

KUBERNETES

Instalación básica

Kubectl

```
curl.exe -LO "https://dl.k8s.io/release/v1.33.0/bin/windows/amd64/kubectl.exe"
```

Verificación: kubectl --help

Minikube

Descarga

```
New-Item -Path 'c:\' -Name 'minikube' -ItemType Directory -Force  
$ProgressPreference = 'SilentlyContinue'; Invoke-WebRequest -OutFile  
'c:\minikube\minikube.exe' -Uri  
'https://github.com/kubernetes/minikube/releases/latest/download/minikube-  
windows-amd64.exe' -UseBasicParsing
```

Agregar al path (ejecutar como admin)

```
$oldPath = [Environment]::GetEnvironmentVariable('Path',  
[EnvironmentVariableTarget]::Machine)  
if ($oldPath.Split(';') -inotcontains 'C:\minikube'){  
    [Environment]::SetEnvironmentVariable('Path', ${'{}0};C:\minikube' -f $oldPath),  
    [EnvironmentVariableTarget]::Machine  
}
```

Verificación: minikube --help

Iniciar minikube con docker (es necesario tener docker abierto)

```
minikube start --drive=docker
```

Comandos basicos

```
kubectl get nodes  
minikube addons list
```

Activar addons

```
minikube addons enable registry  
minikube addons enable metrics-server  
eval $(minikube docker-env) // para linux  
minikube docker-env | Invoke-Expression // para windows  
docker images
```

Saber el contexto de donde se trabaja y cambiarlo

```
kubectl config get-contexts  
kubectl config use-context minikube
```

Hello World en Kubernetes

```
kubectl run hello-cloud --image=gcr.io/google-samples/hello-app:2.0 --restart=Never --  
port=8080
```

Uso de alias:

```
alias k="kubectl" # para Linux  
Set-Alias -Name k -Value kubectl # para windows
```

Crear alias para windows de forma permanente:

```
Add-Content -Path $PROFILE -Value "`nfunction k { kubectl @args }`n"
```

Verificar ahora el pod

```
k get pods
```

Dashboard de kubernetes con minikube

```
minikube dashboard
```

Arquitectura de kubernetes

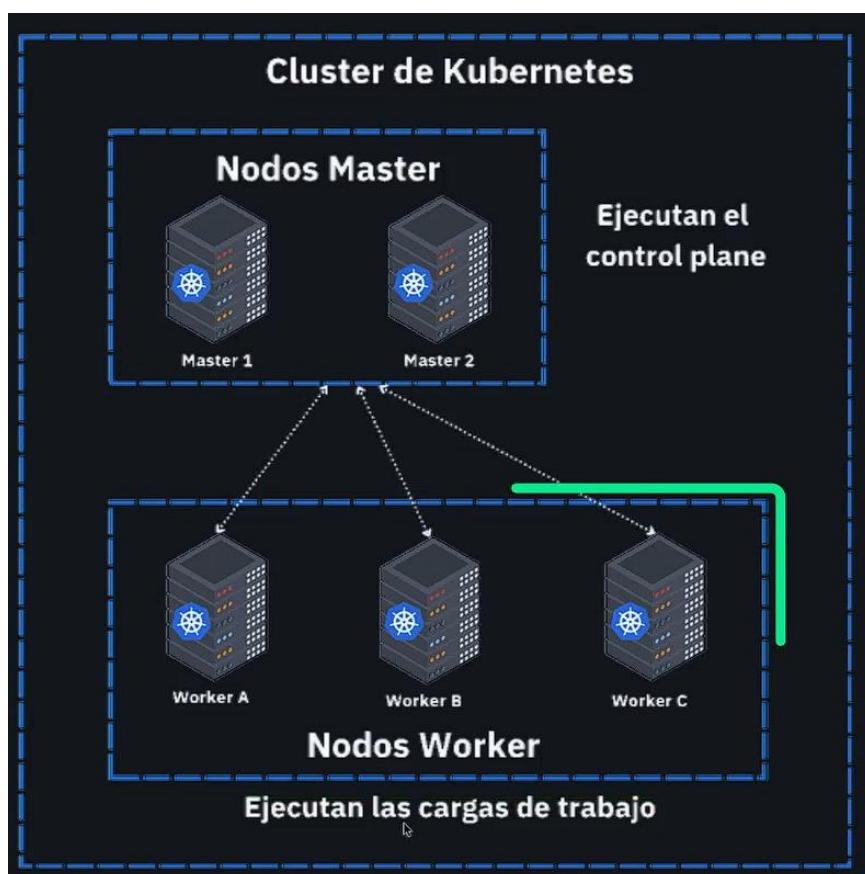
Nodos Master: Son servidores que nos permiten alojar nuestro control plane

Nodos Worker: Tambien llamados trabajadores o slaves, nos permiten gestionar exclusivamente las cargas de computo

Recomendación:

Tener más de un servidor en la capa de nodos maestros, permitiendo una alta disponibilidad y más resiliente a fallos.

La cantidad de nodos workers va a variar dependiendo de cuantas aplicaciones tenemos en ejecución.



Nodo Master:

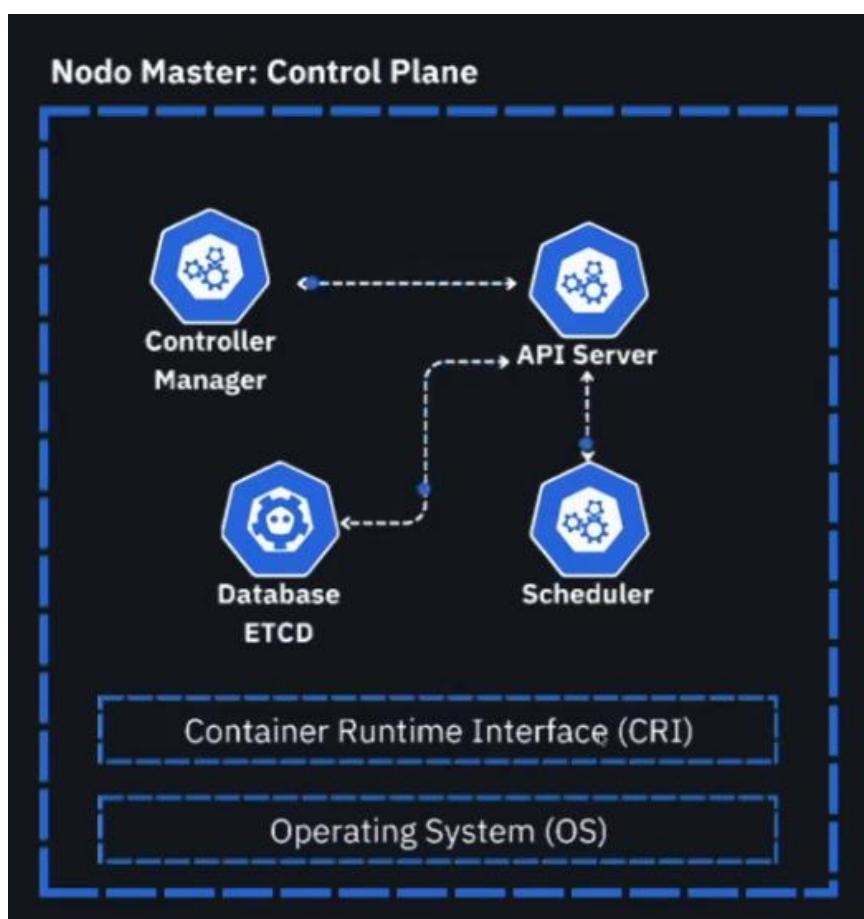
API Server: Es nuestro punto de entrada para toda la comunicación dentro del cluster. Para desplegar una aplicación vamos a usar los comandos como kubectl.

ETCD: Es nuestra base de datos de clave valor de alta concurrencia nos va a permitir registrar cada cambio que hagamos para que nuestro cluster siempre tenga el estado deseado de forma actualizada.

Controller Manager: Multiples controladores que validan diferentes funcionalidades o requerimientos dentro de nuestro cluster.

- Node Controller: Se encarga de la salud de todos los nodos.
- Replication Controller: Se encarga de que siempre haya una cantidad indicada o específica de pods, lo que más adelante nos ayudará a escalar nuestras aplicaciones en múltiples pods.
- Endpoint Controller: Este nos va a permitir manejar la comunicación de nuestros servicios y permitir que las aplicaciones sean visibles al mundo y los pods tengan el tráfico deseado.

Scheduler: Se encarga de decidir en que nodo se va a ejecutar cada uno de nuestros pods dependiendo de la cpu, gpu, memoria o almacenamiento que espera tener.



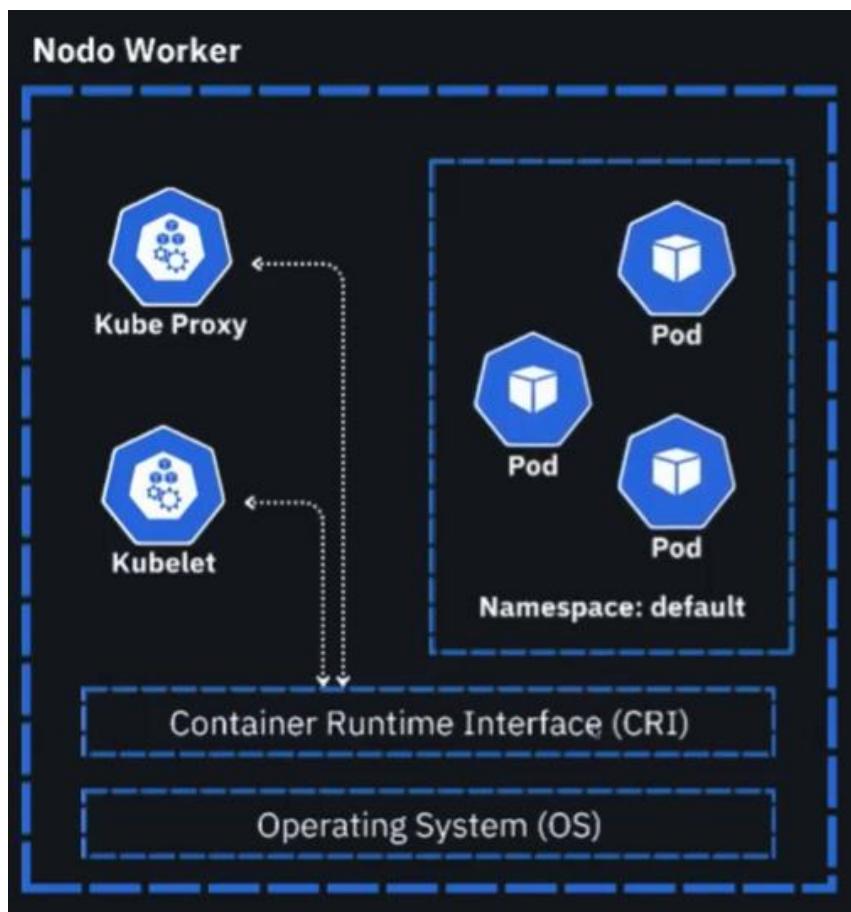
Nodo Worker:

Kubelet: Este es el agente que corre en cada nodo y se va a comunicar con el API Server asegurándose de que cada contenedor se ejecute de forma correcta, si hay algún error en alguno de estos, este va a comunicarlo a los demás componentes para que se tomen las acciones de generar nuevamente el contenedor que está fallando.

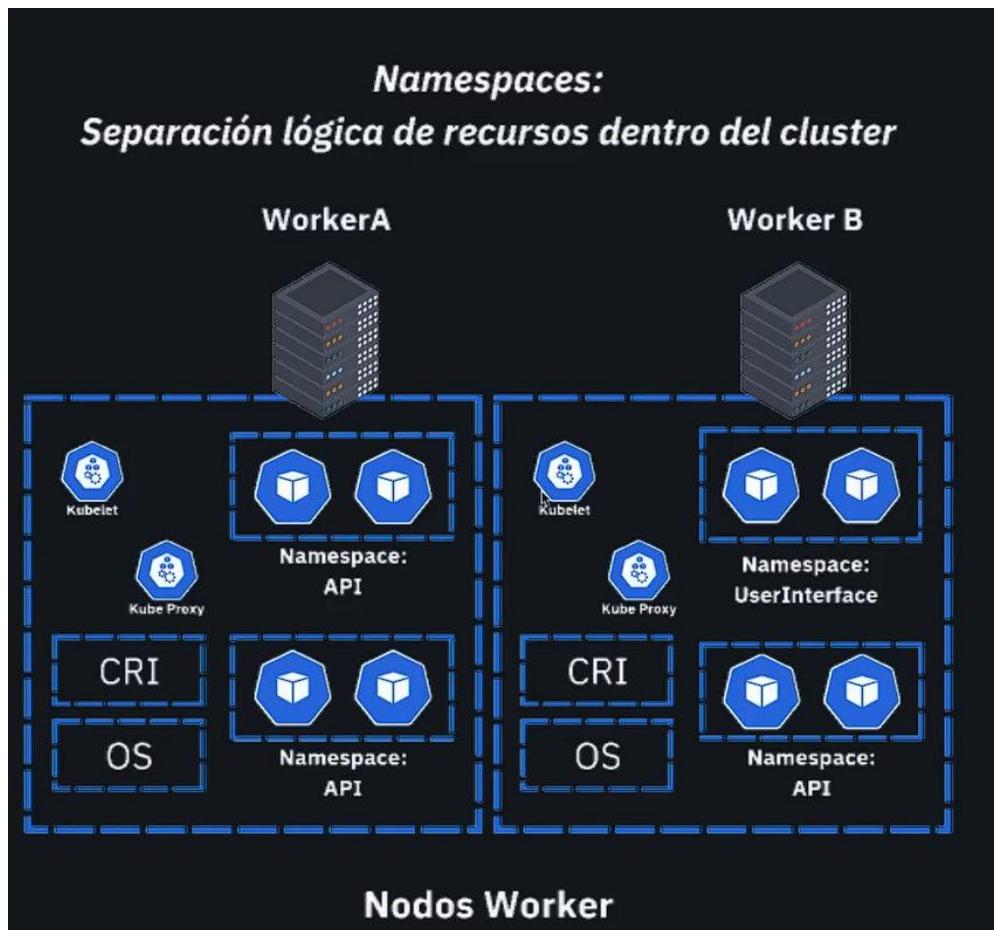
Kube Proxy: Este va a gestionar toda la capa de red de nuestro cluster, este se ejecuta en cada uno de los nodos worker, y se va a permitir que los pods se comuniquen entre sí y con el exterior.

