Normal        SuccessfulCreate 17m    replicaset-controller Created pod: fr
  Normal        SuccessfulCreate 17m    replicaset-controller Created pod: fr
  Normal        SuccessfulCreate 17m    replicaset-controller Created pod: fr
  Normal        SuccessfulCreate 17m    replicaset-controller Created pod: fr

```

Escalar pods con replicaset

Podemos escalar el número de réplicas del pod que hemos lanzado en caliente. Para ellos podemos modificar el fichero .yaml y cambiar el número de réplicas, en este caso vamos a reducir a 3.

```

$ vim frondend.yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3

$ kubectl apply -f fronted.yaml
replicaset.apps/frontend configured

```

Comprobamos cómo ha cambiado el número de pods que está corriendo en el orquestador.

```
$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
frontend-4nwkk      1/1     Running   0           20m
frontend-n672g      1/1     Running   0           20m
frontend-pn728      1/1     Running   0           20m

$ kubectl get rs
NAME        DESIRED   CURRENT   READY   AGE
frontend    3          3          3       21m
```

También podemos ejecutar la misma acción a través de los comandos de kubectl sin necesidad de modificar el fichero de configuración. En este caso volveremos a tener 3 réplicas de nuevo.

```
$ kubectl scale replicaset frontend --replicas=5
replicaset.apps/frontend scaled

$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
frontend-4nwkk      1/1     Running   0           21m
frontend-kbvvs      1/1     Running   0           25s
frontend-kwdkl      1/1     Running   0           25s
frontend-n672g      1/1     Running   0           21m
frontend-pn728      1/1     Running   0           21m

$ kubectl get rs
NAME        DESIRED   CURRENT   READY   AGE
frontend    5          5          5       22m
```

Eliminar replicaset

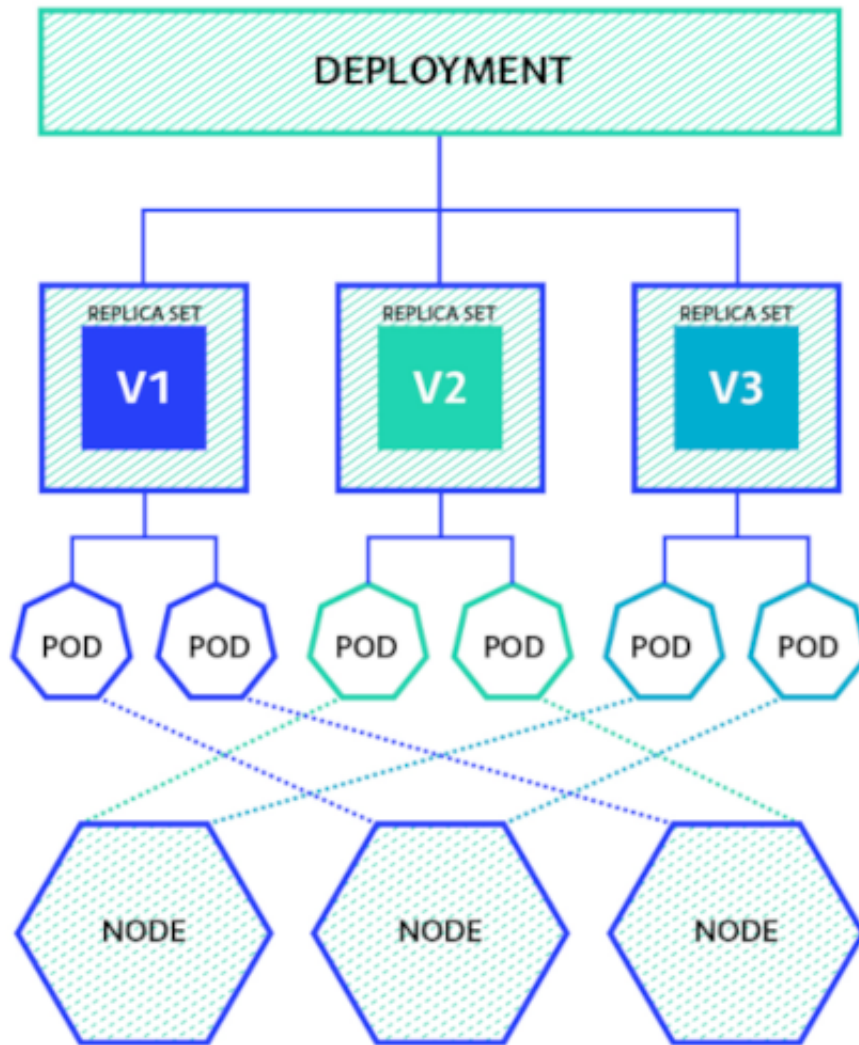
Eliminamos el replicaset

```
$ kubectl delete replicaset frontend
replicaset.apps "frontend" deleted

$ kubectl get rs
No resources found in default namespace.
```

DEPLOYMENTS

Una configuración de deployment pide a Kubernetes que cree y actualice las instancias de una aplicación. Tras crear el deployment, el control plane organiza las instancias de aplicación en los nodos disponibles del cluster.



Una vez creadas las instancias de aplicación, el controlador de Deployment de Kubernetes monitoriza continuamente las instancias. Si un nodo en el que está una instancia cae o es eliminado, el controlador sustituye la instancia por otra en otro nodo disponible del cluster.

Esta característica de recuperación de fallos mediante la creación de nuevas instancias que reemplazan a las defectuosas o desaparecidas no existía antes de los orquestadores.

Al crear un deployment se especifica la imagen del contenedor que usará la aplicación y el número de réplicas que se quieren mantener en ejecución. El número de réplicas se puede modificar en cualquier momento actualizando el deployment.

Crear deployments

Vamos a crear un ejemplo de un deployment que crea un replicaset de pods de un servidor web nginx.

```
$ vim deployment-nginx
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: nginx-deployment
```

```

labels:
  app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.7.9
          ports:
            - containerPort: 80

```

Creamos un Deployment denominado nginx-deployment, indicado a través del campo **.metadata.name**.

El deployment crea tres pods replicados, indicado a través del campo **replicas**.

El campo selector define cómo el deployment identifica los pods que debe gestionar. En este caso, simplemente seleccionamos una etiqueta que se define en la plantilla Pod (app: nginx).

matchLabels es un mapa de entradas {clave,valor}.

El campo template contiene los siguientes subcampos:

- Los Pods se etiquetan como app: nginx usando el campo **labels**.
- La especificación de la plantilla Pod, o el campo **.template.spec**, indica que los pods ejecutan un contenedor, nginx, que utiliza la versión 1.7.9 de la imagen de nginx.
- Crea un contenedor y lo llamar nginx usando el campo **name**.
- Ejecuta la imagen nginx en su versión 1.7.9.
- Abre el puerto 80 para que el contenedor pueda enviar y recibir tráfico.

Creamos el deployment ejecutando el cliente kubectl.

```

$ kubectl apply -f nginx-deployment.yaml
deployment.apps/nginx-deployment created

```

Inspeccionamos el deployment lanzado en nuestro clúster con los siguientes comandos.

```

$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    3/3     3             3           103s

$ kubectl rollout status deployment nginx-deployment
deployment "nginx-deployment" successfully rolled out

```

```
$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-5d59d67564-7fs5x	1/1	Running	0	5m44s
nginx-deployment-5d59d67564-csbz7	1/1	Running	0	5m44s
nginx-deployment-5d59d67564-mv1ml	1/1	Running	0	5m44s

También podemos ver las etiquetas (**labels**) creadas automáticamente.

```
$ kubectl get pods --show-labels
```

NAME	READY	STATUS	RESTARTS	AGE	Labels
nginx-deployment-5d59d67564-7fs5x	1/1	Running	0	10m	app=nginx
nginx-deployment-5d59d67564-csbz7	1/1	Running	0	10m	app=nginx
nginx-deployment-5d59d67564-mv1ml	1/1	Running	0	10m	app=nginx

Actualizar deployments

Actualizamos la versión de nuestra app de la version nginx:1.7.9 a nginx:1.9.1.

```
$ kubectl --record deployment.apps/nginx-deployment set image deployment.v1.
deployment.apps/nginx-deployment image updated
deployment.apps/nginx-deployment image updated
```

También podemos editar el fichero .yaml. Ahora vamos a volver a la versión inicial.

```
$ kubectl edit deployment.v1.apps/nginx-deployment
# Please edit the object below. Lines beginning with a '#' will be ignored
# and an empty file will abort the edit. If an error occurs while saving
# the file will be reopened with the relevant failures.
#
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
    deployment.kubernetes.io/revision: "3"
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"apps/v1","kind":"Deployment","metadata":{"annotations":{"deployment.kubernetes.io/revision":"3"},"name":"nginx-deployment","namespace":"default"},"spec":{"progressDeadlineSeconds":600,"replicas":3,"revisionHistoryLimit":10,"selector":{"matchLabels":{"app":"nginx"},"strategy":{"rollingUpdate":{"maxSurge":"25%","maxUnavailable":"25%"},"type":"RollingUpdate"},"type":"RollingUpdate"},"template":{"metadata":{"creationTimestamp":"2021-04-29T07:46:09Z"},"spec":{"containers":[{"image":"nginx:1.9.1","name":"nginx","resources":{}}]}}}}
    kubernetes.io/change-cause: kubectl deployment.apps/nginx-deployment deployment.v1.apps/nginx-deployment nginx=nginx:1.9.1 --record=true
  creationTimestamp: "2021-04-29T07:46:09Z"
  generation: 3
  labels:
    app: nginx
  name: nginx-deployment
  namespace: default
  resourceVersion: "6681"
  uid: 6d59839a-fcbc-4b98-ae8f-971299e7a40e
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
```

```

    maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
      labels:
        app: nginx
    spec:
      containers:
        - image: nginx:1.7.9
          imagePullPolicy: IfNotPresent
          name: nginx
          ports:
            - containerPort: 80
              protocol: TCP
          resources: {}
          terminationMessagePath: /dev/termination-log
          terminationMessagePolicy: File
      dnsPolicy: ClusterFirst
      restartPolicy: Always
      schedulerName: default-scheduler
      securityContext: {}
      terminationGracePeriodSeconds: 30
  status:
    availableReplicas: 3
    conditions:
      - lastTransitionTime: "2021-04-29T07:46:21Z"
        lastUpdateTime: "2021-04-29T07:46:21Z"
        message: Deployment has minimum availability.
        reason: MinimumReplicasAvailable
        status: "True"

```

Comprobamos el estado del deployment y de los pods (debemos tener 3 réplicas).

```

$ kubectl rollout status deployment.v1.apps/nginx-deployment
deployment "nginx-deployment" successfully rolled out

$ kubectl get deployments
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment    3/3     3             3           113m

$ kubectl get pods
nginx-deployment-5d59d67564-4kqkd    1/1     Running    0           14m
nginx-deployment-5d59d67564-fw5p5v   1/1     Running    0           14m
nginx-deployment-5d59d67564-vfdbn    1/1     Running    0           14m

$ kubectl describe deployments
Name:                nginx-deployment
Namespace:            default
CreationTimestamp:    Thu, 29 Apr 2021 09:46:09 +0200
Labels:               app=nginx
Annotations:          deployment.kubernetes.io/revision: 3
                     kubernetes.io/change-cause: kubectl deployment.apps/nginx-deployment set i
Selector:             app=nginx
Replicas:             3 desired | 3 updated | 3 total | 3 available |
StrategyType:         RollingUpdate
MinReadySeconds:      0
RollingUpdateStrategy: 25% max unavailable, 25% max surge

```

```

Pod Template:
  Labels:  app=nginx
  Containers:
    nginx:
      Image:          nginx:1.7.9
      Port:           80/TCP
      Host Port:      0/TCP
      Environment:    <none>
      Mounts:         <none>
  Volumes:          <none>
Conditions:
  Type            Status  Reason
  ----            -
  Available       True    MinimumReplicasAvailable
  Progressing     True    NewReplicaSetAvailable
OldReplicaSets:  <none>
NewReplicaSet:   nginx-deployment-5d59d67564 (3/3 replicas created)
Events:
  Type            Reason              Age             From
  ----            -
  Normal          ScalingReplicaSet   17m             deployment-controller
  Normal          ScalingReplicaSet   17m             deployment-controller
  Normal          ScalingReplicaSet   17m             deployment-controller
  Normal          ScalingReplicaSet   17m (x2 over 117m) deployment-controller
  Normal          ScalingReplicaSet   17m             deployment-controller
  Normal          ScalingReplicaSet   17m             deployment-controller

```

Historial/rollout

Podemos también revisar el historial de los despliegues realizados y de uno en concreto.

```

$ kubectl rollout history deployment.v1.apps/nginx-deployment
deployment.apps/nginx-deployment
REVISION  CHANGE-CAUSE
2         kubectl deployment.apps/nginx-deployment set image deployment.
3         kubectl deployment.apps/nginx-deployment set image deployment.

$ kubectl rollout history deployment.v1.apps/nginx-deployment --revision=2
deployment.apps/nginx-deployment with revision #2
Pod Template:
  Labels:  app=nginx
          pod-template-hash=69c44dfb78
  Annotations:  kubernetes.io/change-cause:
                kubectl deployment.apps/nginx-deployment set image deployment.v1.a
  Containers:
    nginx:
      Image: nginx:1.9.1
      Port:  80/TCP
      Host Port: 0/TCP
      Environment:  <none>
      Mounts:       <none>
  Volumes:  <none>

```

También podemos cambiar de versión. Vamos a volver a la anterior.

```
$ kubectl rollout undo deployment.v1.apps/nginx-deployment
```

```
deployment.apps/nginx-deployment
```

O especificarlo con un parámetro.

```
$ kubectl rollout undo deployment.v1.apps/nginx-deployment --to-revision=2
deployment.apps/nginx-deployment
```

Escalar pods horizontal

Otra de las funciones que nos ofrece deployment es la de poder escalar los pods del clúster de manera horizontal.

```
$ kubectl scale deployment.v1.apps/nginx-deployment --replicas=10
deployment.apps/nginx-deployment scaled

$ kubectl get deployment
NAME                READY    UP-TO-DATE    AVAILABLE    AGE
nginx-deployment    10/10    10            10           153m

$ kubectl get pods
NAME                                     READY    STATUS    RESTARTS    AGE
nginx-deployment-5d59d67564-4h6rh       1/1     Running   0           47s
nginx-deployment-5d59d67564-4kqkd       1/1     Running   0           53m
nginx-deployment-5d59d67564-4l29n       1/1     Running   0           47s
nginx-deployment-5d59d67564-5kf7l       1/1     Running   0           47s
nginx-deployment-5d59d67564-d4n2h       1/1     Running   0           47s
nginx-deployment-5d59d67564-fwp5v       1/1     Running   0           53m
nginx-deployment-5d59d67564-m4tjs       1/1     Running   0           47s
nginx-deployment-5d59d67564-m9zd9       1/1     Running   0           47s
nginx-deployment-5d59d67564-mnr6b       1/1     Running   0           47s
nginx-deployment-5d59d67564-vfdbn       1/1     Running   0           53m
```

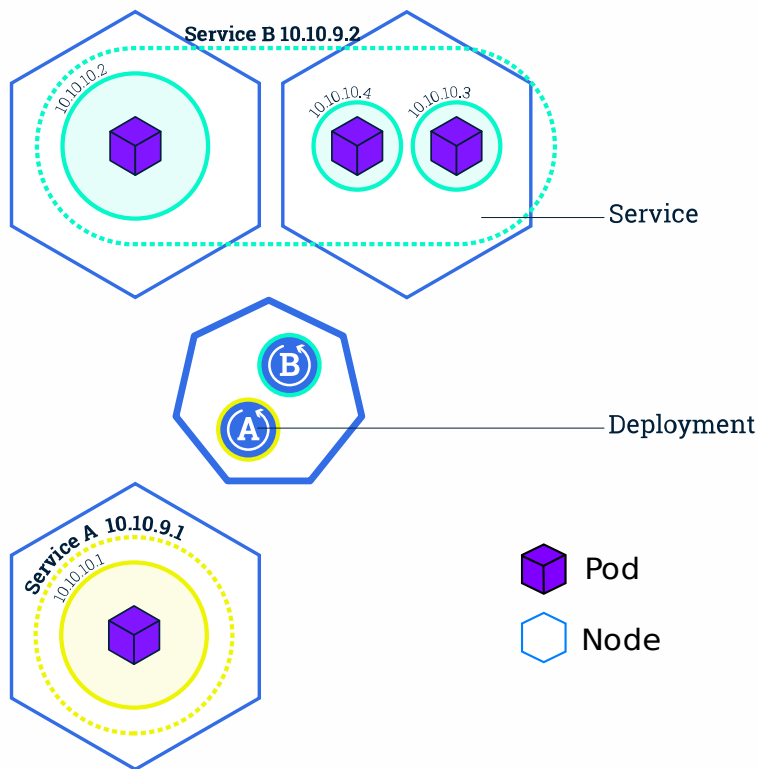
Pausar y reanudar deployments

Para pausar y reanudar un deployment es tan sencillo como ejecutar los siguientes comandos.

```
$ kubectl rollout pause deployment.v1.apps/nginx-deployment
deployment.apps/nginx-deployment paused

$ kubectl rollout resume deployment.v1.apps/nginx-deployment
deployment.apps/nginx-deployment resumed
```

SERVICES/ENDPOINTS



El elemento **service** es el conjunto abstracto de pods que queremos exponer. Balancea la carga entre los diferentes pods. Lo gestiona mediante labels para identificarlos, sin importar que esos pods están en un replicaset u otro.

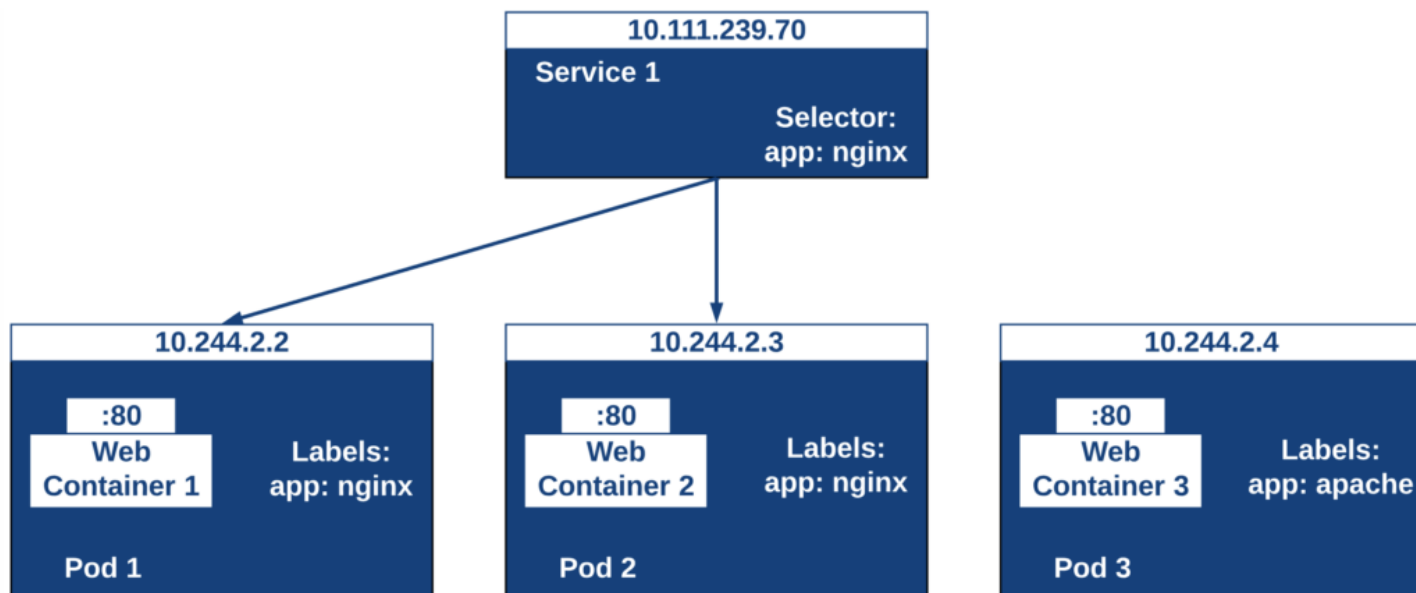
El balanceo de carga sirve (en el caso de una web) para aumentar las peticiones que puede llegar a recibir al mismo tiempo, ya que se distribuirán entre los múltiples pods en vez de uno solo, es decir un cliente hace la petición a una IP y se encarga a redirigir esa petición al pod indicado.

El **endpoint** de un servicio es el encargado de guardar la lista de direcciones IP de los pods, en el caso de que un pod muera y se arranque otro, borrará la IP del pod muerto y añadirá la del pod nuevo. Las IP's de los pods son dinámicas.

Agrupación de pods en servicios (labels)

Los pods pueden ser etiquetados con metadatos. Estos metadatos posteriormente pueden ser usados por otros objetos Kubernetes (p.e. ReplicaSet, Deployment) para seleccionar los pods y crear una unidad lógica (p.e. todas las réplicas de un contenedor de frontend)

La siguiente imagen muestra como un servicio agrupa mediante el selector `app:nginx` a aquellos pods que están etiquetados con `app:nginx`.



```
$ vim nginx.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: webcontainer
        image: nginx
        ports:
        - containerPort: 80
```

Al desplegar este deployment se crearán dos pods (replicas: 2), que quedarán agrupados por la coincidencia entre el selector que pide el deployment (app: nginx) y la etiqueta con los que son creados los pods (app: nginx).

```
$ kubectl apply -f nginx.yaml
deployment.apps/nginx created
```

Observamos ahora como los dos pods de nginx creados están agrupados lógicamente en el deployment nginx. Esta información está realmente en el objeto replicaSet creado por el deployment.

```
$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-59d9d8477-7wj7r              1/1     Running   0           101s
```



```
nginx-59d9d8477-rbl29 1/1 Running 0 101s
```

```
$ kubectl get deployments
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
nginx	2/2	2	2	3m1s

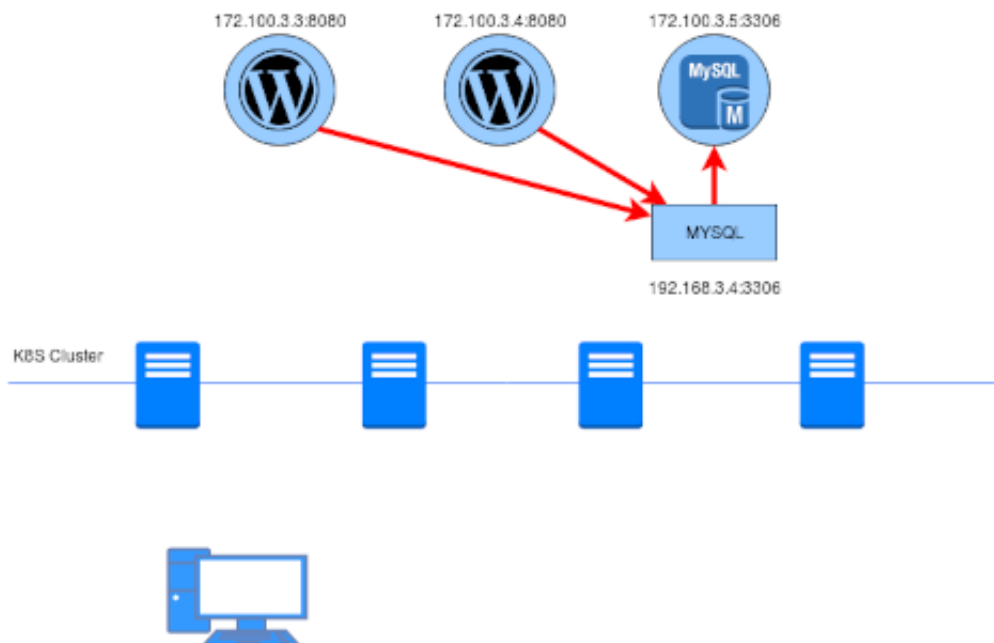
Cada pod tiene una dirección IP única, pero esa IP no se expone fuera del cluster sin lo que se denomina un servicio. Los servicios permiten que las aplicaciones reciban tráfico.

Tipos de servicios

En función del ámbito de la exposición del servicio tenemos cuatro tipos de servicios:

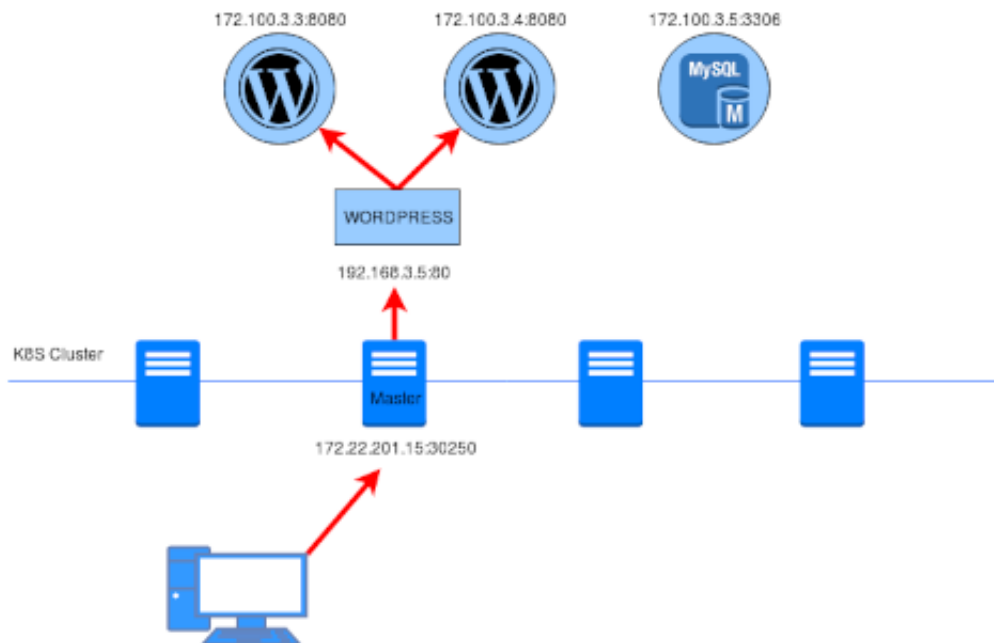
- **Cluster IP:** El servicio recibe una IP interna a nivel de clúster y hace que el servicio sólo sea accesible al mismo nivel.

Services - ClusterIP



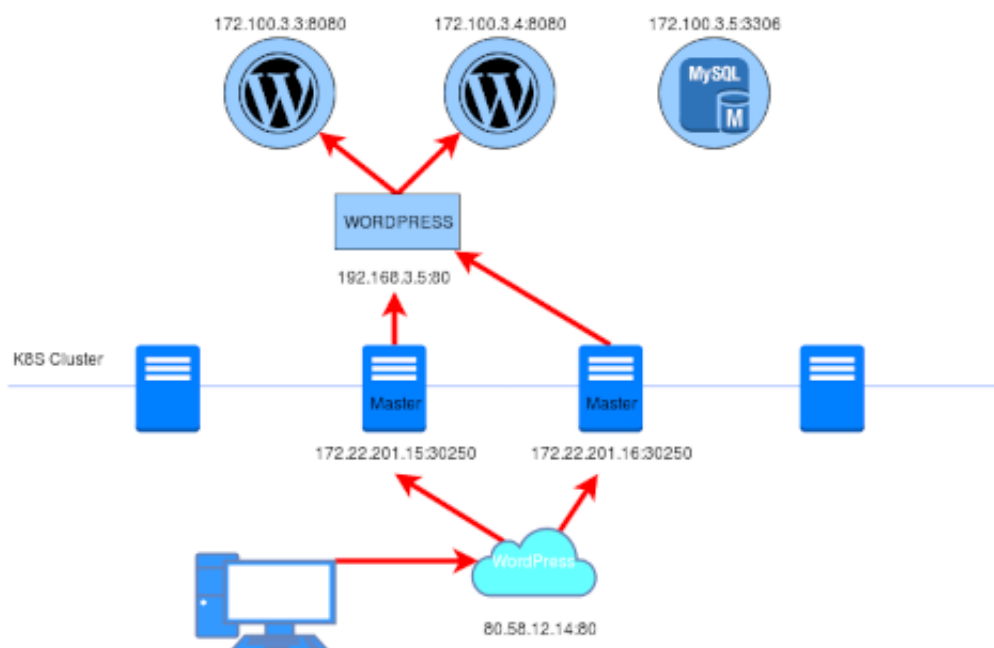
- **NodePort:** Expone el servicio fuera del clúster concatenando la IP del nodo en el que está el pod y un número de puerto entre 30000 y 32767, que es el mismo en todos los nodos

Services - NodePort



- **LoadBalancer:** Crea en cloud, si es posible, un balanceador externo con una IP externa asignada.

Services - LoadBalancer



- **ExternalName:** Expone el servicio usando un nombre arbitrario (especificado en externalName).

Desplegar servicios

Vamos a crear un archivo de servicio denominado `json-reader-service.yaml`. Este fichero, básicamente, contiene entre otros el nombre de servicio, el tipo del servicio (ClusterIP, NodePort, etc), el puerto de acceso a los pods del despliegue y el selector que identifica al despliegue con el que se corresponde el servicio creado.