Container Runtime Interface (CRI): Es el encargado de ejecutar los contenedores dentro del nodo. Se cambió de docker a containerd actualmente.

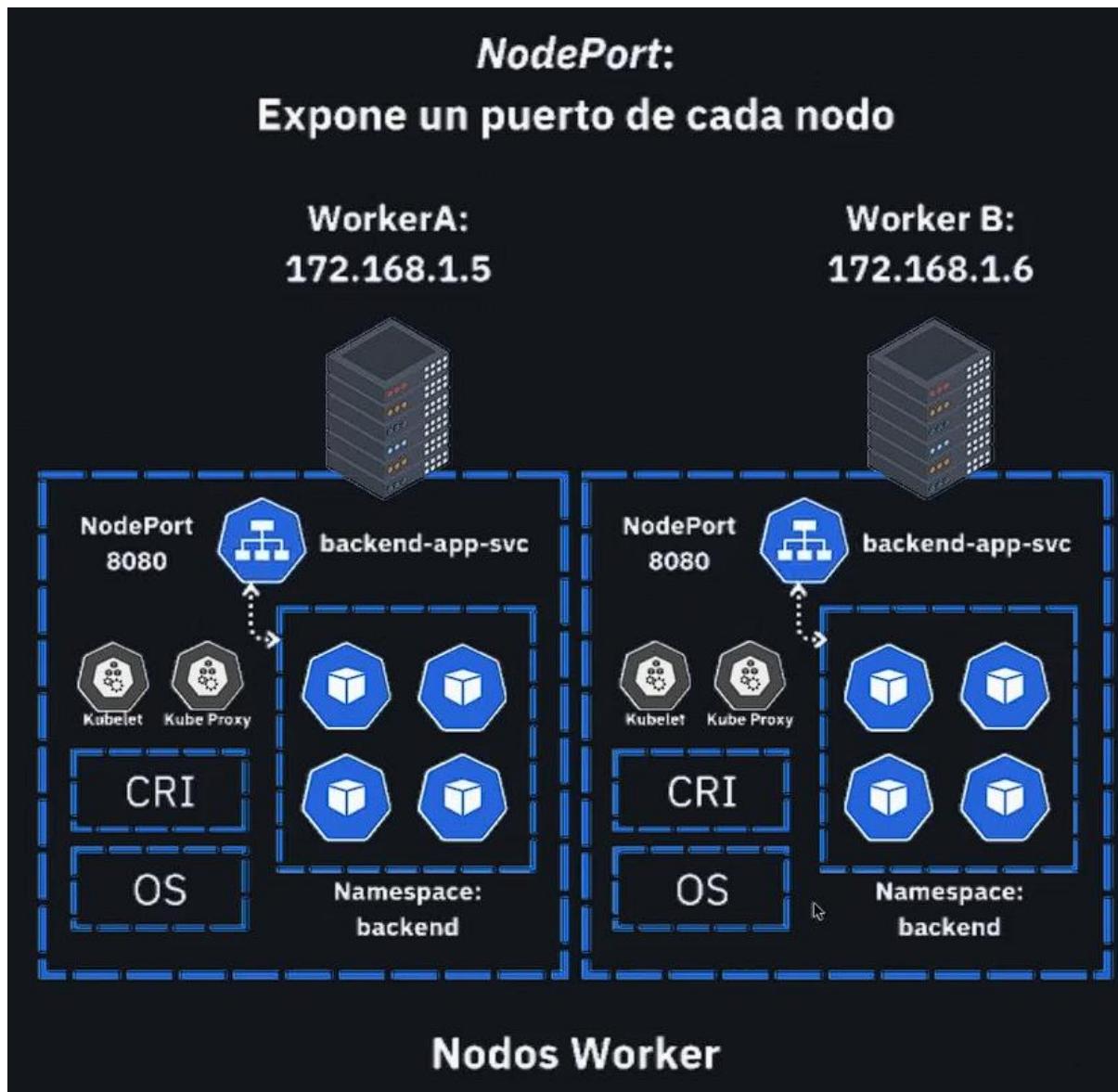
Namespaces: Son la separación lógica de recursos y dentro de cada uno de estos va a vivir nuestro pod.



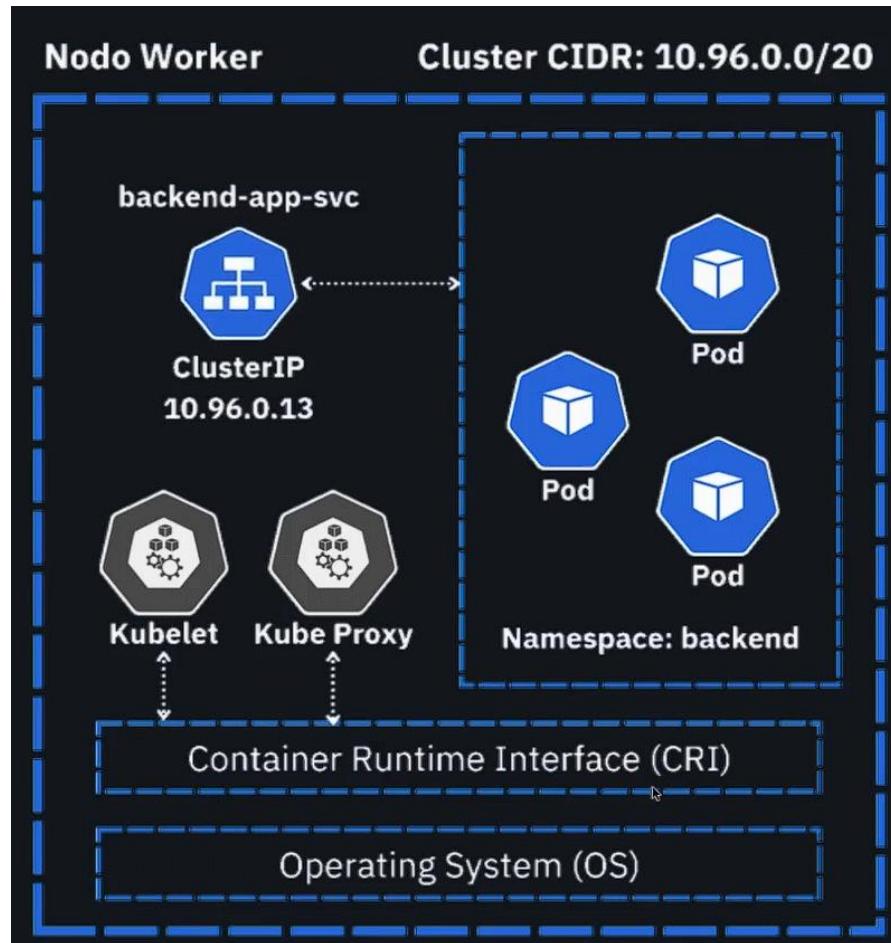
Se puede tener una separación por backend y frontend o por tipo de aplicación, equipo de pago, equipo de facturación.



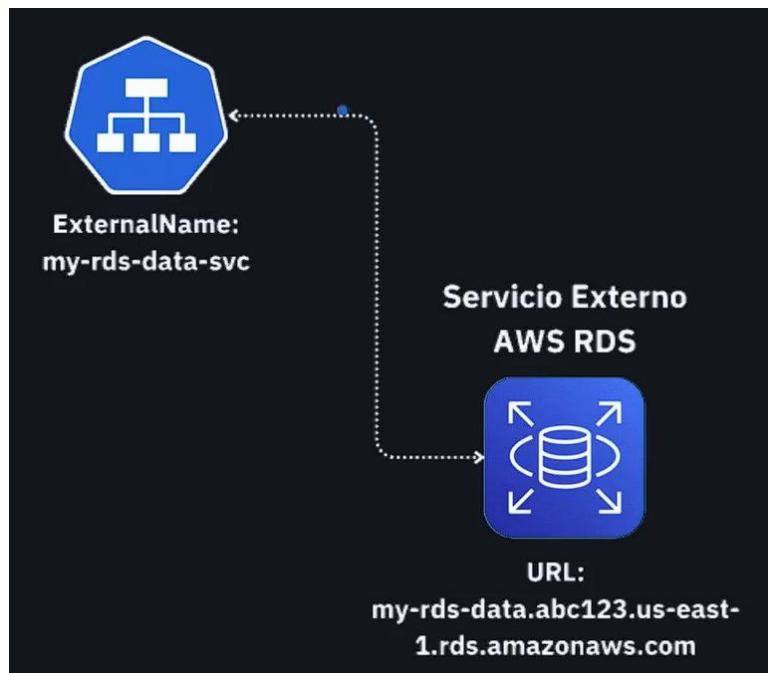
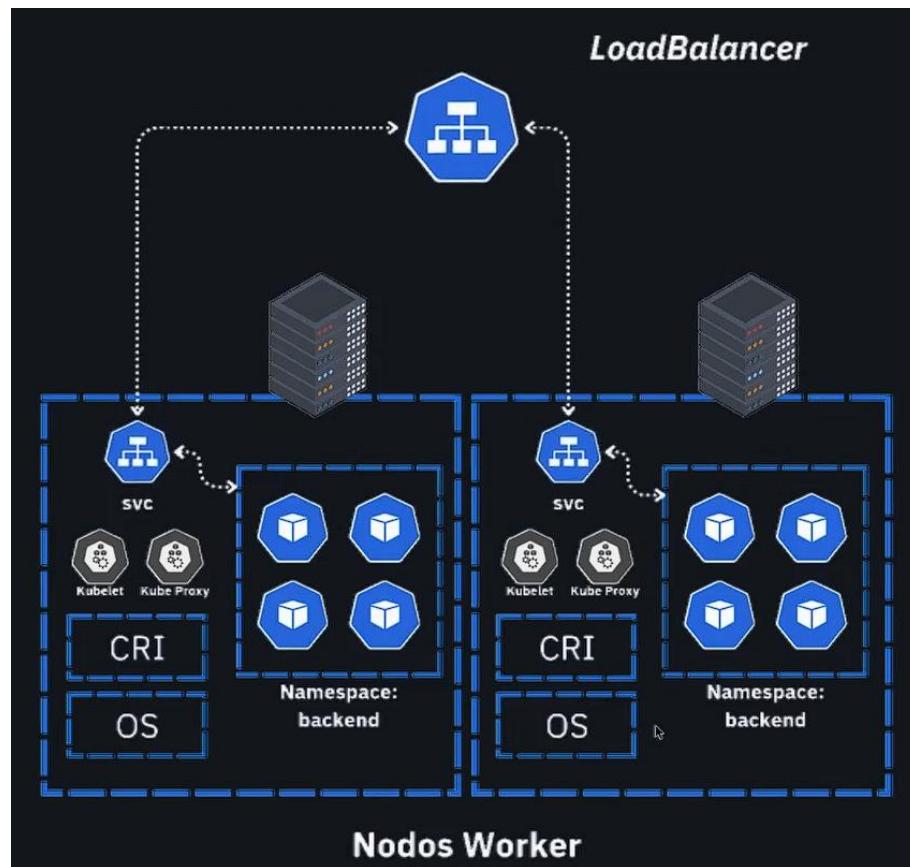
El NodePort es un tipo de servicio que nos va a permitir exponer un puerto de cada uno de nuestros nodos worker para que nos permita la comunicación hacia dentro del cluster con cada uno de nuestros pods