```
$ vim json-reader-service.yaml
  apiVersion: v1
  kind: Service
  metadata:
    name: jsonreader
    namespace: default
  spec:
    type: NodePort
    ports:
      - name: http
        port: 80
        targetPort: http
    selector:
      app: jsonreader
```

Ejecutamos el fichero con kubectl para desplegar nuestro servicio.

```
$ kubectl create -f json-reader-service.yaml
service/jsonreader created
```

Mostramos la información del servicio que hemos desplegado.

```
$ kubectl get services
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
jsonreader    NodePort    10.108.222.193 <none>         80:31623/TCP    1
kubernetes    ClusterIP   10.96.0.1      <none>         443/TCP         2

$ kubectl get svc jsonreader
NAME          TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
jsonreader    NodePort    10.108.222.193 <none>         80:31623/TCP    3m

$ kubectl describe service jsonreader
Name:          jsonreader
Namespace:     default
Labels:        <none>
Annotations:    <none>
Selector:      app=jsonreader
Type:          NodePort
IP Families:    <none>
IP:            10.108.222.193
IPs:           <none>
Port:          http 80/TCP
TargetPort:    http/TCP
NodePort:      http 31623/TCP
Endpoints:     <none>
Session Affinity: None
External Traffic Policy: Cluster
Events:        <none>
```

Comprobamos como el servicio crea IPs asociadas al deployment

```
$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP
nginx-59d9d8477-7wj7               1/1     Running   0          27m   172.17.0.4
nginx-59d9d8477-rbl29              1/1     Running   0          27m   172.17.0.3
```

```
$ kubectl get endpoints
```

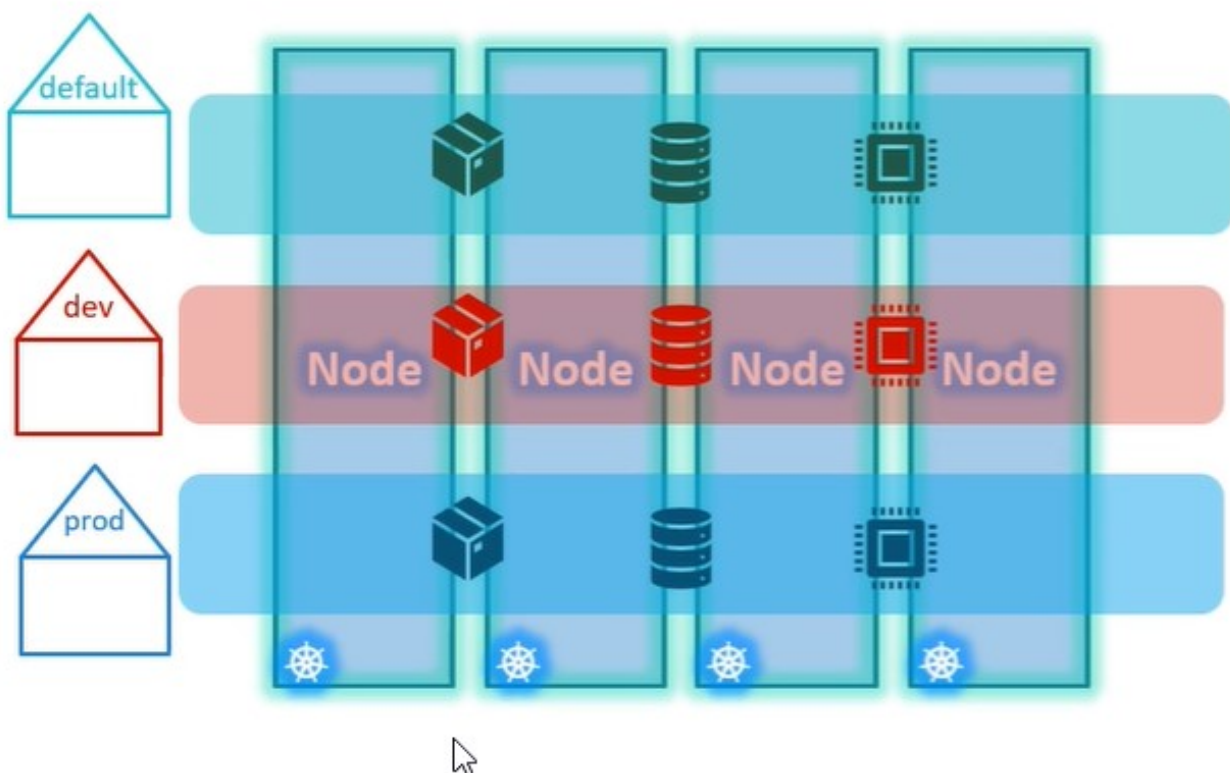
NAME	ENDPOINTS	AGE
jsonreader	<none>	3m
kubernetes	192.168.49.2:8443	22h

NAMESPACES

Namespaces son espacios de trabajo dentro de un clúster, donde cada namespace es independiente del otro.

En clústers con varios usuarios los namespaces proporcionan una forma de agrupar los recursos de cada usuario. Además, los administradores pueden establecer cuotas a nivel de namespace limitando a los usuarios la cantidad de objetos que pueden crear y la cantidad de recursos del cluster que pueden consumir (p.e. CPU, memoria).

No es necesario usar múltiples espacios de nombres sólo para separar recursos ligeramente diferentes, como versiones diferentes de la misma aplicación: para ello utiliza etiquetas para distinguir tus recursos dentro del mismo espacio de nombres.



Inicialmente tenemos 3 namespaces por defecto:

- **default:** Espacio de nombres por defecto.

- **kube-system:** Espacio de nombres creado y gestionado por Kubernetes.
- **kube-public:** Espacio de nombres accesible por todos los usuarios, reservado para uso interno del clúster.

Vamos a visualizarlo:

```
$ kubectl get namespaces
NAME              STATUS    AGE
default           Active    23h
kube-node-lease   Active    23h
kube-public       Active    23h
kube-system       Active    23h
```

Administración de namespaces

Crear namespace.

```
$ kubectl create namespace prod
namespace/prod created
```

Crear un pod indicando el namespace.

```
$ kubectl run nginx-prod --image=nginx --port 80 --namespace prod
pod/nginx-prod created
```

Inspeccionar namespace.

```
$ kubectl get namespaces
NAME              STATUS    AGE
default           Active    24h
kube-node-lease   Active    24h
kube-public       Active    24h
kube-system       Active    24h
prod              Active    119s
```

Mostrar los pods de un namespace.

```
$ kubectl get pods --namespace prod
NAME          READY   STATUS    RESTARTS   AGE
nginx-prod    1/1     Running   0           2m10s
```

Cambiar de namespace (dev).

Nota: Antes de cambiarlo, necesitamos crear el namespace.

```
$ kubectl create namespace dev
namespace/dev created
```

```
$ kubectl config set-context --current --namespace=dev
Context "minikube" modified.
```

Volver al namespace anterior (prod).

```
$ kubectl config set-context --current --namespace=prod
Context "minikube" modified.
```

Eliminar namespace.

```
$ kubectl delete namespace prod
namespace "prod" deleted
```

VOLUMES

El almacenamiento en contenedores es efímero. Una vez que el contenedor se elimina son eliminados también sus ficheros. Pero además, cuando un contenedor falla, kubelet lo reiniciará con un estado limpio habiéndose perdido todo lo que había en sus ficheros.

Kubernetes cuenta con una gran cantidad de tipos de volúmenes. Los hay de almacenamiento local, almacenamiento en el sistema de archivos de los nodos de Kubernetes, NFS y almacenamiento cloud, como en AWS, Azure, Google y OpenStack, etc.

También permite volúmenes configmap y secret, útiles para el compartir entre pods datos de configuración o información sensible, como contraseñas. En cualquier caso, los volúmenes son montados por los pods y accederán a sus datos.



Tipos de volumes

- **emptyDir:** es un directorio vacío que se crea como volumen y va ligado a la vida del pod. Mientras el pod siga corriendo, el volumen seguirá existiendo y manteniendo sus datos. Si el pod muere, el volumen desaparece
- **hostPath:** es un directorio que va ligado a la vida del nodo. Si un pod muere y se despliega en otro nodo no podrá acceder a la información del volumen donde murió. Este volumen existirá hasta que se elimine o el nodo muera.
- **Cloud Volumes:** son directorios de almacenamiento que proporcionan puntos de montaje en diferentes plataformas cloud (Azure, AWS, Google, OpenStack, ...).
- **Persistent Volumes (PV):** los recursos PersistentVolume se usan para administrar el almacenamiento duradero en un clúster. A diferencia de lo que sucede con los volúmenes, el ciclo de vida de PersistentVolume es administrado por Kubernetes. No necesitamos crear y borrar manualmente el almacenamiento de copia de seguridad. Los recursos PersistentVolume son recursos de clúster que existen de forma independiente de los pods. Esto significa que el disco y los datos representados por un PersistentVolume continúan existiendo a medida que el clúster cambia y los pods se borran y se vuelven a crear. Los recursos PersistentVolume se pueden aprovisionar de manera dinámica a través de PersistentVolumeClaims, o el administrador del clúster puede crearlos de manera estática.
- **Persistent Volumes Claims (PVC):** es una solicitud y una reclamación de un recurso PersistentVolume. Los objetos PVC solicitan un tamaño específico, un modo de acceso y una StorageClass para PersistentVolume. Si existe un PV que satisface la solicitud o se puede aprovisionar, PersistentVolumeClaim se vincula a ese PersistentVolume. Los pods usan reclamaciones como volúmenes. El clúster inspecciona la reclamación a fin de encontrar el volumen vinculado y lo activa para el pod.

Acceso y políticas de reciclaje

Tenemos tres modos de acceso, que depende del backend que vamos a utilizar:

- ReadWriteOnce: read-write solo para un nodo (RWO)
- ReadOnlyMany: read-only para muchos nodos (ROX)
- ReadWriteMany: read-write para muchos nodos (RWX)

Las políticas de reciclaje de volúmenes también depende del backend y son:

- Retain: Reclamación manual
- Recycle: Reutilizar contenido
- Delete: Borrar contenido

Creación volumes

- **emptyDir**

Vamos a crear un primer ejemplo con emptyDir. Como hemos indicado antes, este tipo de volúmenes dejan de existir una vez el pod muere. Con lo cual estos volúmenes se

utilizan para guardar información que si perdermos nos es indiferente. Un ejemplo sería utilizarlo para caché o memoria volátil. Lo creamos, ejecutamos el pod y observamos su contenido.

```
$ vim emptydir.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: test-pd
  spec:
    containers:
      - image: k8s.gcr.io/test-webserver
        name: test-container
        volumeMounts:
          - mountPath: /cache
            name: cache-volume
    volumes:
      - name: cache-volume
        emptyDir: {}

$ kubectl apply -f emptydir.yaml
pod/test-pd created

$ kubectl describe pod test-pd
Name:          test-pd
Namespace:     default
Priority:       0
Node:          minikube/192.168.39.88
Start Time:    Mon, 03 May 2021 12:20:38 +0200
Labels:        <none>
Annotations:   <none>
Status:        Running
IP:            172.17.0.5
IPs:
  IP: 172.17.0.5
Containers:
  test-container:
    Container ID:  docker://31dfdc4f8174b760f25760d37e7a76e6eb90b3bd
    Image:         k8s.gcr.io/test-webserver
    Image ID:      docker-pullable://k8s.gcr.io/test-webserver@sha256:
    Port:         <none>
    Host Port:    <none>
    State:        Running
      Started:    Mon, 03 May 2021 12:20:40 +0200
      Ready:      True
      Restart Count: 0
    Environment:  <none>
Mounts:
  /cache from cache-volume (rw)
  /var/run/secrets/kubernetes.io/serviceaccount from default-token-jc
Conditions:
  Type           Status
  Initialized     True
  Ready           True
  ContainersReady True
  PodScheduled    True
Volumes:
  cache-volume:
    Type: EmptyDir (a temporary directory that shares a pod's life
```



```

Medium:
SizeLimit: <unset>
default-token-jclqr:
  Type: Secret (a volume populated by a Secret)
  SecretName: default-token-jclqr
  Optional: false
QoS Class: BestEffort
Node-Selectors: <none>
Tolerations: node.kubernetes.io/not-ready:NoExecute op=Exists for 3
              node.kubernetes.io/unreachable:NoExecute op=Exists for 3

Events:
  Type    Reason      Age   From          Message
  ----    -
Normal    Scheduled   4m42s default-scheduler Successfully assigned def
Normal    Pulling     4m42s kubelet        Pulling image "k8s.gcr.io
Normal    Pulled      4m41s kubelet        Successfully pulled image
Normal    Created     4m41s kubelet        Created container test-co
Normal    Started     4m41s kubelet        Started container test-co

```

• hostPath

Este tipo de volúmenes montan un sistema de ficheros en el pod a nivel de nodo. Realizamos las mismas acciones que en la prueba anterior pero con la configuración de hostPath.

```

$ vim hostpath.yaml
  apiVersion: v1
  kind: Pod
  metadata:
    name: test-pd-hp
  spec:
    containers:
      - image: k8s.gcr.io/test-webserver
        name: test-container
        volumeMounts:
          - mountPath: /test-pd-hp
            name: test-volume
    volumes:
      - name: test-volume
        hostPath:
          # ruta directorio en el host
          path: /data
          # campo opcional
          type: Directory

$ kubectl apply -f hostpath.yaml
pod/test-pd-hp created

$ kubectl get pods
test-pd          1/1      Running    0          29m
test-pd-hp       1/1      Running    0          9m7s

$ kubectl describe pod
Name:          test-pd-hp
Namespace:     default
Priority:       0
Node:          minikube/192.168.39.88
Start Time:    Mon, 03 May 2021 12:40:41 +0200

```

```

Labels:      <none>
Annotations: <none>
Status:      Running
IP:          172.17.0.6
IPs:
  IP: 172.17.0.6
Containers:
  test-container:
    Container ID:  docker://afa1904d0536e219ee9db365af54d037f642d5e8c5
    Image:         k8s.gcr.io/test-webserver
    Image ID:      docker-pullable://k8s.gcr.io/test-webserver@sha256:
    Port:         <none>
    Host Port:    <none>
    State:        Running
      Started:    Mon, 03 May 2021 12:40:43 +0200
    Ready:        True
    Restart Count: 0
    Environment:  <none>
    Mounts:
      /test-pd-hp from test-volume (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  test-volume:
    Type:          HostPath (bare host directory volume)
    Path:          /data
    HostPathType:  Directory
  default-token-jclqr:
    Type:          Secret (a volume populated by a Secret)
    SecretName:    default-token-jclqr
    Optional:      false
QoS Class:         BestEffort
Node-Selectors:    <none>
Tolerations:       node.kubernetes.io/not-ready:NoExecute op=Exists for 3
Events:
  Type     Reason      Age   From          Message
  ----     -
  Normal   Scheduled   10m   default-scheduler   Successfully assigned def
  Normal   Pulling     10m   kubelet         Pulling image "k8s.gcr.io
  Normal   Pulled      10m   kubelet         Successfully pulled image
  Normal   Created     10m   kubelet         Created container test-co
  Normal   Started     10m   kubelet         Started container test-co

```

- **PersistentVolume y PersistentVolumeClaims (PV/PVC)**

En el siguiente ejemplo observamos que cada uno de los dos elementos desempeñan funciones distintas. PVC solicita espacio de almacenamiento y PV lo asigna si es posible. Comprobamos que es así.

```

$ vim pvpvc.yaml
# PV
apiVersion: v1
kind: PersistentVolume

```

```

metadata:
  name: pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
# PVC
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: pv-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 10Gi

$ kubectl apply -f pvpvc.yaml
persistentvolume/pv-volume created
persistentvolumeclaim/pv-claim created

$ kubectl get pv; kubectl get pvc
NAME          CAPACITY  ACCESS MODES  RECLAIM POLICY  STATUS  CLAIM
pv-volume     10Gi      RWO           Retain          Bound   default
NAME          STATUS    VOLUME          CAPACITY  ACCESS MODES  STORAGECLASS
pv-claim      Bound    pv-volume       10Gi      RWO           manual

```

StorageClass

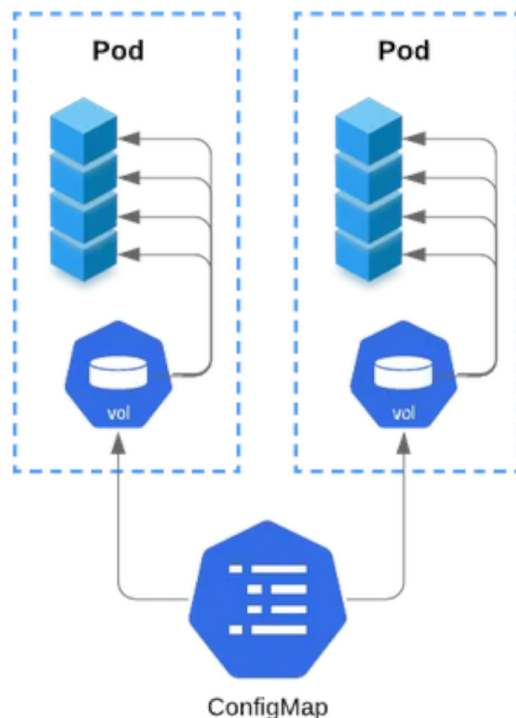
Es un objeto que proporciona a los administradores una forma de describir las clases de almacenamiento que ofrecen. Cada StorageClass contiene los campos provisioner, parameters y reclaimPolicy, que se usan cuando un PV que pertenece a la clase necesita almacenamiento dinámico. Mostramos un fichero yaml para ver sus opciones.

```

apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
  - debug
volumeBindingMode: Immediate

```

CONFIGMAPS / VARIABLES DE ENTORNO



Configuración app con variables de entorno

Para configurar las aplicaciones que vamos a desplegar usamos variables de entorno. Por ejemplo podemos ver las variables de entorno que podemos definir para configurar la imagen docker de MariaDB.

Definimos un deployment que despliegue un contenedor configurado por medio de variables de entorno y lo aplicamos.

```
$ vim mariadb-deployment.yaml
  apiVersion: apps/v1
  kind: Deployment
  metadata:
    name: mariadb-deployment
  labels:
    app: mariadb
    type: database
  spec:
    replicas: 1
    selector:
      matchLabels:
        app: mariadb
    template:
      metadata:
        labels:
          app: mariadb
          type: database
      spec:
        containers:
          - name: mariadb
```

```
image: mariadb
ports:
  - containerPort: 3306
    name: db-port
env:
  - name: MYSQL_ROOT_PASSWORD
    value: secret

$ kubectl apply -f mariadb-deployment.yaml
deployment.apps/mariadb-deployment created
```

También podemos configurarlo directamente pasando parámetros al cliente kubectl.

```
$ kubectl run mariadb --image=mariadb --env MYSQL_ROOT_PASSWORD=secret
```

Comprobamos como se ha desplegado el pod y accedemos a la app con la contraseña que hemos configurado.

```
$ kubectl get pods -l app=mariadb
NAME                                READY   STATUS    RESTARTS   AGE
mariadb-deployment-68768cb968-flljq 1/1     Running   0           5m54s

$ kubectl exec -it mariadb-deployment-68768cb968-flljq -- mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.5.9-MariaDB-1:10.5.9+maria~focal mariadb.org binary d

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input stat

MariaDB [(none)]>
```

Configuración app ConfigMaps

Los objetos ConfigMap permiten almacenar datos en forma de pares clave-valor (tuplas) para que puedan usarse posteriormente en despliegues. Son muy útiles para tener diferentes configuraciones de un mismo contenedor y utilizar la más adecuada según nuestras necesidades.

Hay que tener en cuenta que nuestros datos sensibles como las contraseñas estarán en texto plano. Para estos casos utilizaremos el objeto secrets que después explicaremos.

Vamos a crear y desplegar un configMap, a examinar su contenido y a comprobar su acceso.

```
$ kubectl create configmap mariadb --from-literal=root_password=my-password \
--from-literal=mysql_usuario=usuario \
--from-literal=mysql_password=password-user \
--from-literal=basededatos=test
configmap "mariadb" created

$ kubectl get configmaps
```

NAME	DATA	AGE
mariadb	4	2m14s

```
$ kubectl describe cm mariadb
```

```
Name:          mariadb
Namespace:      default
Labels:         <none>
Annotations:    <none>
```

Data

====

```
basededatos:
```

```
----
```

test

```
mysql_password:
```

```
----
```

password-user

```
mysql_usuario:
```

```
----
```

```
usuario
```

```
root_password:
```

```
----
```

```
my-password
```

```
Events:  <none>
```

```
$ vim mariadb-deployment-cm.yaml
```

```
apiVersion: apps/v1beta1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: mariadb-deploy-cm
```

```
  labels:
```

```
    app: mariadb
```

```
    type: database
```

```
spec:
```

```
  replicas: 1
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: mariadb
```

```
        type: database
```

```
    spec:
```

```
      containers:
```

```
        - name: mariadb
```

```
          image: mariadb
```

```
          ports:
```

```
            - containerPort: 3306
```

```
              name: db-port
```

```
          env:
```

```
            - name: MYSQL_ROOT_PASSWORD
```

```
              valueFrom:
```

```
                configMapKeyRef:
```

```
                  name: mariadb
```

```
                  key: root_password
```

```
            - name: MYSQL_USER
```

```
              valueFrom:
```

```
                configMapKeyRef:
```

```
                  name: mariadb
```

```
                  key: mysql_usuario
```

```
            - name: MYSQL_PASSWORD
```

```
              valueFrom:
```

```
                configMapKeyRef:
```

```

        name: mariadb
        key: mysql_password
-   name: MYSQL_DATABASE
    valueFrom:
        configMapKeyRef:
            name: mariadb
            key: basededatos

$ kubectl apply -f mariadb-deployment-cm.yaml
deployment.apps/mariadb-deployment configured

$ kubectl get pods -l app=mariadb
NAME                                READY   STATUS
mariadb-deployment-68768cb968-flljq 1/1     Running

$ kubectl exec -it mariadb-deploy-cm-57f7b9c7d7-ll6pv --mysql -u usuario -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 3
Server version: 10.5.9-MariaDB-1:10.5.9+maria~focal mariadb.org binary d

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input stat

MariaDB [(none)]>

```

Eliminamos el configMap

```

$ kubectl delete cm mariadb
configmap "mariadb" deleted

```

SECRETS

Los objetos Secret se usan para almacenar información sensible, como contraseñas, tokens, autenticación y claves ssh, etc. Almacenar esta información en objetos Secret es más seguro que colocarla en texto plano y legible como hace ConfigMap.

No obstante, los datos de los objetos Secret no están cifrados. Están codificados en base64, con lo cual hay que tener en cuenta que pueden hacerse visibles fácilmente.

Creamos el secret y lo examinamos.

```

$ kubectl create secret generic mariadb --from-literal=password=root
secret/mariadb created

$ kubectl get secret
NAME                                TYPE                                DATA   AGE
default-token-jclqr                kubernetes.io/service-account-token 3       5dlh
mariadb                             Opaque                                3       5dlh

$ kubectl describe secret mariadb

```

```
Name:      mariadb
Namespace:  default
Labels:     <none>
Annotations: <none>
```

Type: Opaque

Data

====

password: 4 bytes

Observamos ahora porque los secrets no son del todo seguros.