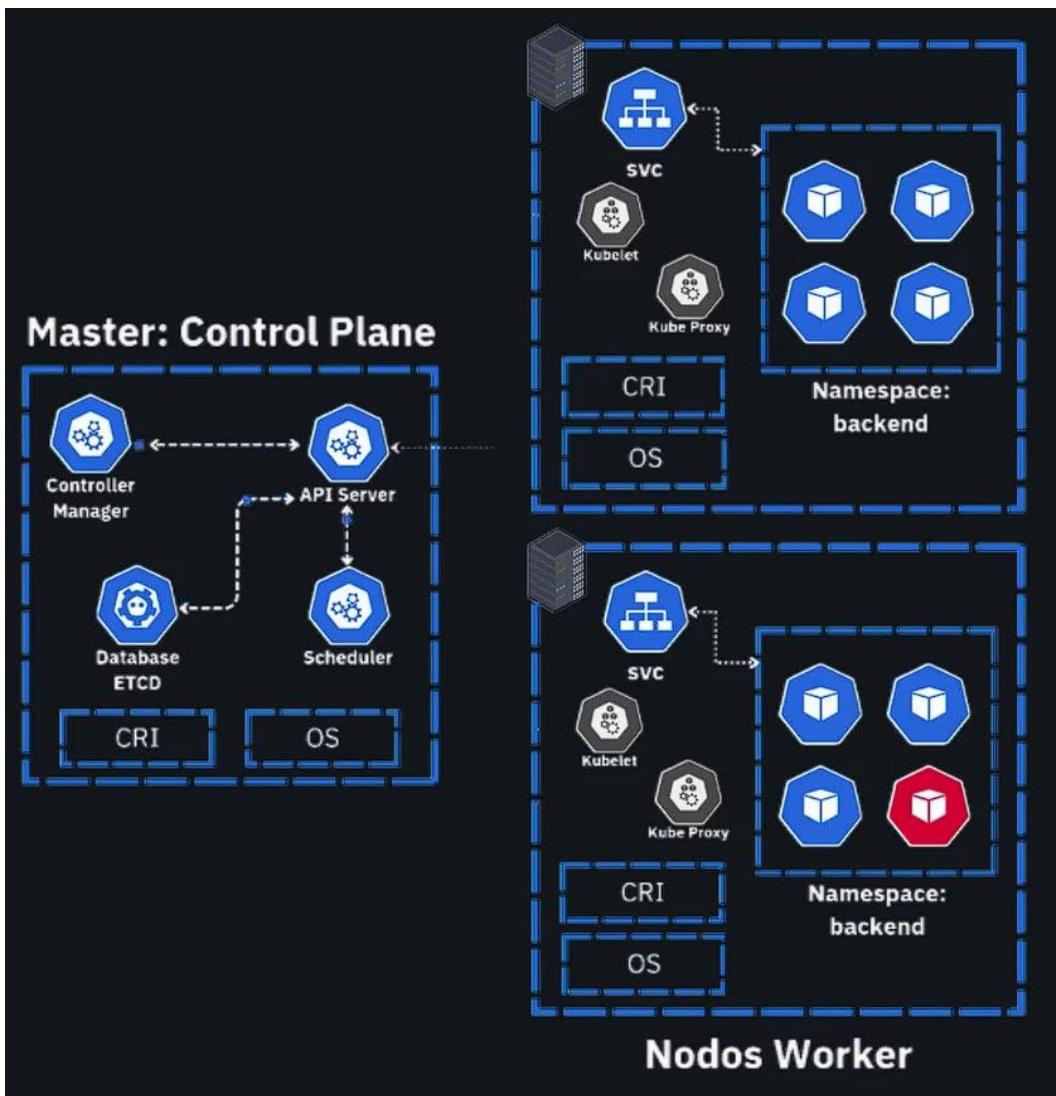


ClusterIP: Es un tipo de servicio nos va a permitir una comunicación interna dentro del cluster. Utiliza una dirección IP del cluster para la comunicación de esa dirección IP específica para un grupo de pods que tenemos asociados a nuestros servicios.



LoadBalancer: Es un tipo de servicio va a garantizar un tráfico más escable y resiliente en entornos productivos, porque se están agregando componentes de red que permiten el balanceo de carga de forma más optima.





API de Kubernetes y kubectl

Obtener namespaces

```
kubectl get namespaces # ó  
k get ns
```

Ver todos los componentes de k8s

```
k get pods -o wide -A
```

Crear un namespace

```
k create ns demoapp
```

Consultar namespace

```
k get ns
```

Eliminar namespace

```
k delete ns demoapp
```

Información del nodo creado de minikube

```
k describe node minikube
```

Crear pod

```
k apply -f ./simple-pod.yaml
```

Describe del pod

```
k describe pod lonely-pod
```

Eliminar pod

```
k delete pod lonely-pod
```

Enfoque declarativo e imperativo

La principal diferencias entre estos 2 son que, el imperativo ejecuta directamente comandos para una tarea específica y el declarativo ejecuta archivos que dentro tienen instrucciones que debe de seguir.

05-declarative-vs-imperative

```
# Imperative
```

```
## Create a pod imperatively
```

```
kubectl run mypod --image=nginx
```

```
## Get a pod imperatively
```

```
kubectl get pods
```

```
## Delete a pod imperatively
```

```
kubectl delete pod mypod
```

```
# Declarative
```

```
## Create a pod declaratively
```

```
kubectl apply -f mypod.yaml
```

```
## Delete a pod declaratively
```

```
kubectl delete -f mypod.yaml
```



Pods, ReplicaSets y Deployments

ReplicaSets:

- Se utilizan para garantizar que un número especificado de réplicas de un pod estén corriendo en cualquier momento.
- Asegura la disponibilidad y escalabilidad automática de los pods.
- Usa un selector para identificar los pods que debe gestionar.

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
  labels:
    app: nginx
spec:
  replicas: 3 # numero de replicas
  selector: # son importantes para saber que pods deben gestionar
  matchLabels:
    app: nginx # los nodos con este nombre están asociados al replicaset
template:
  metadata:
    labels:
      app: nginx
  spec:
    # Estos nos van a permitir relacionar toda la información de la imagen del
    # contenedor, la versión del tag y el puerto que queremos exponer
    containers:
      - name: nginx
        image: nginx:latest
        ports:
          - containerPort: 80

```

Ver pods con 3 replicas

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kub
  NAME          READY  STATUS   RESTARTS  AGE
  hello-cloud    0/1    Error     0        14d
  nginx-replicaset-bpt6w  1/1    Running   0        5s
  nginx-replicaset-l26tz  1/1    Running   0        5s
  nginx-replicaset-n76qz  1/1    Running   0        5s
```

Ver el número de réplicas

```
k get replicaset
```

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\De
  NAME      DESIRED  CURRENT  READY  AGE
  nginx-replicaset  3        3        3    105s
```

Eliminar pod para probar que k8s lo levanta

```
k delete pod xxx
```

Este automáticamente lo levanta.

Deployments:

- Proveen una manera declarativa de actualizar aplicaciones y gestionar la versión de los pods.
- Supervisan y actualizan ReplicaSets.
- Permiten el despliegue de nuevas versiones de la aplicación sin tiempo de inactividad.

```
apiVersion: apps/v1
kind: Deployment # Nuevo tipo
metadata:
  name: hello-deployment
  labels:
    app: hello
spec:
  replicas: 4
  selector:
    matchLabels:
      app: hello
  template:
    metadata:
      labels:
        app: hello
    spec:
      containers:
        - name: hello-app
          image: gcr.io/google-samples/hello-app:1.0
          ports:
            - containerPort: 8080
```

Deployment con 4 replicas

NAME	READY	STATUS	RESTARTS	AGE
hello-cloud	0/1	Error	0	14d
hello-deployment-584f88f49b-9hm6s	0/1	ContainerCreating	0	4s
hello-deployment-584f88f49b-cwhvf	0/1	ContainerCreating	0	4s
hello-deployment-584f88f49b-pkw78	1/1	Running	0	4s
hello-deployment-584f88f49b-wjxqn	0/1	ContainerCreating	0	4s
nginx-replicaset-9lwfx	1/1	Running	0	4m57s
nginx-replicaset-l26tz	1/1	Running	0	8m57s
nginx-replicaset-n76qz	1/1	Running	0	8m57s

Consultar deployments

```
k get deployment
```

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11.
  NAME          READY   UP-TO-DATE   AVAILABLE   AGE
  hello-deployment   4/4      4           4          111s
```

Consultar replicaset de nuevo

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes
  NAME          DESIRED   CURRENT   READY   AGE
  hello-deployment-584f88f49b   4         4         4        2m15s
  nginx-replicaset            3         3         3        11m
```

Verificar a que deployment pertenece el replicaset

```
k describe replicaset xxx_name
```

```
84f88f49b
Name:          hello-deployment-584f88f49b
Namespace:     default
Selector:      app=hello,pod-template-hash=584f88f49b
Labels:        app=hello
               pod-template-hash=584f88f49b
Annotations:   deployment.kubernetes.io/desired-replicas: 4
               deployment.kubernetes.io/max-replicas: 5
               deployment.kubernetes.io/revision: 1
Controlled By: Deployment/hello-deployment ↙
```

Cambiar la imagen del deployment sin modificar el yaml

```
k set image deployment/hello-deployment hello-app=gcr.io/google-samples/hello-
app:2.0
```

Verificar el progreso de la actualización

```
kubectl rollout status deployment/hello-deployment
```

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11.
  deployment
  deployment "hello-deployment" successfully rolled out
  previous: <none> → hello-deployment-584f88f49b
  pods: 4 → 4
```

Al correr el rollout con una imagen inexistente “hello-app:3.0”, verificamos que hay pods con errores

NAME	READY	STATUS	RESTARTS	AGE
hello-cloud	0/1	Error	0	14d
hello-deployment-577b47bd4c-hwts7	1/1	Running	0	3m24s
hello-deployment-577b47bd4c-lb5qt	1/1	Running	0	3m26s
hello-deployment-577b47bd4c-z975l	1/1	Running	0	3m26s
hello-deployment-657cbcfd fd-prz5d	0/1	ErrImagePull	0	10s
hello-deployment-657cbcfd fd-whrwp	0/1	ErrImagePull	0	10s
nginx-replicaset-9lwfx	1/1	Running	0	15m
nginx-replicaset-l26tz	1/1	Running	0	19m
nginx-replicaset-n76qz	1/1	Running	0	19m

Para esto podemos regresar los cambios anterior con:

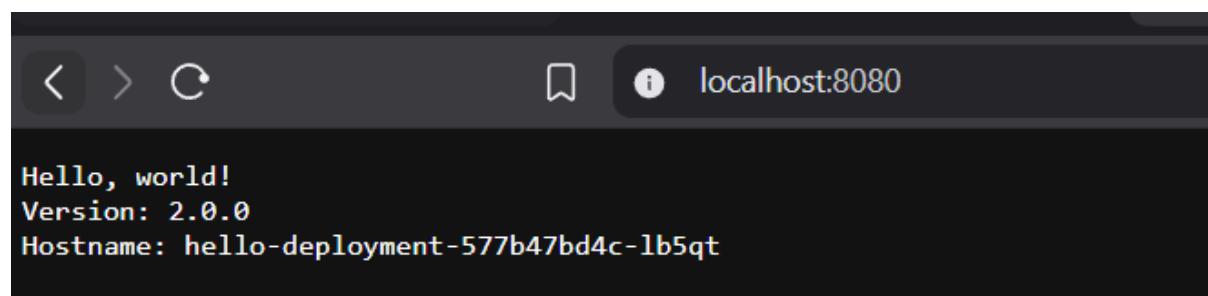
```
kubectl rollout undo deployment/hello-deployment
```

Con esto verificamos que los pods nuevamente estan activos

NAME	READY	STATUS	RESTARTS	AGE
hello-cloud	0/1	Error	0	14d
hello-deployment-577b47bd4c-7m5cc	1/1	Running	0	16s
hello-deployment-577b47bd4c-hwts7	1/1	Running	0	5m22s
hello-deployment-577b47bd4c-lb5qt	1/1	Running	0	5m24s
hello-deployment-577b47bd4c-z975l	1/1	Running	0	5m24s
nginx-replicaset-9lwfx	1/1	Running	0	17m
nginx-replicaset-l26tz	1/1	Running	0	21m
nginx-replicaset-n76qz	1/1	Running	0	21m

Exponer un Deployment

```
kubectl port-forward deploy/hello-deployment 8080:8080
```



Stateless (sin estado):