```
$ kubectl get secret mariadb -o yaml
apiVersion: v1
data:
  password: cm9vdA==
kind: Secret
metadata:
  creationTimestamp: "2021-05-03T17:56:35Z"
  name: mariadb
  namespace: default
  resourceVersion: "29390"
  selfLink: /api/v1/namespaces/default/secrets/mariadb
  uid: fb40d107-9780-4301-92fc-0d8310d0ee67
type: Opaque

$ echo 'cm9vdA==' | base64 --decode
root
```

Creamos el despliegue y probamos el acceso:

```
$ vim mariadb-deployment-secret.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mariadb-deploymentcm
  labels:
    app: mariadb
    type: database
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mariadb
  template:
    metadata:
      labels:
        app: mariadb
        type: database
    spec:
      containers:
        - name: mariadb
          image: mariadb
          ports:
            - containerPort: 3306
              name: db-port
```



```

env:
  - name: MYSQL_ROOT_PASSWORD
    valueFrom:
      secretKeyRef:
        name: mariadb
        key: password

$ kubectl create -f mariadb-deployment-secret.yaml
deployment.apps/mariadb-deployment-secret created

$ kubectl exec -it mariadb-deploy-secret-f946dddfd-kkmlb -- mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 8
Server version: 10.2.15-MariaDB-10.2.15+maria~jessie mariadb.org binary

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input stat

MariaDB [(none)]>

```

Eliminamos el secret.

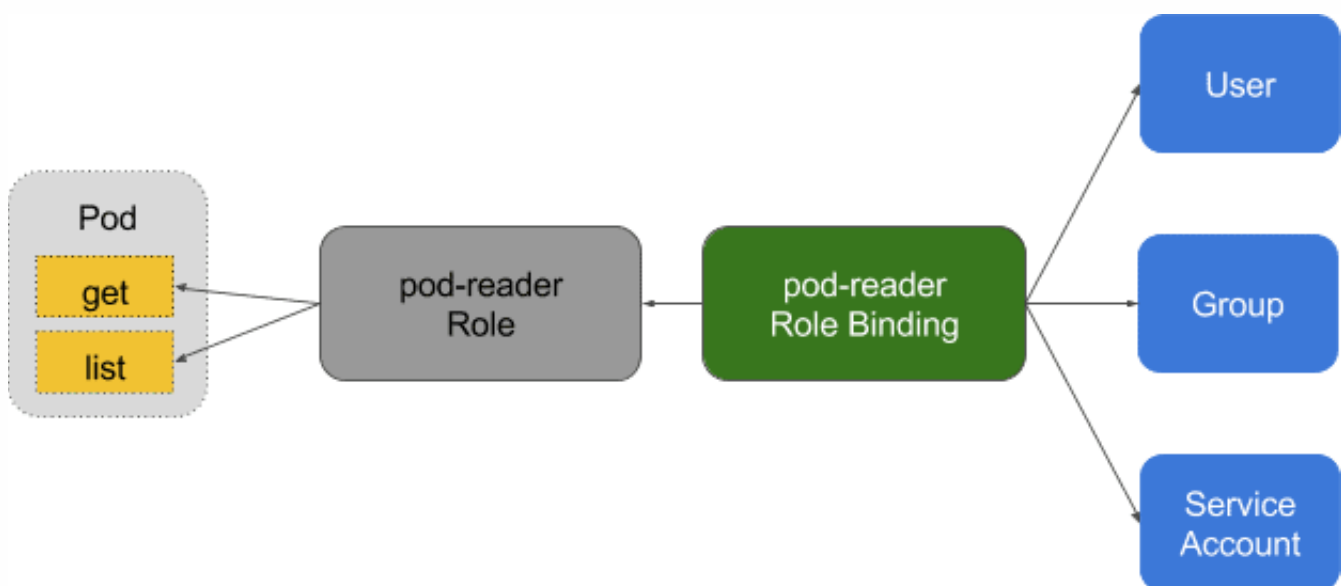
```

$ kubectl delete secret mariadb
secret "mariadb" deleted

```

RBAC

Role Based Access Control se utiliza para configurar las tareas que puede realizar cada usuario especificando diferentes roles en los que los clasificaremos. Por defecto el acceso está restringido por completo, con lo cual construimos las reglas a partir de permisos.



Usuarios

En Kubernetes no tiene una API para crear usuarios, sin embargo puede autenticar y autorizar

usuarios externos. Hay varios métodos para la autenticación (clients certs x509 , token files, passwords, ...). El método más utilizado es el certificado. Debemos tener instalado el cliente kubectl ya que es el modo de interactuar con el API-Server.

Vamos a crear un usuario siguiendo los pasos que describimos a continuación:

- Generamos el certificado de cliente.

```
$ openssl genrsa -out roberto_key.pem 2048  
$ openssl req -new -key roberto_key.pem -out roberto_csr.pem -subj "/CN
```

- Como administradores consultamos la petición y firmamos el certificado

```
$ kubectl config view  
apiVersion: v1  
clusters:  
- cluster:  
  certificate-authority: /home/users/inf/hisx2/isx43457566/.minikube  
  extensions:  
  - extension:  
    last-update: Tue, 04 May 2021 09:39:54 CEST  
    provider: minikube.sigs.k8s.io  
    version: v1.19.0  
    name: cluster_info  
  server: https://192.168.39.210:8443  
  name: minikube  
contexts:  
- context:  
  cluster: minikube  
  extensions:  
  - extension:  
    last-update: Tue, 04 May 2021 09:39:54 CEST  
    provider: minikube.sigs.k8s.io  
    version: v1.19.0  
    name: context_info  
  namespace: default  
  user: minikube  
  name: minikube  
current-context: minikube  
kind: Config  
preferences: {}  
users:  
- name: minikube  
  user:  
    client-certificate: /home/users/inf/hisx2/isx43457566/.minikube  
    client-key: /home/users/inf/hisx2/isx43457566/.minikube/profile  
  
$ openssl x509 -req -in roberto_csr.pem -CAkey ~/.minikube/ca.key -CA ~  
Signature ok  
subject=CN = roberto, O = adm  
Getting CA Private Key
```

- Como cliente, configuramos kubectl para acceder al clúster de minikube, añadimos credenciales y creamos y accedemos a nuestro espacio de trabajo. Comprobamos visualizando lo que hemos ejecutado.

```

$ kubectl config set-cluster minikube --certificate-authority=ca.crt
Cluster "minikube" set.

$ kubectl config set-credentials roberto --client-certificate=roberto_c
User "roberto" set.

$ kubectl config set-context roberto --cluster=minikube --user=roberto
Context "roberto" created.

$ kubectl config use-context roberto
Switched to context "roberto".

$ kubectl config view
apiVersion: v1
clusters:
- cluster:
    certificate-authority: /home/users/inf/hisx2/isx43457566/Kuberne
    extensions:
    - extension:
        last-update: Tue, 04 May 2021 09:39:54 CEST
        provider: minikube.sigs.k8s.io
        version: v1.19.0
        name: cluster_info
        server: https://192.168.39.210:8443
    name: minikube
contexts:
- context:
    cluster: minikube
    extensions:
    - extension:
        last-update: Tue, 04 May 2021 09:39:54 CEST
        provider: minikube.sigs.k8s.io
        version: v1.19.0
        name: context_info
    namespace: default
    user: minikube
    name: minikube
- context:
    cluster: minikube
    user: roberto
    name: roberto
current-context: roberto
kind: Config
preferences: {}
users:
- name: minikube
  user:
    client-certificate: /home/users/inf/hisx2/isx43457566/.minikube
    client-key: /home/users/inf/hisx2/isx43457566/.minikube/profile
- name: roberto
  user:
    client-certificate: /home/users/inf/hisx2/isx43457566/Kubernet
    client-key: /home/users/inf/hisx2/isx43457566/Kubernetes/robert

```

Permisos

Diferenciamos dos tipos de permisos según el ámbito en el que se aplican: Role y ClusterRole. El primero hace referencia a los permisos según namespace y el segundo al

clúster. Por defecto, no se permite hacer nada a los usuarios.

Role

Un rol en Kubernetes RBAC define lo que hará con un grupo de recursos. Contiene un grupo de reglas que definen un conjunto de permisos. Hay que tener en cuenta el namespace en el que se asignan los permisos, los resources a los que se podrá acceder y los verbs que se aplican sobre los objetos.

- **Resources:** pod, deployment, namespace, secret, configmap, service, persistentvolume...

```
$ kubectl api-resources
bindings                                v1
componentstatuses                      cs    v1
configmaps                            cm    v1
endpoints                             ep    v1
events                                ev    v1
limitranges                           limits v1
namespaces                            ns    v1
nodes                                  no    v1
persistentvolumeclaims                pvc   v1
persistentvolumes                     pv    v1
pods                                   po    v1
podtemplates                          v1
replicationcontrollers                 rc    v1
resourcequotas                        quota v1
secrets                               v1
serviceaccounts                       sa    v1
services                              svc    v1
mutatingwebhookconfigurations          admissionregistration.k8
validatingwebhookconfigurations        admissionregistration.k8
customresourcedefinitions              crd,crds apiextensions.k8s.io/v1
apiservices                           apiregistration.k8s.io/v
controllerrevisions                   apps/v1
daemonsets                             ds    apps/v1
deployments                           deploy apps/v1
replicasets                           rs    apps/v1
statefulsets                          sts    apps/v1
tokenreviews                           authentication.k8s.io/v1
localsubjectaccessreviews              authorization.k8s.io/v1
selfsubjectaccessreviews               authorization.k8s.io/v1
selfsubjectrulesreviews                authorization.k8s.io/v1
subjectaccessreviews                   authorization.k8s.io/v1
horizontalpodautoscalers                hpa    autoscaling/v1
cronjobs                               cj    batch/v1beta1
jobs                                   batch/v1
certificatesigningrequests              csr    certificates.k8s.io/v1
leases                                 coordination.k8s.io/v1
endpointslices                         discovery.k8s.io/v1beta1
events                                 events.k8s.io/v1
ingresses                              ing    extensions/v1beta1
flowschemas                           flowcontrol.apiserver.k8
prioritylevelconfigurations             flowcontrol.apiserver.k8
ingressclasses                         networking.k8s.io/v1
ingresses                              ing    networking.k8s.io/v1
networkpolicies                       netpol networking.k8s.io/v1
runtimeclasses                         node.k8s.io/v1
```

poddisruptionbudgets	pdb	policy/v1beta1
podsecuritypolicies	psp	policy/v1beta1
clusterrolebindings		rbac.authorization.k8s.io
clusterroles		rbac.authorization.k8s.io
rolebindings		rbac.authorization.k8s.io
roles		rbac.authorization.k8s.io
priorityclasses	pc	scheduling.k8s.io/v1
csidrivers		storage.k8s.io/v1
csinodes		storage.k8s.io/v1
storageclasses	sc	storage.k8s.io/v1
volumeattachments		storage.k8s.io/v1

- **Verbs:** get, list, watch, create, delete, update, edit, exec.

```
$ kubectl api-resources --no-headers --sort-by name -o wide | sed 's/.*\[//
create
delete
deletecollection
get
list
patch
update
watch
```

Vamos a crear un objeto Role en el archivo role.yaml. El rol definirá a qué recursos se podría acceder y qué operaciones se podrían usar. Para asignar este rol a un usuario o grupo se tiene que utilizar Rolebinding.

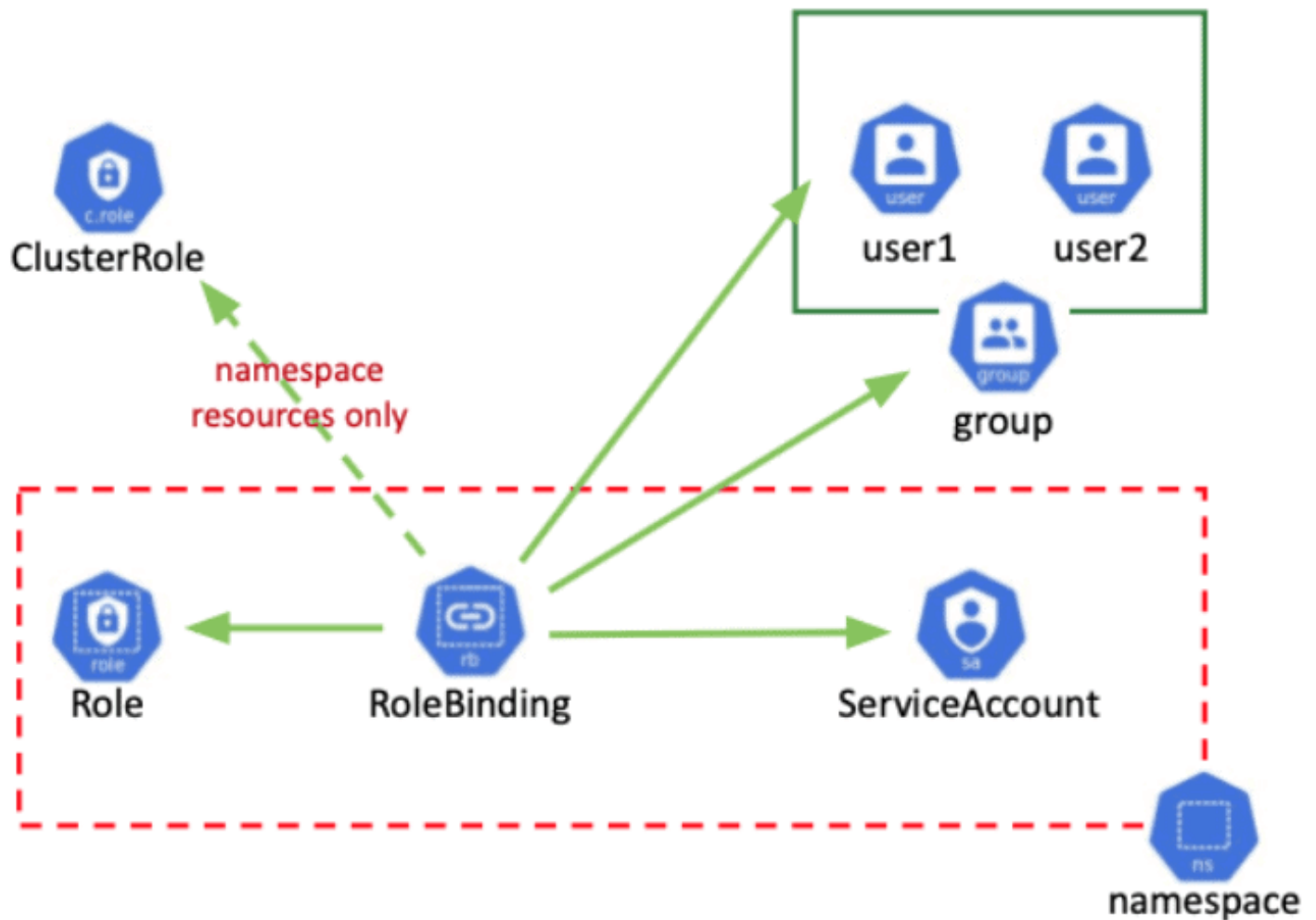
```
$ vim role.yaml
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: pod-reader
rules:
- apiGroups: ["" ] # "" indicates the core API group
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

$ kubectl apply -f role.yaml
role.rbac.authorization.k8s.io/pod-reader created

$ kubectl get roles
NAME          CREATED AT
pod-reader    2021-05-05T07:27:32Z

$ kubectl describe role pod-reader
Name:         pod-reader
Labels:       <none>
Annotations:  <none>
PolicyRule:
  Resources  Non-Resource URLs  Resource Names  Verbs
  -----  -
  "pods".""  []                 []              ["get" "watch" "list"]
```

RoleBinding



El control de acceso basado en roles se usa para otorgar permiso a un sujeto en un clúster. Los sujetos no son más que un grupo de usuarios, servicios o equipos que intentan acceder a la API de Kubernetes. Define qué operaciones puede realizar un usuario, servicio o grupo.

Vamos a enlazar nuestro usuario con el pod que hemos creado anteriormente.

```
$ vim rolebinding.yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    namespace: default
    name: pod-reader
  rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "watch", "list"]
# -----
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: users-read-pods
    namespace: default
  subjects:
  - kind: User
    name: roberto
    apiGroup: rbac.authorization.k8s.io
```

```

roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io

$ kubectl apply -f rolebinding.yaml
role.rbac.authorization.k8s.io/pod-reader configured
rolebinding.rbac.authorization.k8s.io/users-read-pods created

$ kubectl get rolebinding
NAME                ROLE                AGE
users-read-pods    Role/pod-reader    39s

$ kubectl describe rolebinding users-read-pod
Name:                users-read-pods
Labels:              <none>
Annotations:         <none>
Role:
Kind:                Role
Name:                pod-reader
Subjects:
  Kind  Name      Namespace
  ----  -
  User  roberto

```

ClusterRole

Tiene el mismo funcionamiento que Role con la diferencia que con este objeto aplicamos las reglas de permisos sobre el clúster, no sobre el namespace. En este caso vamos a asignar permisos de lectura de pods y deployments del clúster al usuario roberto.

```

$ vim clusterrole.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: cluster-pod-deploy-reader
rules:
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["get", "watch", "list"]
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]
#-----
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: users-read-pods-deploy
subjects:
- kind: User
  name: roberto
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: cluster-pod-deploy-reader
  apiGroup: rbac.authorization.k8s.io

$ kubectl apply -f clusterrole.yaml

```

```
clusterrole.rbac.authorization.k8s.io/cluster-pod-deploy-reader created
clusterrolebinding.rbac.authorization.k8s.io/users-read-pods-deploy crea

$ kubectl get clusterrole | grep cluster
cluster-admin 2
cluster-pod-deploy-reader 2
system:controller:clusterrole-aggregation-controller 2
```

ClusterRoleBinding

Se usa para otorgar permisos a un sujeto a nivel de clúster en todos los namespace En el siguiente ejemplo se crea un clusterRoleBinding de el grupo de usuarios dev asignando los permisos del clusterRole con nombre svc-clusterrole. Nuestro usuario está asignado a ese grupo.

```
$ vim clusterrolebinding.yaml
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  metadata:
    name: svc-clusterrole
  rules:
  - apiGroups: ["apps"]
    resources: ["services", "Pods", "replicasets", "deployments"]
    verbs: ["*"]
  #-----
  apiVersion: rbac.authorization.k8s.io/v1
  kind: ClusterRoleBinding
  metadata:
    name: cluster-svc
  subjects:
  - kind: Group
    name: dev
    apiGroup: rbac.authorization.k8s.io
  roleRef:
    kind: ClusterRole
    name: svc-clusterrole
    apiGroup: rbac.authorization.k8s.io

$ kubectl apply -f clusterrolebinding.yaml
clusterrole.rbac.authorization.k8s.io/svc-clusterrole created
clusterrolebinding.rbac.authorization.k8s.io/cluster-svc created
```

ServiceAccount

El serviceAccount es usado en Kubernetes para proporcionar una identidad a los pods. Todo Pod que quiera interactuar con el API Server deberá de autenticarse con un ServiceAccount. Por defecto, se usa el default.

Cada Pod hará uso de un ServiceAccount con un token y con unos roles establecidos para poder acceder al API Server.

Listamos y consultamos la información que contiene.

```
$ kubectl get serviceaccount
```


NAME	SECRETS	AGE
default	1	147m

```
$ kubectl describe sa default
Name:                default
Namespace:           default
Labels:              <none>
Annotations:         <none>
Image pull secrets:  <none>
Mountable secrets:   default-token-2w6w2
Tokens:              default-token-2w6w2
Events:              <none>
```

Observamos que contiene un secret. Vamos a examinarlo.

```
$ kubectl describe secret default-token-2w6w2
Name:                default-token-2w6w2
Namespace:           default
Labels:              <none>
Annotations:         kubernetes.io/service-account.name: default
                    kubernetes.io/service-account.uid: 20775cf0-fabf-49bb-8ff2

Type:                kubernetes.io/service-account-token

Data
====
token:               eyJhbGciOiJSUzI1NiIsImtpZCI6IkdGNHNHM0lrOThDbHdJY0d6Ymx1MHpx
                    ca.crt: 1111 bytes
namespace:           7 bytes
```

Vamos a hacer ejemplo para explicar su funcionamiento.

Primeramente creamos un serviceAccount, lo asignamos a un pod y creamos un rol al que le aplicaremos una serie de reglas.

```
$ vim sa-test.yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-sa
#-----
apiVersion: apps/v1
kind: Deployment
metadata:
  annotations:
  name: deployment-test
  labels:
    app: front-nginx
spec:
  replicas: 1
  selector:
    matchLabels:
      app: front-nginx
  template:
    metadata:
      labels:
        app: front-nginx
```

```

spec:
  # asignamos sa a pod
  serviceAccountName: my-sa
  containers:
  - name: nginx
    image: nginx:alpine

$ kubectl apply -f sa-test.yaml
serviceaccount/my-sa created
deployment.apps/deployment-test created

$ kubectl get sa
NAME                SECRETS    AGE
default             1          3h1m
my-sa               1          2m33s
$ kubectl get sa
NAME                SECRETS    AGE
default             1          3h1m
my-sa               1          2m33s

$ kubectl get roles
NAME                CREATED AT
pod-reader          2021-05-05T07:27:32Z

```

Asignamos el rol al serviceAccount y lo verificamos.

```

$ vim assign-role-sa.yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: sa-read-pods
  namespace: default
subjects:
- kind: ServiceAccount
  name: my-sa
  apiGroup: ""
roleRef:
  kind: Role
  name: sa-reader
  apiGroup: rbac.authorization.k8s.io

$ kubectl apply -f assign-role-sa.yaml
rolebinding.rbac.authorization.k8s.io/sa-read-pods created

$ kubectl get pod deployment-test-57bfcc4f79-jwtbd -o yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2021-05-05T10:06:33Z"
  generateName: deployment-test-57bfcc4f79-
  labels:
    app: front-nginx
    pod-template-hash: 57bfcc4f79
  name: deployment-test-57bfcc4f79-jwtbd
  namespace: default
  ownerReferences:
  - apiVersion: apps/v1
    blockOwnerDeletion: true
    controller: true
    kind: ReplicaSet

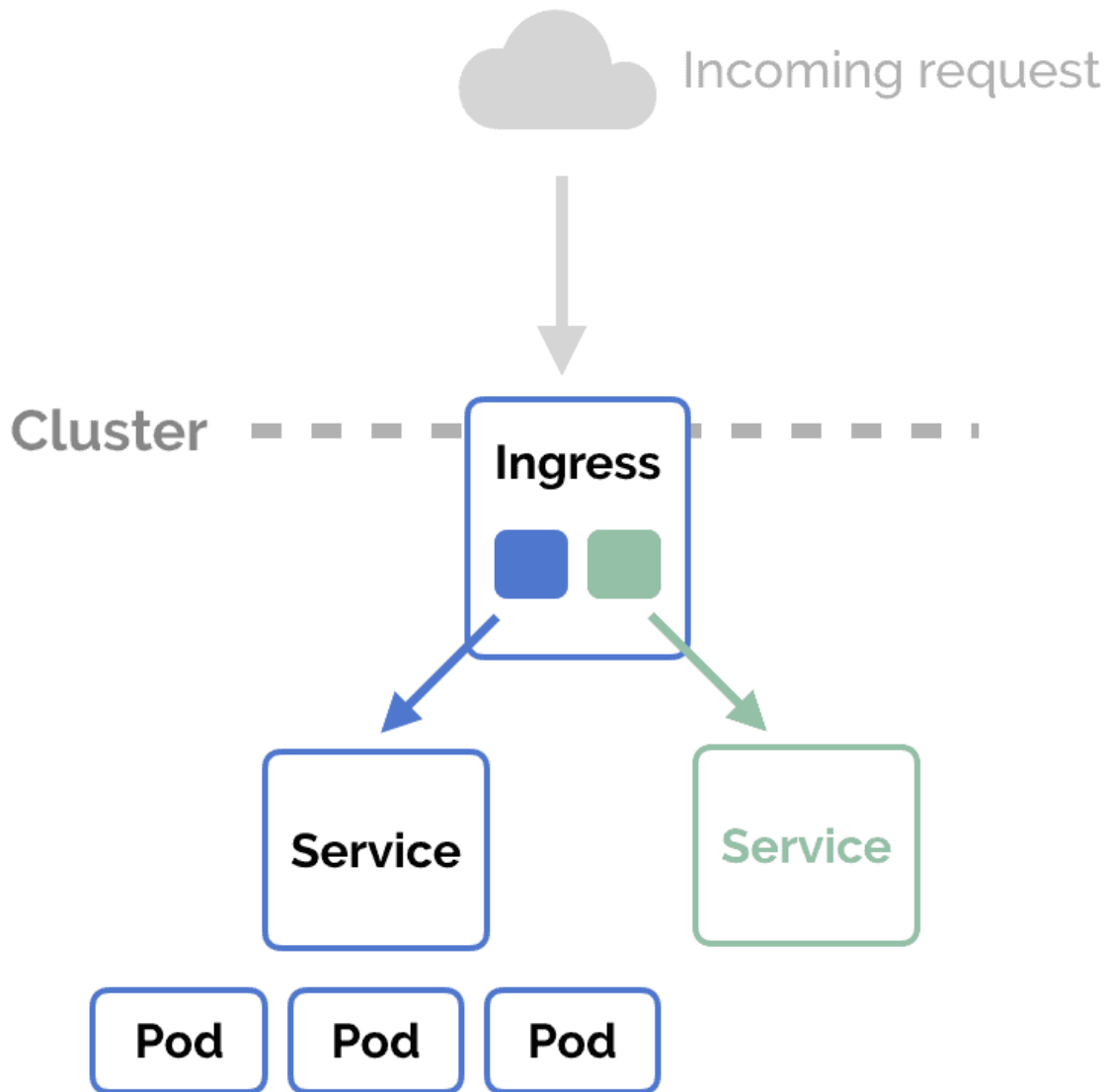
```

```
  name: deployment-test-57bfcc4f79
  uid: d75a3c4b-f761-46ee-bcd4-fdfb6cf9fcf7
  resourceVersion: "8125"
  uid: 0b203066-dd65-443f-aeld-bce2f8fb45f8
spec:
  containers:
  - image: nginx:alpine
    imagePullPolicy: IfNotPresent
    name: nginx
    resources: {}
    terminationMessagePath: /dev/termination-log
    terminationMessagePolicy: File
    volumeMounts:
    - mountPath: /var/run/secrets/kubernetes.io/serviceaccount
      name: my-sa-token-hncz5
      readOnly: true
  dnsPolicy: ClusterFirst
  enableServiceLinks: true
  nodeName: minikube
  preemptionPolicy: PreemptLowerPriority
  priority: 0
  restartPolicy: Always
  schedulerName: default-scheduler
  securityContext: {}
  serviceAccount: my-sa
  serviceAccountName: my-sa
  terminationGracePeriodSeconds: 30
  tolerations:
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  volumes:
  - name: my-sa-token-hncz5
    secret:
      defaultMode: 420
      secretName: my-sa-token-hncz5
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2021-05-05T10:06:33Z"
    status: "True"
    type: Initialized
  - lastProbeTime: null
    lastTransitionTime: "2021-05-05T10:06:40Z"
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2021-05-05T10:06:40Z"
    status: "True"
    type: ContainersReady
  - lastProbeTime: null
    lastTransitionTime: "2021-05-05T10:06:33Z"
    status: "True"
    type: PodScheduled
  containerStatuses:
  - containerID: docker://b6da0235b8c8c19b748b8610aacb81e05713adb933d561
```