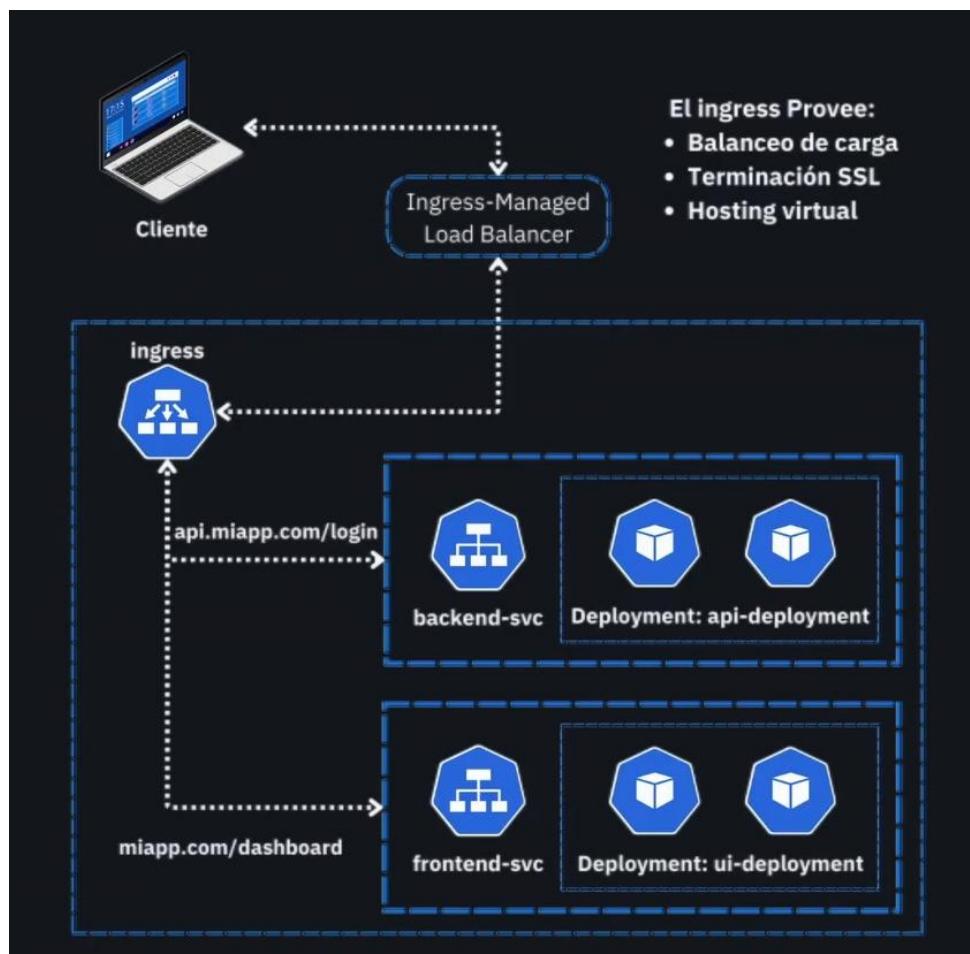
- No guardan datos persistentes entre reinicios.
- Ejemplo: Servidores web como Nginx o aplicaciones que procesan solicitudes HTTP.
- Escalabilidad sencilla: Puedes agregar o eliminar réplicas sin preocuparte por la consistencia de datos.

Stateful (con estado):

- Guardan datos persistentes y necesitan mantener el estado entre reinicios.
- Ejemplo: Bases de datos como MySQL o Redis.
- Requieren volúmenes persistentes (Persistent Volumes) para almacenar datos.

Servicios e Ingress: Exposición de aplicaciones

Un Ingress se utiliza para gestionar el acceso externo a los servicios dentro de un cluster, permitiendo el enruteamiento de tráfico HTTP y HTTPS proporcionando características como balanceo de carga y terminación SSL.



Tipos de Ingress



Habilita el Ingress controller

```
minikube addons enable ingress
```

Verifica que el NGINX Ingress controller esté funcionando:

```
kubectl get pods -n ingress-nginx
```

```
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> kubectl get pods -n ingress-nginx
NAME                               READY   STATUS    RESTARTS   AGE
ingress-nginx-admission-create-rkw4r   0/1     Completed   0          25s
ingress-nginx-admission-patch-nsg8h   0/1     Completed   1          25s
ingress-nginx-controller-67c5cb88f-658q2   0/1     Running    0          25s
```

Crear un deployment de ejemplo

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

Exposición del Deployment usando NodePort

```
kubectl expose deployment web --type=NodePort --port=8080
```

Verificar si el servicio ya existe con

```
k get service
```

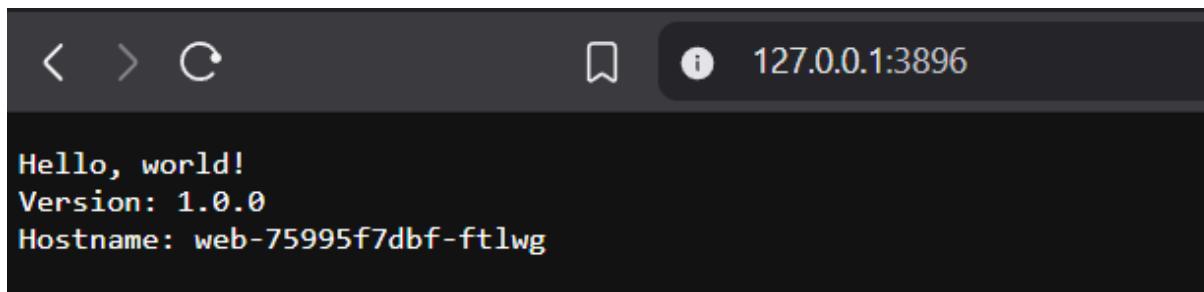
```
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> k get service
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
kubernetes  ClusterIP  10.96.0.1      <none>           443/TCP      14d
web       NodePort    10.111.74.123  <none>           8080:32453/TCP  89s
```

Eliminar el servicio existente

```
k delete service web
```

Accede al servicio usando la IP proporcionada con minikube

```
minikube service web --url
```



Instalación del Ingress de ejemplo:

```
kubectl apply -f https://k8s.io/examples/service/networking/example-ingress.yaml
```

Verifica que el Ingress esté correctamente configurado

```
kubectl get ingress
```

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> kubectl get ingress
NAME      CLASS      HOSTS      ADDRESS      PORTS      AGE
example-ingress  nginx    hello-world.example  192.168.49.2  80        49s
```

Más información del ingress

```
k describe ingress xxx_name
```

```
● PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> kubectl describe ingress example-ingress
Warning: v1 Endpoints is deprecated in v1.33+; use discovery.k8s.io/v1 Endpoints instead.
Name:           example-ingress
Labels:         <none>
Namespace:      default
Address:        192.168.49.2
Ingress Class:  nginx
Default backend: <default>
Rules:
Host          Path  Backends
----          ---   -----
hello-world.example  /    web:8080 (10.244.0.37:8080)
Annotations:   <none>
```

URL para ingresar al Ingress

Crear el tunnel en minikube para visualizar la web con el host

```
minikube tunnel
```

Para windows sino no funciona:

Agregar al archivo C:\Windows\System32\drivers\etc\hosts la línea:

```
127.0.0.1 hello-world.example
```

Detener IIS si está corriendo (esto me bloqueaba el puerto), ejecutar en modo admin

```
iisreset /stop
```

ConfigMaps y Secrets: Configuración y datos sensibles

ConfigMaps

Este objeto nos va a permitir añadir y guardar todas las configuraciones de nuestra aplicación, pero no valores sensibles. Por ejemplo: el entorno donde se encuentra nuestra aplicación (desarrollo, staging, producción), urls de acceso externo.

```
apiVersion: v1
kind: ConfigMap # Nuevo tipo
metadata:
  name: auth-config
data:
  url: "https://auth.service.local"
```

Crear configmap

```
kubectl apply -f auth-config.yaml
```

Ver configmap

```
k get configmaps
```

Más información

```
k describe configmap xxx_name
```

Aparece la información completa

```
Data
=====
url:
-----
https://auth.service.local
```

Secrets:

Este objeto nos permite guardar información sensible

Crear secret de forma imperativa

Windows:

```
kubectl create secret generic auth-secret `  
  --from-literal=client_id=myclientid `  
  --from-literal=client_secret=secret
```

Linux:

```
kubectl create secret generic auth-secret \  
  --from-literal=client_id=myclientid \  
  --from-literal=client_secret=secret
```

Ver secret

```
k get secret
```

Más información

```
k describe secret xxx_name
```

Aparecen en modo opaco

```
Type: Opaque  
  
Data  
====  
client_id: 10 bytes  
client_secret: 6 bytes  
PS C:\Users\g10n\OneDrive\Escritorio
```

Puede editar los valores del secret, pero estarán encriptados en base64 (decodificable)

```
k edit secret auth-secret
```

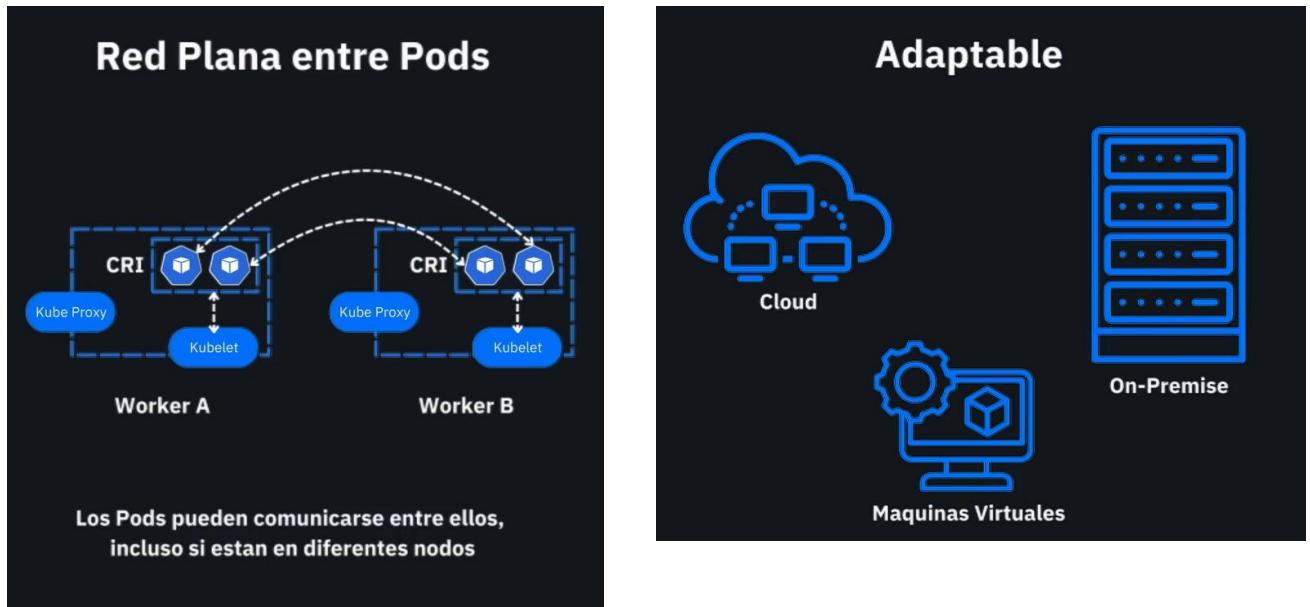
```
"  
apiVersion: v1  
data:  
  client_id: bXljbGllbnRpZA==  
  client_secret: c2VjcmV0  
kind: Secret  
metadata:  
  creationTimestamp: "2025-09-05T19:10:38Z"  
  name: auth-secret  
  namespace: default  
  resourceVersion: "50862"  
  uid: e3e61635-1af1-409b-a965-1be0364a447b  
type: Opaque
```

Se pueden conectar servicios más robustos para mantener estos secretos, como un servicio de terceros de aws, azure, etc.