```
image: nginx:alpine
imageID: docker-pullable://nginx@sha256:07ab71a2c8e4ecb19a5a5abcfb3a
lastState: {}
name: nginx
ready: true
restartCount: 0
started: true
state:
  running:
    startedAt: "2021-05-05T10:06:39Z"
hostIP: 192.168.39.173
phase: Running
podIP: 172.17.0.7
podIPs:
- ip: 172.17.0.7
qosClass: BestEffort
startTime: "2021-05-05T10:06:33Z"
```

INGRESS

Otra de las formas de acceder a los pods desde el exterior del clúster es mediante lo que se conoce como Ingress. Este recurso nos permite acceder a servicios a través del protocolo web http/https. El tráfico se controla utilizando un conjunto de reglas que tú defines. Además de dar a tus aplicaciones una URL externa que permita el acceso, también se puede configurar para el balanceo de carga, es decir que realiza la función de proxy.



Ingress Controller

Es un contenedor que enruta las peticiones hacia los servicios correspondientes en base a la definición del recurso ingress. Hay una gran cantidad de controladores (Traefik, HAproxy, Envoy,...) pero nosotros vamos a utilizar el NGINX controller para exponer las apps.

Instalación y configuración Ingress

Vamos a realizar un ejemplo desplegando la aplicación hello, world a cual le aplicaremos un ingress para exponerla y comprobaremos el resultado.

Primeramente habilitamos el controlador NGINX Ingress y lo verificamos.

```
$ minikube addons enable ingress
  ■ Using image k8s.gcr.io/ingress-nginx/controller:v0.44.0
  ■ Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
  ■ Using image docker.io/jettech/kube-webhook-certgen:v1.5.1
  🌐 Verifying ingress addon...
  🌟 The 'ingress' addon is enabled
```

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

NAME	READY	STATUS	RESTARTS	AGE
coredns-74ff55c5b-m6ffm	1/1	Running	0	23h
etcd-minikube	1/1	Running	0	23h
kube-apiserver-minikube	1/1	Running	0	23h
kube-controller-manager-minikube	1/1	Running	0	23h
kube-proxy-9s4px	1/1	Running	0	23h
kube-scheduler-minikube	1/1	Running	0	23h
storage-provisioner	1/1	Running	0	23h

Una vez comprobado que el controlador está habilitado desplegamos y exponemos la app por el puerto 8080 mediante node-port.

```
$ kubectl create deployment web --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/web created

$ kubectl expose deployment web --type=NodePort --port=8080
service/web exposed
```

Verificamos que el servicio se haya creado en node-port y lo visitamos.

```
$ kubectl get service web
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
web	NodePort	10.110.106.21	<none>	8080:31148/TCP	102s

```
$ minikube service web --url
http://192.168.49.2:31148

$ curl http://192.168.49.2:31148
Hello, world!
Version: 1.0.0
Hostname: web-79d88c97d6-l26fh
```

Hay dos modos de establecer las reglas de ingress: por **ruta** o **dominio**.

Vamos a implementar ahora un ejemplo de dos despliegues de una app (hello, world).

Desplegamos y verificamos el primer ejemplo.

```
$ vim example-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: hello-world.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
```

```

        service:
        name: web
        port:
        number: 8080

$ kubectl apply -f example-ingress.yaml
ingress.networking.k8s.io/example-ingress created

$ kubectl get ingress

```

NAME	CLASS	HOSTS	ADDRESS	PORTS	AGE
example-ingress	<none>	hello-world.info	192.168.49.2	80	523

Editamos el nombre de dominio en el fichero de configuración de DNS **/etc/hosts** y verificamos que podemos acceder a la app.

```

$ vim /etc/hosts
192.168.49.2    hello-world.info

$ curl hello-world.info
Hello, world!
Version: 1.0.0
Hostname: web-79d88c97d6-l26fh

```

Creamos y exponemos el segundo deployment y aplicamos cambio en las reglas.

```

$ kubectl create deployment web2 --image=gcr.io/google-samples/hello-app:2.0
deployment.apps/web2 created

$ kubectl expose deployment web2 --port=8080 --type=NodePort
service/web2 exposed

$ vim example-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
annotations:
  nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - host: hello-world.info
    http:
      paths:
      - path: /v2
        pathType: Prefix
        backend:
          service:
            name: web2
            port:
              number: 8080

$ kubectl apply -f
ingress.networking.k8s.io/example-ingress configured

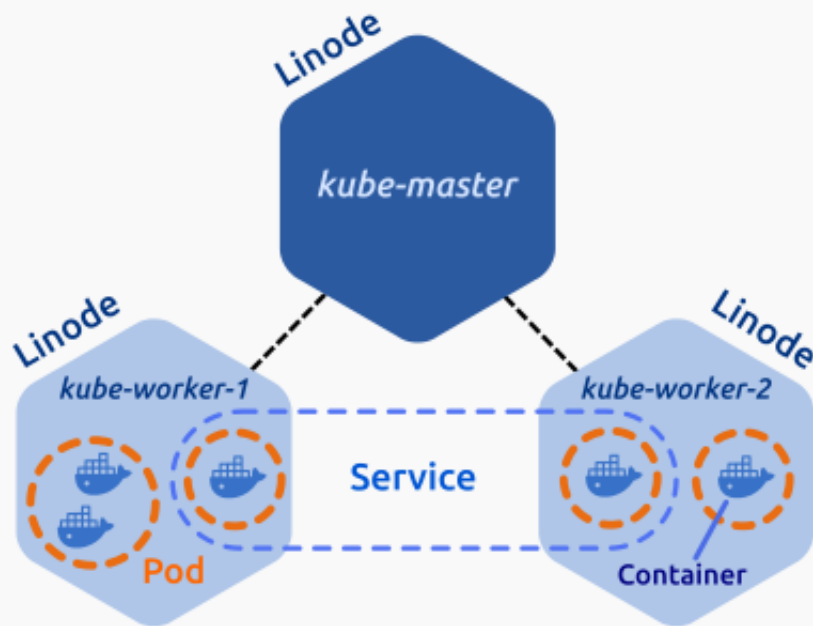
```

Ahora comprobamos como podemos acceder tanto ala primera como a la segunda versión de la app. Al primero accedemos por dominio y al segundo por ruta y dominio a la vez.

```
$ curl hello-world.info
Hello, world!
Version: 1.0.0
Hostname: web-79d88c97d6-l26fh

$ curl hello-world.info/v2
Hello, world!
Version: 2.0.0
Hostname: web2-89cd47949f-t8rst
```

CREACIÓN CLUSTER LOCAL



Instalación

En este ejemplo de instalación se muestra como instalar un cluster de tres máquinas, un master (control-plane) y dos workers, esta instalación se realiza en máquinas virtuales Fedora 32 y la tipología del master es stacked.

Preparación de nodos

Asignar hostname a cada nodo

```
$ hostnamectl set-hostname master
$ hostnamectl set-hostname node1
$ hostnamectl set-hostname node2
```

Deshabilitar swap


```
$ swapoff -a

$ sed -i '/ swap / s/^/#/' /etc/fstab
```

Resolución de nombres

```
/etc/hosts

127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
192.168.122.2 master
192.168.122.3 node1
192.168.122.4 node2
```

Ip fijas (para cada nodo)

```
[root@master ~]# cat /etc/sysconfig/network-scripts/ifcfg-enp1s0
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp1s0
UUID=933cd398-b601-4406-ba4c-6a5bf8106bdc
ONBOOT=yes
AUTOCONNECT_PRIORITY=-999

BOOTPROTO=none
DEVICE=enp1s0
IPADDR=192.168.122.2
NETMASK=255.255.255.0
GATEWAY=192.168.122.1
DNS1=192.168.122.1
DN2=8.8.8.8
```

Deshabilitar firewalld y habilitar forwarding

```
$ systemctl stop firewalld

$ systemctl disable firewalld

# Enable IP Forwarding
$ echo '1' > /proc/sys/net/bridge/bridge-nf-call-iptables

$ cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

```
EOF
```

```
# Si nos da este problema
cannot stat /proc/sys/net/bridge/bridge-nf-call-iptables: No existe el fichero o

# SOLUCIÓN
[root@node1 ~]# modprobe br_netfilter

[root@node1 ~]# sysctl -p
    net.bridge.bridge-nf-call-ip6tables = 1
    net.bridge.bridge-nf-call-iptables = 1
```

Instalar servicios

En todos los nodos es necesario instalar docker, kubelet, kubect, kubeadm.

```
$ sudo dnf install -y kubelet kubect kubeadm
```

Instalación de docker

<https://docs.docker.com/engine/install/fedora/>

Instalación de kubeam, kubect, kubelet

```
# Añadimos repositorio de kubernetes

cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-`$basearc
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg https://package
exclude=kubelet kubeadm kubect
EOF

# Set SELinux in permissive mode (effectively disabling it)

$ setenforce 0

$ sed -i 's/^SELINUX=enforcing$/SELINUX=permissive/' /etc/selinux/config

# instalamos servicios y habilitamos kubelet (Fedora 32)
$ dnf install -y kubelet kubeadm kubect --disableexcludes=kubernetes

$ systemctl enable --now kubelet

[root@master ~]# $ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"21", GitVersion:"v1.21.0",
```

Creación de master

En la creación del master hay que tener en qué red se crearan los Pods, esta opción kubernetes lo deja en addons externos, en nuestro caso utilizaremos el addon calico.

```
Nota importante: El dns del cluster CoreDNS no se iniciará si no hay antes una r
Iniciar configuración del master con red de pod
$ kubeadm init --pod-network-cidr=192.168.0.0/16
```

Descargará imagenes y tardará un poco.(dependiendo la conexión)

Una vez finalizado el comando kubeadm init --pod-network-cidr=192.168.0.0/16 nos saldrá lo siguiente:

```
Your Kubernetes control-plane has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

$ mkdir -p $HOME/.kube

$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

$ sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

Then you can join any number of worker nodes by running the following on each as

$ kubeadm join 192.168.122.2:6443 --token cp5ayc.pbsbruka2leselbe \
  --discovery-token-ca-cert-hash sha256:d41acb35ced40eae84f731f2892a8c131e55a63
```

Es muy importante guardarse bien la línea de kubeadm join ya que con esta juntaremos los nodos al master.

En caso de no querer gestionar el cluster como root y quererlo gestionar como usuario.

```
[master@master ~]$ mkdir -p $HOME/.kube

[master@master ~]$ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config

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

Addon para gestionar las redes de Pods

```
$ kubectl apply -f https://docs.projectcalico.org/v3.11/manifests/calico.yaml
```

Enlazar nodos worker

Para juntar un worker al nodo master simplemente hay que ejecutar kubeadm join con el token del master.

```
$ kubeadm join 192.168.122.2:6443 --token c2bn4c.qzpskak8ryp1uotd \
--discovery-token-ca-cert-hash sha256:485a9c649a4d2a4ad9ec03932f6353fc559a1a
```

En caso de no tener el token, siempre se puede crear uno nuevo desde el master con el siguiente comando:

```
$ kubeadm token create --print-join-command
```

Comprobamos desde el master que hemos añadido el nodo y asignaremos el nodo como worker para identificarlo correctamente.

```
[root@master ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	control-plane,master	68m	v1.21.0
node1	Ready	<none>	66s	v1.21.0

Añadir nodo como worker

```
[root@master ~]# kubectl label node node1 node-role.kubernetes.io/worker=worker
node/node1 labeled
```

```
[root@master ~]# kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
master	Ready	control-plane,master	71m	v1.21.0
node1	Ready	worker	3m59s	v1.21.0

```
[root@master ~]# kubectl label nodes node1 node=worker1
node/node1 labeled
```

```
[root@master ~]# kubectl get nodes --show-labels
```

NAME	STATUS	ROLES	AGE	VERSION	LABELS
master	Ready	control-plane,master	74m	v1.21.0	beta.kubernetes.i
node1	Ready	worker	6m26s	v1.21.0	beta.kubernetes.i

Eliminar un nodo

Desde el nodo master quitamos todas las tareas del nodo y después lo eliminamos.

```
$ kubectl drain node2 --delete-local-data --force --ignore-daemonsets
```

```
$ kubectl delete node node2
```

```
[root@master ~]# kubectl drain node2 --delete-local-data --force --ignore-daemon
Flag --delete-local-data has been deprecated, This option is deprecated and
```

```
node/node2 cordoned
WARNING: ignoring DaemonSet-managed Pods: kube-system/calico-node-4nmzb, kub
evicting pod kube-system/coredns-558bd4d5db-rt6lz
evicting pod kube-system/calico-kube-controllers-5bcd7db644-2wsbw
evicting pod kube-system/coredns-558bd4d5db-k7m56
pod/calico-kube-controllers-5bcd7db644-2wsbw evicted
pod/coredns-558bd4d5db-rt6lz evicted
pod/coredns-558bd4d5db-k7m56 evicted
node/node2 evicted
```

```
[root@master ~]# kubectl delete node node2
node "node2" deleted
```

En el nodo que se a eliminado del cluster restablecer la configuración inicial \$ kubeadm reset

El proceso de reinicio no reinicia ni limpia las reglas de iptables o las tablas de IPVS. Si desea restablecer iptables, debe hacerlo manualmente:

```
$ iptables -F && iptables -t nat -F && iptables -t mangle -F && iptables -X
```

Si desea restablecer las tablas IPVS, debe ejecutar el siguiente comando:

```
$ ipvsadm -C
```

PROVEEDORES DE SERVICIO

Hemos visto cómo implementar y autogestionar nuestro propio servicio de Kubernetes, pero existen también diferentes plataformas que nos lo proporcionan.