```
apiVersion: v1
kind: Secret # Nuevo tipo
metadata:
  name: auth-secret
type: Opaque
data:
  client_id: "bXljbGllbnRpZA=="
  client_secret: "c2VjcmV0"
```



Modelo de red en Kubernetes



Principios fundamentales de kubernetes

Regla 1: Todos los nodos deben poder conectarse entre sí sin NAT.

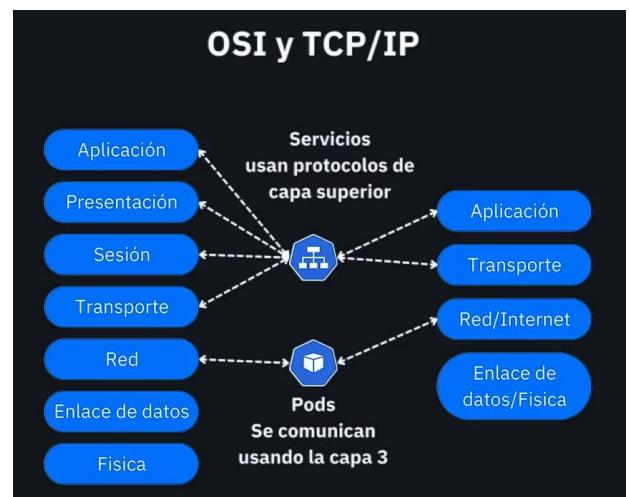
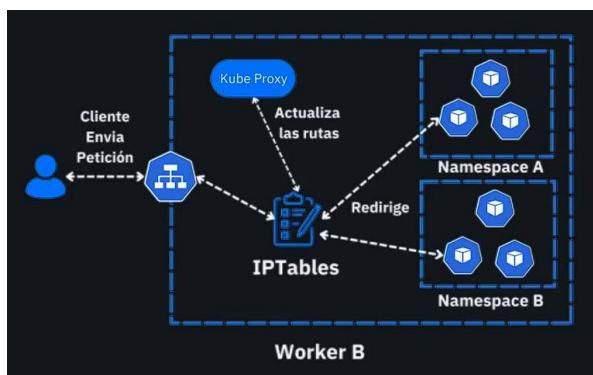
Regla 2: Todos los pos deben poder conectarse entre sí sin NAT.

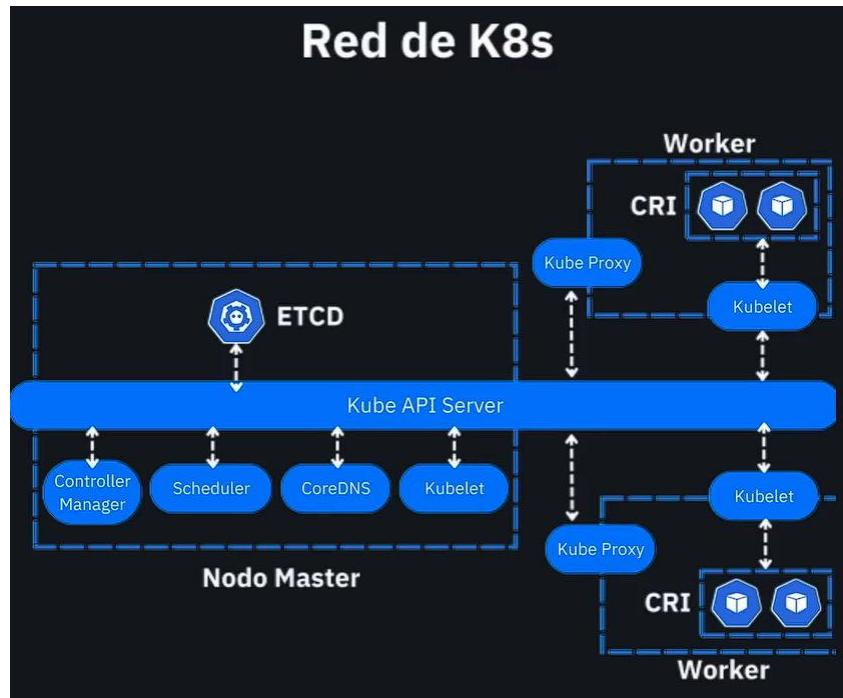
Regla 3: Todos los pods y servicios comparten el mismo segmento de red.

Container Network Interface (CNI)

- Calico
- Flannel

Kube Proxy: Nos permite hacer un enrutamiento de cada una de las peticiones a nivel de network.





CoreDNS: Se encarga de hacer todo el service descory entre los servicios que vamos creando dentro de nuestro cluster y los grupos de pods asociados a este.

La capacidad máxima de nodos, pods y servicios en Kubernetes depende de la implementación y la versión utilizada, pero generalmente, un clúster de Kubernetes puede manejar hasta 5000 nodos y 150,000 pods, con un máximo de 300,000 conexiones simultáneas a la API. Cada nodo puede tener múltiples servicios, pero el número exacto puede variar dependiendo de la arquitectura específica y los recursos del sistema. Estas cifras son orientativas y pueden cambiar según las configuraciones y optimizaciones.

Tipos de servicios: ClusterIP, NodePort, LoadBalancer y ExternalName

NodePort:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment-np
  labels:
    app: hello
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-nodeport

```

```

template:
  metadata:
    labels:
      app: hello-nodeport #Mismo valor
  spec:
    containers:
      - name: hello-app
        image: gcr.io/google-samples/hello-app:1.0
    ports:
      - containerPort: 8080 #Mismo puerto

-----
apiVersion: v1
kind: Service
metadata:
  name: hello-service-np
spec:
  type: NodePort
  selector:
    app: hello-nodeport #Mismo valor
  ports:
    - port: 80 #Salida puerto
      targetPort: 8080 #Mismo puerto
      nodePort: 30007

```

Consultar servicios

```

k get pods
k get services

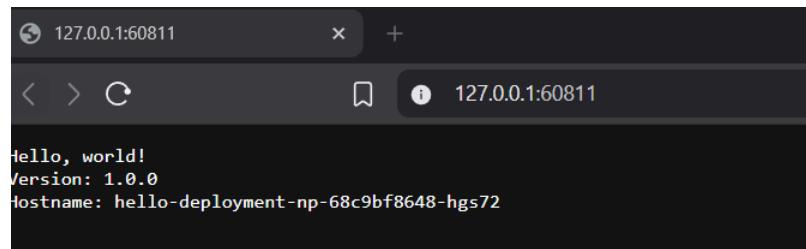
```

Eliminar otros recursos como deploy, replicaset, service con “`k delete xxx_resource xxx_name`”

Crear un tunnel entre el servicio y el NodePort usando minikube

```
minikube service hello-service-np
```

Luego se abre automáticamente



ClusterIP:

Es lo mismo que el anterior pero solo cambia el nombre y el servicio

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment-clusterip
  labels:
    app: hello
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-clusterip
  template:
    metadata:
      labels:
        app: hello-clusterip
    spec:
      containers:
        - name: hello-app
          image: gcr.io/google-samples/hello-app:2.0
          ports:
            - containerPort: 8080
      ---  

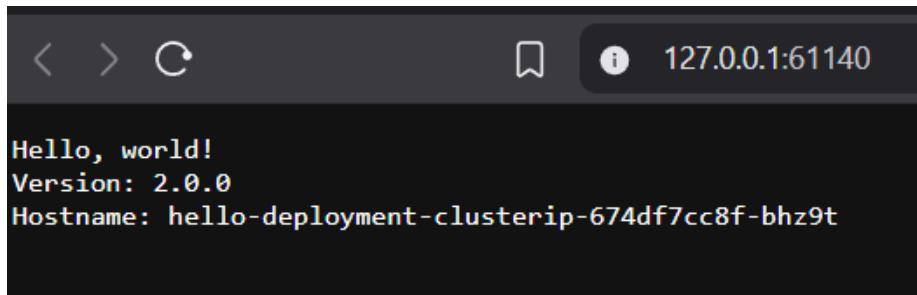
apiVersion: v1
kind: Service
metadata:
  name: hello-service-clusterip
spec:
  type: ClusterIP
  selector:
    app: hello-clusterip
  ports:
    - port: 80
      targetPort: 8080
```

Consultar servicios

```
k get pods
k get services
```

Publicar servicio con minikube

```
minikube service hello-service-clusterip
```



LoadBalancer:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deployment-loadbalancer
  labels:
    app: hello
spec:
  replicas: 2
  selector:
    matchLabels:
      app: hello-loadbalancer
  template:
    metadata:
      labels:
        app: hello-loadbalancer
    spec:
      containers:
        - name: hello-app
          image: gcr.io/google-samples/hello-app:2.0
      ports:
        - containerPort: 8080
  ---  

apiVersion: v1
kind: Service
metadata:
  name: hello-service-loadbalancer
spec:
  type: LoadBalancer
  selector:
```

```
  app: hello-loadbalancer
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080
```

Consultar servicios

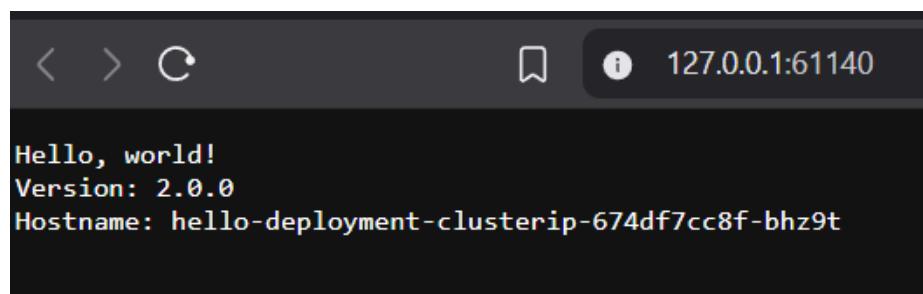
```
k get pods
k get services
```

A diferencia de los anteriores este espera a que se le asigne un external-ip

```
PS C:\Users\giron_\OneDrive\Escritorio\Curso_Platzi\DevOps\11_Kubernetes\kubernetes\10-services-clusterip> k get services
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
hello-service-clusterip  ClusterIP  10.106.230.132 <none>        80/TCP    8m34s
hello-service-loadbalancer LoadBalancer  10.105.85.141  <pending>     80:30685/TCP  9s
hello-service-np          NodePort   10.96.35.233  <none>        80:30007/TCP  22m
kubernetes               ClusterIP  10.96.0.1    <none>        443/TCP   17m
PS C:\Users\giron_\OneDrive\Escritorio\Curso_Platzi\DevOps\11_Kubernetes\kubernetes\10-services-clusterip>
```

Publicar servicio con minikube

```
minikube tunnel
```



Es necesario parar IIS:

```
iisreset /stop
```

Para verificar los balanceadores:

Linux

```
watch -n 0.5 curl localhost:80
```

Windows:

```
while ($true) {
    Invoke-WebRequest -Uri http://localhost:80 -UseBasicParsing
    Start-Sleep -Seconds 1
}
```

ExternalName:

Este es importante para conectar otros servicios con kubernetes, por ejemplo conectar la base de datos de aws al cluster

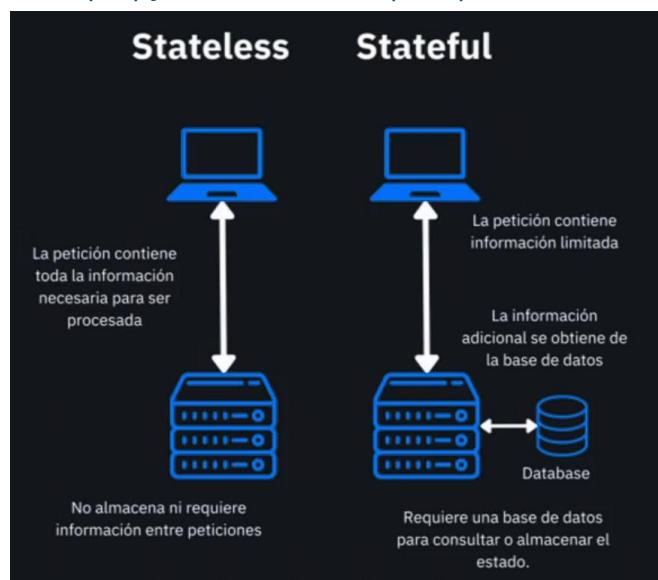
```
apiVersion: v1
kind: Service
metadata:
  name: my-database-service
spec:
  type: ExternalName
  externalName: my-database.cluster-abcdef123456.us-west-2.rds.amazonaws.com
```

Resumen

Los tipos de servicios en Kubernetes tienen diferentes casos de uso en la nube:

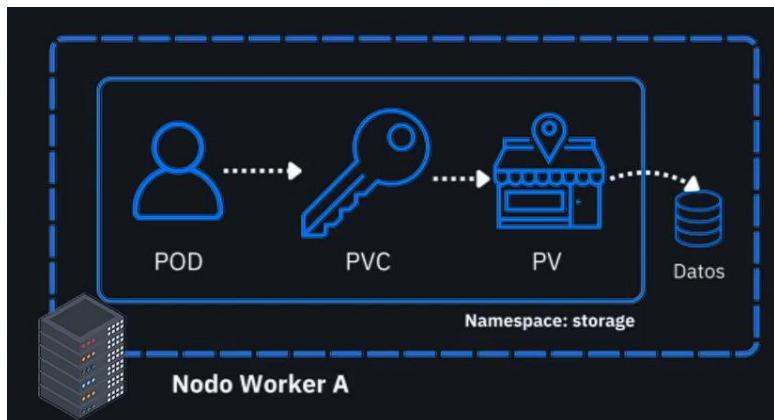
1. **ClusterIP**: Ideal para comunicar servicios internos en un clúster. Por ejemplo, microservicios que se comunican entre sí sin exponer tráfico externo.
2. **NodePort**: Útil para acceder a un servicio desde fuera del clúster utilizando el puerto de un nodo. Puede ser usado para pruebas rápidas de aplicaciones.
3. **LoadBalancer**: Se usa en entornos de producción en la nube (como AWS, GCP, Azure) para distribuir tráfico a múltiples instancias de un servicio. Permite que aplicaciones estén disponibles públicamente.
4. **ExternalName**: Facilita la integración con servicios externos, como bases de datos o APIs, al permitir que se utilice un nombre DNS en lugar de una URL directa, simplificando la configuración.

Volumenes persistentes (PV) y reclamaciones (PVC)

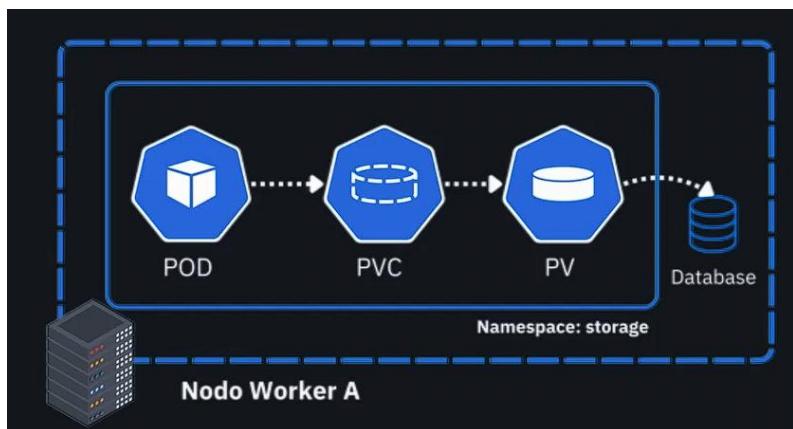


Llave y almacén

PVC (Persistent Volume Claim)



PersistentVolume y PersistentVolumeClaim



Storage Classes: Esto nos permite garantizar no tener la información en local sino un proveedor externo, lo cual nos permite una mejor eficiencia, alta disponibilidad y backup



Primero vamos a crear un directorio para el hostPath

```
minikube ssh  
sudo su  
echo '<h1>Hello from Volume!</h1>' > /mnt/data/index.html
```

Luego aplicar el PV y PVC

```
apiVersion: v1  
kind: PersistentVolume #Nuevo tipo  
metadata:  
  name: my-pv  
  labels:  
    type: local  
    app: nginx-storage # Tienen que ser iguales  
spec:  
  capacity:  
    storage: 1Gi  
  accessModes:  
    - ReadWriteOnce # Hay diferentes modos de acceso, este se monta para 1 solo pod  
  storageClassName: manual # Para entornos locales  
hostPath:  
  path: /mnt/data  
  type: Directory  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: my-pvc  
spec:  
  storageClassName: manual  
  selector:  
    matchLabels:  
      type: local  
    app: nginx-storage  
  accessModes:  
    - ReadWriteOnce  
resources:  
  requests:  
    storage: 1Gi
```

Crear el pod

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html
          name: my-storage
  volumes:
    - name: my-storage
      persistentVolumeClaim: # Referencia al volume con la llave
        claimName: my-pvc
```

Verificar el setup

```
kubectl get pv,pvc
```

Chequear el status del pod

```
kubectl get pod my-pod
```

```
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\11-storage-pv-pvc> kubectl get pv,pvc
NAME           CAPACITY   ACCESS MODES  RECLAIM POLICY  STATUS   CLAIM           STORAGECLASS  VOLUMEATTRIBUTESCLASS
persistentvolume/my-pv  1Gi       RWO         Retain        Bound    default/my-pvc  manual          <unset>
3m7s

NAME          STATUS  VOLUME   CAPACITY  ACCESS MODES  STORAGECLASS  VOLUMEATTRIBUTESCLASS  AGE
● persistentvolumeclaim/my-pvc  Bound   my-pv   1Gi       RWO         manual          <unset>          3m7s
```

```
k describe pod my-pod
```

```
Volumes:
  my-storage:
    Type:     PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName: my-pvc
    ReadOnly:  false
  kube-api-access-t9gkw:
    Type:     Projected (a volume that contains injected data from multiple sources)
    TokenExpirationSeconds: 3607
    ConfigMapName:          kube-root-ca.crt
    ConfigMapOptional:      <nil>
    DownwardAPI:            true
    QoS Class:              BestEffort
```

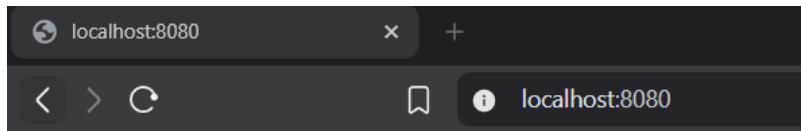
Verificar el contenido montado

```
kubectl exec my-pod -- ls -la /usr/share/nginx/html
```

```
● sr/share/nginx/html
total 12
drwxr-xr-x 2 root root 4096 Sep  8 21:04 .
drwxr-xr-x 3 root root 4096 Aug 13 18:52 ..
-rw-r--r-- 1 root root   28 Sep  8 21:04 index.html
```

Testear el servidor de nginx

```
kubectl port-forward my-pod 8080:80
```



Hello from Volume!

DaemonSets y StatefulSets

Crear un DaemonSet

```
apiVersion: apps/v1
kind: DaemonSet #Nuevo tipo
metadata:
  name: nginx-ds
spec:
  selector:
    matchLabels:
      app: nginx-app #Los pods se crean con este selector para la asociación
  template:
    metadata:
      labels:
        app: nginx-app
    spec:
      containers:
        - name: nginx
          image: nginx
        ports:
          - containerPort: 80
            name: http
      resources: #Especificacion de recursos
        requests: #Empieza con:
          memory: "64Mi"
          cpu: "250m"
      limits: #Maximo de recursos:
        memory: "128Mi"
        cpu: "500m"
```

Verificar el setup

```
kubectl get ds  
kubectl get pods -l app=nginx-app
```

Crear un StatefulSet y su PersistenVolume

PV y PVC

```
apiVersion: v1  
kind: PersistentVolume  
metadata:  
  name: nginx-pv  
  labels:  
    type: local  
    app: nginx-app  
spec:  
  capacity:  
    storage: 1Gi  
  accessModes:  
    - ReadWriteOnce  
  storageClassName: manual  
  hostPath:  
    path: /mnt/data  
    type: Directory  
---  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: nginx-pvc  
spec:  
  storageClassName: manual  
  selector:  
    matchLabels:  
      type: local  
      app: nginx-app  
  accessModes:  
    - ReadWriteOnce  
  resources:  
    requests:  
      storage: 1Gi
```

StatefulSet: Está enfocado para aplicaciones que requieren conexión constamente con una base de datos

```
apiVersion: apps/v1
kind: StatefulSet #Nuevo tipo
metadata:
  name: nginx-sts
spec:
  serviceName: nginx-service
  replicas: 2
  selector:
    matchLabels:
      app: nginx-app
  template:
    metadata:
      labels:
        app: nginx-app
  spec:
    containers:
      - name: nginx
        image: nginx
        ports:
          - containerPort: 80
            name: http
        resources: #Definición de recursos
          requests:
            memory: "64Mi"
            cpu: "250m"
          limits:
            memory: "128Mi"
            cpu: "500m"
    volumeMounts: #Montamos volumen
      - name: nginx-storage
        mountPath: /usr/share/nginx/html
    volumes:
      - name: nginx-storage
    persistentVolumeClaim: #Referencia al PVC
      claimName: nginx-pvc # Updated to match new PVC name
---
apiVersion: v1
kind: Service
metadata:
```

```

name: nginx-service
spec:
  clusterIP: None # Headless service (esto es necesario para la persistencia)
  selector:
    app: nginx-app
  ports:
    - port: 80
      name: http
      targetPort: 80

```

Verificar el setup

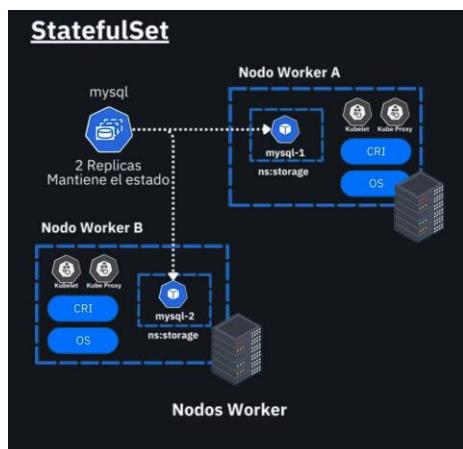
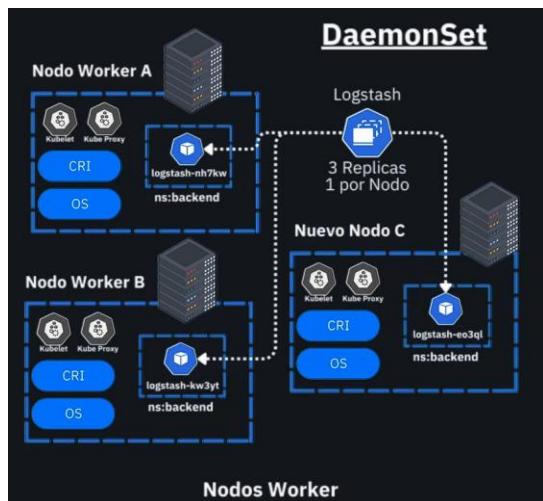
```

kubectl get sts
kubectl get pods -l app=nginx-app
kubectl get pvc my-pvc

```

Eliminar StafefulSet

```
kubectl delete sts nginx-sts
```





RESUMEN

Pod = Unidad Básica

- Es lo más pequeño en Kubernetes
 - Contiene uno o más contenedores que se escalan juntos
 - Son EFÍMEROS (se pueden eliminar y recrear)
-

🚀 LOS 3 TIPOS DE WORKLOADS

1. DEPLOYMENT 🚀 Para aplicaciones web normales

- ✓ Uso: Aplicaciones sin estado (APIs, sitios web)
- ✓ Ventaja: Fácil de escalar y gestionar
- ✗ Limitación: Los pods son intercambiables, pierden datos al reiniciar
 - 💡 Ejemplo: Una API REST que no guarda datos importantes

2. STATEFULSET 🚀 Para aplicaciones que NECESITAN guardar datos

- ✓ Uso: Bases de datos, aplicaciones con estado
- ✓ Ventaja: Cada pod tiene su propio almacenamiento permanente
- ✓ Extra: Cada pod tiene un nombre DNS único y estable (<pod-name>.<service-name>.<namespace>.svc.cluster.local)
 - 💡 Ejemplo: MySQL, PostgreSQL, MongoDB

3. DAEMONSET 🚀 Para servicios que deben estar en TODOS los nodos

- ✓ Uso: Monitoreo, logging, servicios de sistema
 - ✓ Ventaja: Se ejecuta automáticamente en cada servidor del cluster
 - ✓ Auto-escala: Si agregas un servidor, se instala automáticamente
 - 💡 Ejemplo: Agente de monitoreo como Prometheus
-

🧐 ¿CUÁL USAR? - GUÍA RÁPIDA

¿Mi aplicación guarda datos importantes?

NO → DEPLOYMENT

SÍ → STATEFULSET

¿Necesito que algo corra en CADA servidor?

SÍ → DAEMONSET

TABLA COMPARATIVA RÁPIDA

DEPLOYMENT STATEFULSET DAEMONSET

Datos: Se pierden Se mantienen Depende

Escalamiento: Manual Manual Automático

Identidad: Intercambiable Única Una por nodo

Uso típico: Web apps Bases de datos Agentes/Monitoreo

EJEMPLOS DEL MUNDO REAL

DEPLOYMENT:

- Sitio web de e-commerce
- API de usuarios
- Frontend de React/Vue

STATEFULSET:

- Base de datos MySQL
- Sistema de colas como Kafka
- Elasticsearch cluster

DAEMONSET:

- Antivirus en cada servidor
- Recolector de logs
- Agente de backup

Despliegue de una aplicacion multi-tier en Local

Primero es necesario ejecutar estos comandos

```
minikube docker-env | Invoke-Expression #Windows  
eval $(minikube docker-env) #Linux  
iisreset /stop
```

Luego crear las imágenes con docker

```
docker compose build
```

Levantar el servicio con docker

```
docker compose up
```

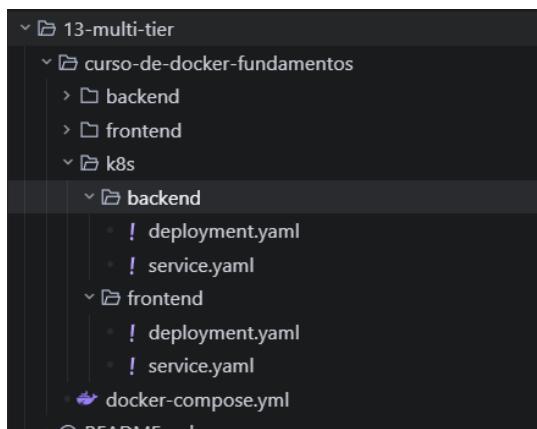
Verificar la URL



Ahora para hacerlo con kubernetes vamos a usar las carpetas de despliegue en yaml del proyecto, para esto tenemos dos archivos:

- Deployment: Sirve para crear el pod con sus replicas
- Service: Para exponer el pod al exterior

Esto debe aplicarse para el backend como para frontend



Se puede desplegar todos los yamls de una carpeta

Para el backend:

```
k apply -f k8s/backend
```

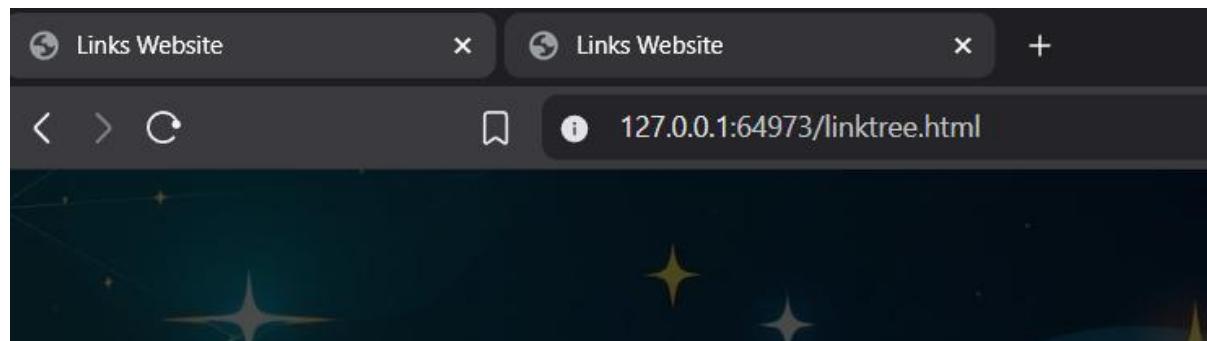
Para el frontend:

```
k apply -f k8s/frontend
```

```
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\13-multi-tier\curso-de-docker-fundamentos>
● k apply -f k8s/backend
  deployment.apps/backend unchanged
  service/backend-service unchanged
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\13-multi-tier\curso-de-docker-fundamentos>k
● apply -f k8s/frontend
  deployment.apps/frontend unchanged
  service/frontend-service unchanged
  no changes
```

Ver el servicio usando minikube

```
minikube service frontend-service
```



Esto solo funciona con el front, pero el backend esta caido y hay que levantarla primero
hay que cambiar el yaml service del front y back por un LoadBalancer

```
! service.yaml ×
13-multi-tier > curso-de-docker-fundamentos > k8s > frontend > ! service.yaml
  4   name: frontend-service
  5   spec:
  6     selector:
  7       app: frontend
  8     ports:
  9       - protocol: TCP
 10         port: 80
 11         targetPort: 80
 12       type: LoadBalancer
```

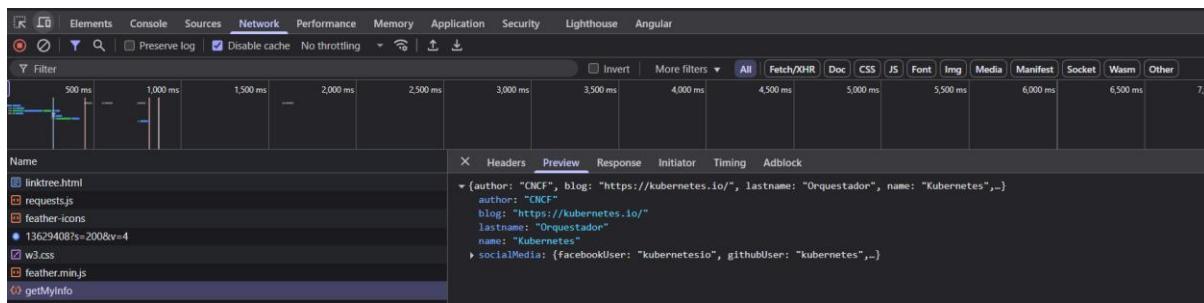
```
! service.yaml ×
13-multi-tier > curso-de-docker-fundamentos > k8s > backend > ! service.yaml
  3   metadata:
  4   spec:
  5     selector:
  6       app: backend
  7     ports:
  8       - protocol: TCP
  9         port: 5001
 10         targetPort: 5001
 11       type: LoadBalancer
```

Luego realizamos el apply nuevamente y verificamos que sean de loadbalancer

```
r-fundamentos> k get service
NAME           TYPE      CLUSTER-IP    EXTERNAL-IP    PORT(S)        AGE
backend-service  LoadBalancer  10.108.230.15  <pending>    5001:32411/TCP  23m
frontend-service  LoadBalancer  10.106.64.134  <pending>    80:32181/TCP  23m
```

Ahora hacemos el tunnel

minikube tunnel



Jobs y CronJobs

Para este ejercicio primero vamos a levantar una base de datos de MySQL para realizar backups continuamente, para esto vamos a seguir los pasos del archivo readme.

Para esto se utilizara el secret donde estas las credenciales de la BD y un StafulSet para persistir los datos y a la vez creando un volumen.

Recordar que los StatefulSet trabajan con volumenes (no siempre pero es lo recomendable), sino usara volumen entonces seria mejor un Deployment.

```
kubectl apply -f mysql-secret.yaml  
kubectl apply -f mysql-sts.yaml  
kubectl get pods -w -l app=mysql  
kubectl get sts mysql  
kubectl get pvc -l app=mysql  
kubectl get svc mysql-service-sts
```

Verificar la base de datos

```
kubectl exec -it mysql-0 -- mysql -u root -p
```

Password: rootpassword

Procedemos a crear la tabla

```
use mydb;  
CREATE TABLE testdb (id INT, name VARCHAR(255));  
SHOW TABLES;
```

Archivo yaml de backup: Job

```
apiVersion: batch/v1  
kind: Job #Nuevo tipo  
metadata:
```

```

name: mysql-backup
spec:
  template:
    spec:
      containers:
        - name: mysql-backup
          image: mysql:8.0
          command:
            - /bin/bash
            - -c
            - |
              mysqldump -h mysql-service-sts -u root -p$(MYSQL_ROOT_PASSWORD)
$(MYSQL_DATABASE) > /backup/mysql-backup-$(date +%Y%m%d).sql #Comando para hacer
un backup con mysql en una carpeta
          env: #Variables de BD encriptadas por el secret.yaml
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: root-password
            - name: MYSQL_DATABASE
              valueFrom:
                secretKeyRef:
                  name: mysql-secret
                  key: database
          volumeMounts: #Referencia de volumenes
            - name: backup-storage
              mountPath: /backup
      volumes:
        - name: backup-storage
          persistentVolumeClaim:
            claimName: mysql-backup-pvc
      restartPolicy: Never
      backoffLimit: 4

---
# create a pvc to claim the pv
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-backup-pvc
spec:

```

```

accessModes:
  - ReadWriteOnce
storageClassName: manual
selector:
  matchLabels:
    app: mysql-backup
    purpose: backup
resources:
  requests:
    storage: 5Gi

---
# create a pv to store the backup
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-backup-pv
  labels:
    app: mysql-backup
    purpose: backup
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: manual
  hostPath:
    path: /mnt/mysql-backup
    type: DirectoryOrCreate

```

Ahora ejecutamos el job

```

kubectl apply -f backup-job.yaml
kubectl get jobs mysql-backup
kubectl get pods -l job-name=mysql-backup

```

Ver backups logs:

```

kubectl logs -l job-name=mysql-backup

```

Verificar la existencia de los backups

```
minikube ssh  
cd /mnt/mysql-backup/  
ll
```

```
docker@minikube:/mnt/mysql-backup$ ll  
total 12  
drwxr-xr-x 2 root root 4096 Sep 11 19:33 ./  
drwxr-xr-x 1 root root 4096 Sep 11 19:33 ../  
-rw-r--r-- 1 root root 1832 Sep 11 19:33 mysql-backup-20250911.sql
```

Este job corre solo una vez, para correr multiples veces necesitamos crear un CronJob, este no puede reutilizar los comando anteriores, tiene que recrearse usando un nuevo template, si se quiere reutilizar o hacer alguna practica mas algoritmica o programatica, se debe usar helm, donde tiene la posibilidad de usar plantillas, condicionales, bucles, etc

```
apiVersion: batch/v1  
kind: CronJob #Nuevo tipo  
metadata:  
  name: mysql-backup  
spec: #A diferencia del anterior este tiene un scheduler y después el resto es igual.  
  schedule: "*/1 * * * *"  
  concurrencyPolicy: Forbid # Don't start new job if previous is still running  
  successfulJobsHistoryLimit: 3  
  failedJobsHistoryLimit: 1  
  jobTemplate:  
    spec:  
      template:  
        spec:  
          containers:  
            - name: mysql-backup  
              image: mysql:8.0  
              command:  
                - /bin/bash  
                - -c  
                - |  
                  mysqldump -h mysql-service-sts -u root -p$(MYSQL_ROOT_PASSWORD)  
$(MYSQL_DATABASE) > /backup/mysql-backup-$(date +%Y%m%d_%H%M).sql  
      env:  
        - name: MYSQL_ROOT_PASSWORD  
          valueFrom:
```

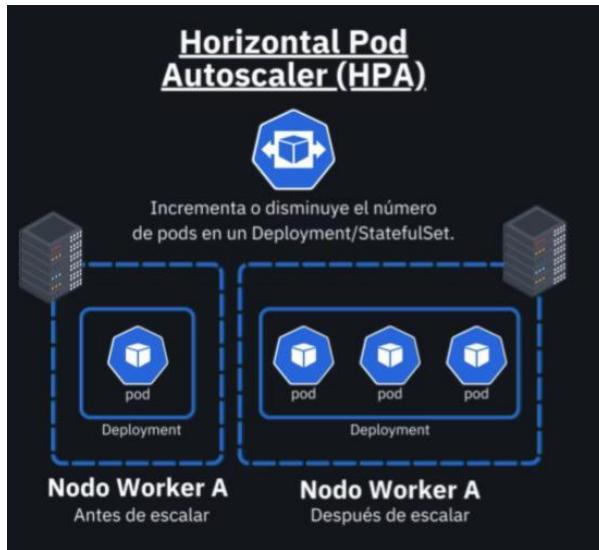
```
    secretKeyRef:
      name: mysql-secret
      key: root-password
    - name: MYSQL_DATABASE
      valueFrom:
        secretKeyRef:
          name: mysql-secret
          key: database
    volumeMounts:
    - name: backup-storage
      mountPath: /backup
  volumes:
  - name: backup-storage
  persistentVolumeClaim:
    claimName: mysql-backup-pvc
  restartPolicy: Never
```

```
---  
# create a pvc to claim the pv  
apiVersion: v1  
kind: PersistentVolumeClaim  
metadata:  
  name: mysql-backup-pvc  
spec:  
  accessModes:  
    - ReadWriteOnce  
  storageClassName: manual  
  selector:  
    matchLabels:  
      app: mysql-backup  
      purpose: backup  
  resources:  
    requests:  
      storage: 5Gi
```

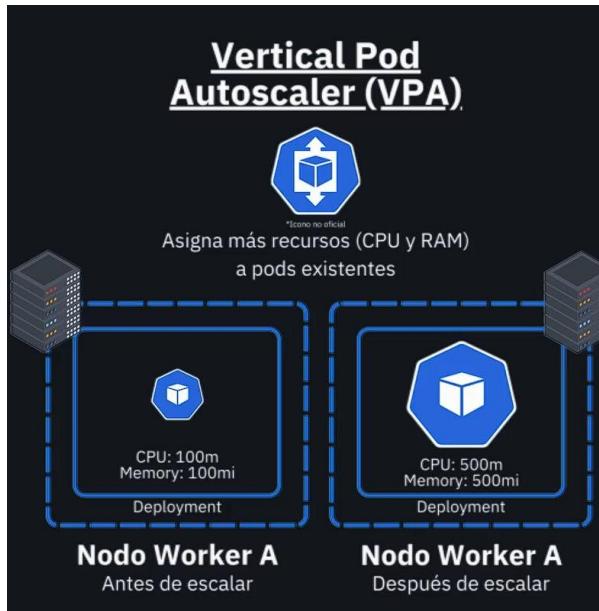


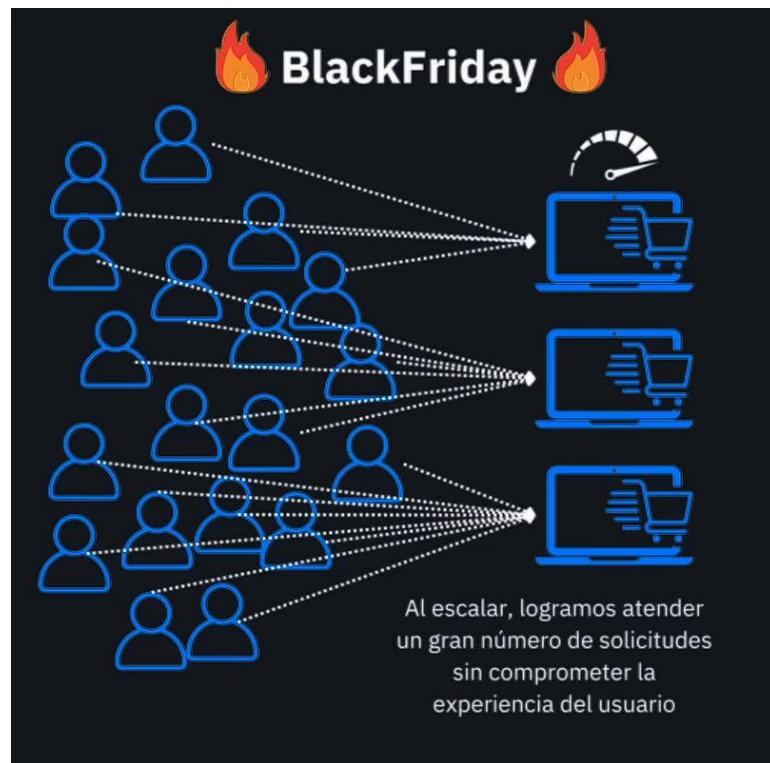
Escalado de aplicaciones: HPA y VPA

Escalado Horizontal



Escalado Vertical





Archivo yaml de HPA

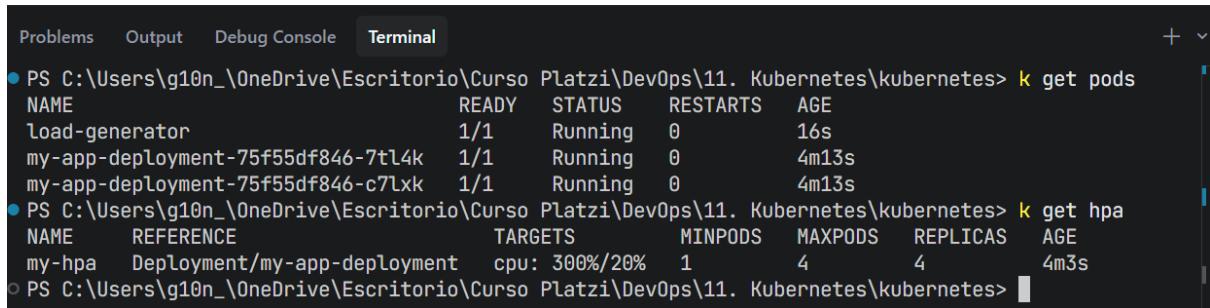
```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler #Nuevo tipo
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app-deployment
  minReplicas: 1
  maxReplicas: 4
  metrics:
  - type: Resource
    resource:
      name: cpu
    target:
      type: Utilization
      averageUtilization: 20 #Al superar el 20% va a aumentar las replicas
```

Ver los hpa