Podemos implementar el clúster de Kubernetes en servicios administrados por proveedores, como Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), entre otros muchos. Al hacerlo no tenemos que administrar el control-plane del clúster, solo debemos preocuparnos por cómo implementar la aplicación y seleccionar el tipo de instancia.

Aparte de eso, los proveedores manejan todas las demás inquietudes (soporte, entornos preconfigurados, alojamiento, ...)



Google Kubernetes Engine (GKE)

Kubernetes fue creado por Google para su propia herramienta de orquestación de contenedores inicialmente llamada BORG, luego cambiaron su nombre a Omega; esta larga historia es la razón por la que se considera el servicio de Kubernetes administrado más avanzado.

También incluye una verificación de estado y reparación automática de microservicios, registro y monitoreo. Además, viene con escalado automático de cuatro vías y soporte para múltiples clústeres.

Algunas de las características principales del servicio Google Kubernetes son:

- Se inicia rápidamente con clústeres de un solo clic
- Aprovecha un plano de control de alta disponibilidad que incluye clústeres regionales y multizona.
- Elimina la sobrecarga operativa con canales de liberación, actualización automática y reparación automática.
- Es seguro de forma predeterminada, incluido el escaneo de vulnerabilidades de las imágenes del contenedor y el cifrado de datos.
- Monitoreo integrado en la nube con vistas específicas de infraestructura, aplicaciones y Kubernetes.
- Función de escalado automático de cuatro vías: Autoescalado horizontal de pods basado en el uso de CPU o métricas personalizadas, autoescalado de clúster que funciona por grupo de nodos y autoescalado vertical de pods que escanea automáticamente el uso de CPU y memoria de los pods, y ajusta dinámicamente su CPU y aplicaciones de memoria en respuesta. Escala automáticamente el grupo de nodos y los clústeres en varios grupos de nodos, según los requisitos cambiantes de la

carga de trabajo.

Costo: GKE cobra 0.10 \$ por hora por la administración del clúster de Kubernetes y cobra por el resto de servicios según una escala de precios.

Amazon Elastic Kubernetes Service (EKS)

El servicio web de Amazon tiene su propio servicio de Kubernetes administrado llamado EKS. También es otro servicio de Kubernetes administrado en el que no es necesario mantener o crear el control plane del clúster.

EKS ejecuta un plano de control de clúster en varias zonas de disponibilidad para garantizar que mantiene una alta disponibilidad y reemplaza automáticamente las instancias en mal estado. Funciona con diferentes AWS para proporcionar escalabilidad y seguridad para su aplicación, como las siguientes:

- Amazon ECR para imágenes de contenedor.
- Equilibrio de carga elástico para distribución de carga.
- IAM para autenticación.
- Amazon VPC para aislamiento.

EKS cobra 0.10 \$ por hora por la administración del clúster de Kubernetes y cobra por los servicios subyacentes de acuerdo con una escala de precios.

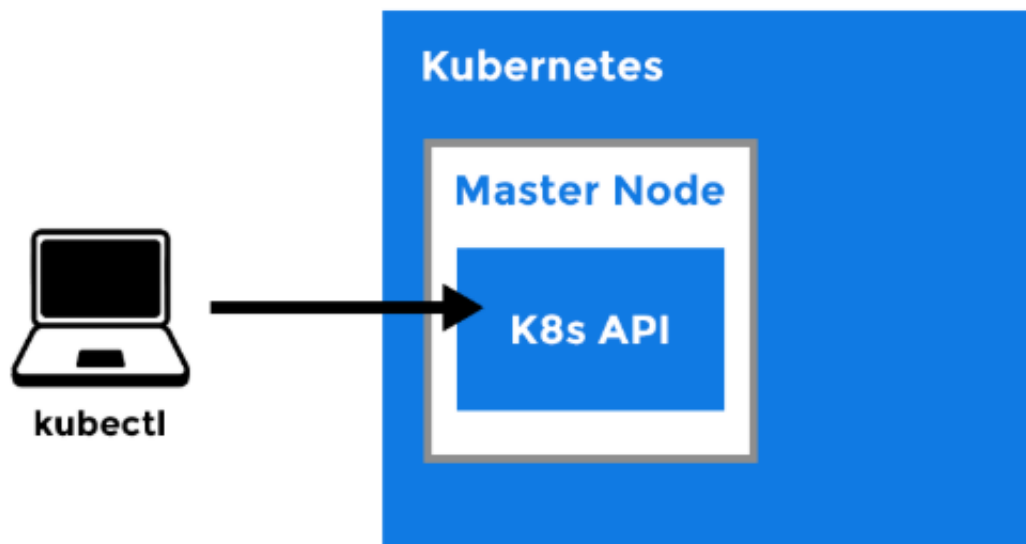
Azure Kubernetes Service (AKS)

AKS también es un servicio administrado por Kubernetes, que reduce la complejidad y la sobrecarga operativa de administrar Kubernetes al descargar gran parte de esa responsabilidad a Azure. Maneja todas sus tareas críticas, monitoreo de salud y mantenimiento. Ofrece Kubernetes sin servidor, una experiencia integrada de integración continua y entrega continua (CI / CD), y seguridad y gobernanza de nivel empresarial.

Costo: AKS no cobra nada por la administración del clúster de Kubernetes. Solo cobra por sus servicios subyacentes.

ANEXO

Comandos básicos kubectl



Visualizar información

- **Nodos:**

```
$ kubectl get no
$ kubectl get no -o wide
$ kubectl describe no
$ kubectl get no -o yaml
$ kubectl get nodes -o [jsonpath]
$ kubectl top node [node-name]
```

- **Pods**

```
$ kubectl get po
$ kubectl get pod -o wide
$ kubectl describe pods
$ kubectl get po --show-labels
$ kubectl get po -l app=nginx
$ kubectl get po -o yaml
$ kubectl get pod [pod_name] -o yaml --export
$ kubectl get pod [pod_name] -o yaml --export > file.yaml
```

- **Namespaces:**


```
$ kubectl get namespaces  
  
$ kubectl get ns -o yaml  
  
$ kubectl describe ns
```

- **Deployments:**

```
$ kubectl get deployments  
  
$ kubectl describe deploy  
  
$ kubectl get deploy -o wide  
  
$ kubectl get deploy -o yaml
```

- **Services:**

```
$ kubectl get services  
  
$ kubectl describe svc  
  
$ kubectl get svc -o wide  
  
$ kubectl get svc -o yaml  
  
$ kubectl get svc --show-labels
```

- **ServiceAccounts:**

```
$ kubectl get serviceaccounts  
  
$ kubectl get sa -o yaml  
  
$ kubectl get serviceaccounts default -o yaml > sa.yaml  
  
$ kubectl replace serviceaccount default -f sa.yaml
```

- **ReplicaSets:**

```
$ kubectl get replicaset  
  
$ kubectl describe rs  
  
$ kubectl get rs -o wide  
  
$ kubectl get rs -o yaml
```

- **Secrets:**

```
$ kubectl get secrets
```

```
$ kubectl get secrets --all-namespaces  
$ kubectl get secrets -o yaml
```

- **ConfigMaps:**

```
$ kubectl get configmaps  
$ kubectl get cm --all-namespaces  
$ kubectl get cm --all-namespaces -o yaml
```

- **Ingress:**

```
$ kubectl get ingress  
$ kubectl get ing --all-namespaces
```

- **PV/PVC:**

```
$ kubectl get pv|pvc  
$ kubectl describe pv|pvc
```

- **Logs:**

```
$ kubectl logs [pod_name]  
$ kubectl logs --since=1h [pod_name]  
$ kubectl logs --tail=20 [pod_name]  
$ kubectl logs -f -c [container_name] [pod_name]  
$ kubectl logs [pod_name] > pod.log
```

Modificar/Eliminar atributos de los recursos

- **Nodos:**

```
$ kubectl edit node [node-name]  
$ kubectl delete node [node-name]
```

- **Pods:**

```
$ kubectl edit pod [pod-name]
```

```
$ kubectl delete pod [pod-name]
```

- **Deployments:**

```
$ kubectl edit deploy [deploy-name]

$ kubectl delete deploy [deploy_name]

$ kubectl expose deploy [deploy_name] --port=80 --type=NodePort

$ kubectl scale deploy [deploy_name] --replicas=5
```

- **Namespaces:**

```
$ kubectl delete ns

$ kubectl edit ns [ns_name]
```

- **ServiceAccounts:**

```
$ kubectl edit sa [sa_name]

$ kubectl delete sa [sa_name]
```

Crear recursos

- **Crear Pod:**

```
$ kubectl create -f [file]

$ kubectl apply -f [file]

$ kubectl run [pod_name] --image=nginx --restart=Never
```

- **Crear/Exponer Service:**

```
$ kubectl create svc nodeport [svc_name] --tcp=8080:80

$ kubectl expose -f file.yaml --port=80 --target-port=8080
```

- **Crear deployment:**

```
$ kubectl create -f [deploymentfile]

$ kubectl apply -f [deploymentfile]

$ kubectl create deploy [deploy_name] --image nginx
```

- **Crear pod interactivo:**

```
$ kubectl run [pod_name] --image=busybox --rm -it --restart=Never -- /bin/b
```

- **Crear fichero mediante output yaml:**

```
$ kubectl create deploy [deploy_name] --image=nginx --dry-run -o yaml > de  
$ kubectl get po [pod_name] -o yaml --export > pod.yaml
```

Info cluster

```
$ kubectl config  
$ kubectl cluster -info  
$ kubectl get componentstatuses
```

Help

```
$ kubectl -h  
$ kubectl create - h$  
$ kubectl run - h  
$ kubectl explain deploy.spec
```

HELM



Helm (del término marítimo de timón) es una herramienta para gestionar aplicaciones de Kubernetes. Helm te ayuda a "timonear" Kubernetes usando cartas de navegación que se conocen como Helm Charts.

La principal función de Helm es definir, instalar y actualizar aplicaciones complejas de Kubernetes. Helm es mantenido por la CNCF en colaboración con Microsoft, Google, Bitnami y la comunidad de Helm.

Helm charts

Las cartas de navegación permiten crear, versionar y publicar una aplicación Kubernetes. Cuando usamos Helm Charts tenemos un asistente de optimización que facilita la administración e instalación de las apps en Kubernetes y el proceso de empaquetamiento.

Helm Charts se divide en dos: una parte es Helm, como cliente y la otra es Tiller como servidor.

Tiller (server)

Es el componente que se encarga de la gestión de los Charts, específicamente en sus instalaciones. Es decir el servidor Interactúa directamente con el API de Kubernetes para instalar, actualizar, consultar y eliminar recursos de Kubernetes. También almacena los objetos de cada release o distribución.

Helm (client)

Helm se ejecuta directamente en un equipo o en el dispositivo elegido para la ejecución. Los paquetes Helm están compuestos de una descripción del paquete y de archivos contenedores de manifiestos Kubernetes.

Beneficios Helm

Los Helm Charts sirven para describir incluso las aplicaciones más complejas. Ofrecen una instalación repetible de la aplicación, manteniendo un único punto de control.

Las actualizaciones de Helm Charts son sencillas y más fáciles de utilizar para los desarrolladores. Los Helm Charts buscan ser fáciles de versionar, compartir y alojar a través de tecnologías cloud.

Al desplegar una aplicación nueva, es posible tener de devolverla por cualquier razón. El proceso de "rollback" con Helm Charts es sencillo: se usa para retroceder la versión a una publicación anterior en caso de que el proceso no haya sido satisfactorio.

Una de las funciones más interesantes de este software es que nos permite crear nuestras propias cartas de navegación.

Instalación y documentación

<https://helm.sh/es/docs/intro/install/>

BIBLIOGRAFIA



[Página oficial Kubernetes](#)

[Red hat + Kubernetes](#)

[Curso Kubernetes \(OpenWebinars\)](#)

[Vídeo arquitectura](#)

[Apuntes universidad de Almería](#)

[Tutoriales devopsschool](#)