```
Normal Started 5s kubelet          Started container nginx
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\15-autoscaling-hpa-vpa> k get hpa
NAME      REFERENCE          TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
my-hpa    Deployment/my-app-deployment  cpu: 0%/20%  1          4          2          70s
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\15-autoscaling-hpa-vpa> k get pods
NAME                  READY   STATUS    RESTARTS   AGE
my-app-deployment-75f55df846-7tl4k  1/1    Running   0          112s
my-app-deployment-75f55df846-c7lxk  1/1    Running   0          112s
```

Ejecutar comando para estresar la aplicación y asi pueda levantar más pods:

```
kubectl run -i --tty load-generator --rm --image=busybox --restart=Never -- /bin/sh -c
"while sleep 0.001; do wget -q -O- http://my-app-service; done"
```

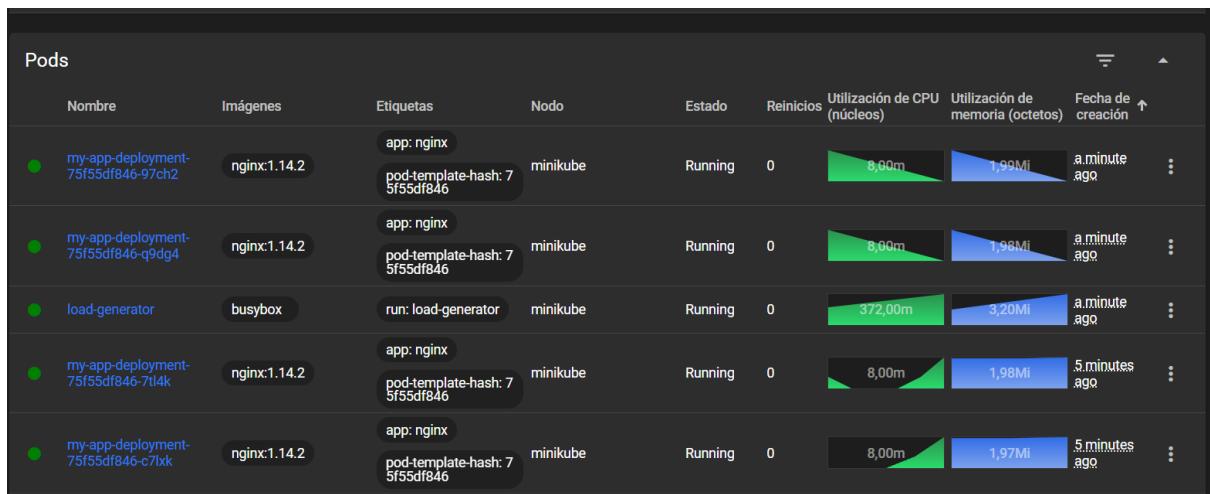


The screenshot shows a terminal window with the following history:

- PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> k get pods
- NAME READY STATUS RESTARTS AGE
- Load-generator 1/1 Running 0 16s
- my-app-deployment-75f55df846-7tl4k 1/1 Running 0 4m13s
- my-app-deployment-75f55df846-c7lxk 1/1 Running 0 4m13s
- PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> k get hpa
- NAME REFERENCE TARGETS MINPODS MAXPODS REPLICAS AGE
- my-hpa Deployment/my-app-deployment cpu: 300%/20% 1 4 4 4m3s
- PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes>

Tambien se puede ver desde el dashboard

minikube dashboard



Para instalar el VPA, es necesario seguir los pasos de este enlace

<https://github.com/kubernetes/autoscaler/blob/master/vertical-pod-autoscaler/docs/installation.md>

Paso 1:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/autoscaler/vpa-release-1.0/vertical-pod-autoscaler/deploy/vpa-v1-crd-gen.yaml  
kubectl apply -f https://raw.githubusercontent.com/kubernetes/autoscaler/vpa-release-1.0/vertical-pod-autoscaler/deploy/vpa-rbac.yaml
```

Paso 2:

```
git clone https://github.com/kubernetes/autoscaler.git
```

Paso 3:

```
cd ./autoscaler/vertical-pod-autoscaler/hack\
```

Paso 4:

Para esto es necesario abrir una terminal de git bash (no usar wsl), para instalar el vpa
bash vpa-up.sh

A partir de aquí ya se puede hacer el apply del vpa

```
k apply -f vpa.yaml
```

```
apiVersion: "autoscaling.k8s.io/v1"  
kind: VerticalPodAutoscaler #Nuevo tipo  
metadata:  
  name: my-vpa  
spec:  
  targetRef:  
    apiVersion: "apps/v1"  
    kind: Deployment  
    name: my-app-deployment  
  updatePolicy:  
    updateMode: "Auto"  
  resourcePolicy:  
    containerPolicies: #Se definen un número mínimo y máximo de recursos  
    - containerName: '*'  
      minAllowed: #minimo  
        cpu: 1m  
        memory: 4Mi  
      maxAllowed: #maximo  
        cpu: 8m  
        memory: 32Mi  
    controlledResources: ["cpu", "memory"]
```

Escalamiento de los pods

```
Image:          nginx:1.14.2
Image ID:       docker-pullable://nginx@sha256:f7988fb6c02e0ce69257d9bd9c
Port:           80/TCP
Host Port:     0/TCP
State:          Running
Started:       Wed, 23 Apr 2025 09:59:56 -0500
Ready:          True
Restart Count:  0
Limits:
  cpu:    8m
  memory: 64Mi
Requests:
  cpu:    8m
  memory: 32Mi
Environment:   <none>
Mounts:
```



Azure Kubernetes Service (AKS)

Primero instalar la cli de aks

```
az aks install-cli
```

Crear el grupo de recurso

```
az group create --name k8scourse-aks-demo --location eastus
```

Activar servicios para kubernetes

```
az provider register --namespace Microsoft.insights
az provider register --namespace Microsoft.ContainerService
```

Crear el cluster

```
az aks create --resource-group k8scourse-aks-demo --name k8sCourseDemoAKS --
node-count 3 --enable-addons monitoring --generate-ssh-keys --node-vm-size
Standard_D2s_v3
```

Conectarse al cluster

```
az aks get-credentials --resource-group k8scourse-aks-demo --name
k8sCourseDemoAKS
```

Configurar contexto de k8s

```
k config get-contexts
```

```
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes> k get nodes
  NAME           STATUS  ROLES   AGE      VERSION
  aks-nodepool1-35742505-vmss000000  Ready   <none>  4m29s  v1.32.6
  aks-nodepool1-35742505-vmss000001  Ready   <none>  4m26s  v1.32.6
  aks-nodepool1-35742505-vmss000002  Ready   <none>  4m35s  v1.32.6
  PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes>
```

Crear nuevo namespace

```
k create ns platzi-aks
```

Subir imagen a Azure Container Registry (ACR)

1. Autenticación

```
az login
```

```
az acr login --name apphubregistry
```

2. Etiquetar la imagen local

```
docker tag <nombre_imagen_local>:<tag_local>
apphubregistry.azurecr.io/<nombre_imagen>:<tag>
```

3. Subir la imagen

```
docker push apphubregistry.azurecr.io/<nombre_imagen>:<tag>
```

Base de datos Mysql

<https://mail.google.com/mail/u/2/#inbox/1MfcgzQcpnRqqqmKPByCVtWINGxVMVjx>

Aplicación real con AKS

Crear los namespaces.

```
k create namespace frontend
```

```
k create namespace backend
```

```
k create namespace storage
```

Crear external name para la conexión a la base de datos MySQL

```
kubectl -n storage create service externalname mysql --external-name
sql7.freesqldatabase.com
```

Crear los secretos para el backend

```
kubectl -n backend create secret generic mysql --from-
literal=DB_PASSWORD=lG2cuBA9mA --from-literal=DB_USER=sql7799390 --from-
literal=DB_HOST=sql7.freesqldatabase.com --from-literal=DB_NAME=sql7799390
```

Ver todos los recursos de los namespaces

```
k get all -A
```

Ejecutar un job para crear las tablas en mysql usando el externalname y los secretos antes creados, lo intentara hasta 4 veces según el config del yaml

```
k -n backend apply -f app/k8s/db-job-init/config.yaml # script  
k -n backend apply -f app/k8s/db-job-init/init-job.yaml #job
```

Ejecutar el script para subirlo al registry (se cambio el original para que sea en powershell y para apuntar azure)

```
cd 20-aws-eks-deploy\app\backend; .\backendbuild.ps1
```

Ejecutar los yamls para crear los pods del backend

```
k apply -f .\app\k8s\backend\ -n backend
```

Si AKS no tiene los permisos, hay que ejecutar los siguientes pasos:

- Obtener el ID del recurso de ACR:

```
az acr show --name <tu-nombre-acr> --query id --output tsv
```

- Obtener el ID de la identidad administrada de AK

```
az aks show --name <tu-nombre-cluster-aks> --resource-group <tu-grupo-recursos> --query identity.principalId --output tsv
```

- Asignar el rol AcrPull a la identidad administrada de AKS:

```
az role assignment create --assignee <AKS-Managed-Identity-ID> --role AcrPull --scope <ACR-Resource-ID>
```

- Forma recomendada de integrar AKS con ACR:

```
az aks update -n k8sCourseDemoAKS -g aks-group-test --attach-acr apphubregistry
```

- Aplicar el archivo de despliegue:

```
kubectl apply -f <ruta-a-deployment.yaml>
```

Ver el externalIP:

```
k get svc -n backend
```

```
oy> k get svc -n backend  
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP     PORT(S)        AGE  
backend-service LoadBalancer 10.0.139.212 20.246.181.182  5001:31236/TCP  12m  
o PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\20-aws-  
sv>
```



Ahora es necesario cambiar la URL del front

```
4  };
5
6- fetch('http://a809588120a774135ab2a784ee84b746-1942087673.us-east-1.elb.amazonaws.com/getMyInfo', requestOptions)
6+ fetch('http://20.246.181.182:5001/getMyInfo', requestOptions)
7  .then(res => {
8    if (!res.ok) {
```

Ejecutar script para subir imagen del front (cambiado como el anterior)

```
cd .\app\frontend\; .\frontendbuild.ps1
```

Ejecutar yaml's del front

```
k apply -f .\app\k8s\frontend\ -n frontend
```

Ver todos los servicios

```
k get svc -A
```

```
PS C:\Users\g10n_\OneDrive\Escritorio\Curso Platzi\DevOps\11. Kubernetes\kubernetes\20-aws-eks-deploy> k get svc -A
NAMESPACE      NAME        TYPE        CLUSTER-IP      EXTERNAL-IP      PORT(S)        AGE
backend        backend-service   LoadBalancer  10.0.139.212  20.246.181.182  5001:31236/TCP  30m
default        kubernetes     ClusterIP    10.0.0.1       <none>          443/TCP       106m
frontend        frontend-service   LoadBalancer  10.0.218.69   48.223.229.155  80:30735/TCP  61s
kube-system    kube-dns       ClusterIP    10.0.0.10      <none>          53/UDP,53/TCP  106m
kube-system    metrics-server   ClusterIP    10.0.236.57   <none>          443/TCP       106m
storage        mysql         ExternalName  <none>          sql7.freessqldatabase.com  <none>          93m
```



Validar que se conecta a nuestro backend

Name	Headers	Preview	Response	Initiator	Timing	Adblock
getMyInfo			{author: "CNCF", blog: "https://kubernetes.io/", lastname: "Orquestador", name: "Kubernetes", ...}			

```
{author: "CNCF", blog: "https://kubernetes.io/", lastname: "Orquestador", name: "Kubernetes", ...}
  ↴ socialMedia: {facebookUser: "kubernetesio", githubUser: "kubernetes", ...}
```

Desplegar modelos de IA en kubernetes



La vista a 10K pies de altura

Control Plane

- API Server: El frontend del control plane
- Fuente única de la verdad (etcd)
- Revisar por nuevos pods y asignarlos a un nodo (kube-scheduler)

Controladores

- Responden a cambios en los objetos
- Mover estado actual a estado deseado

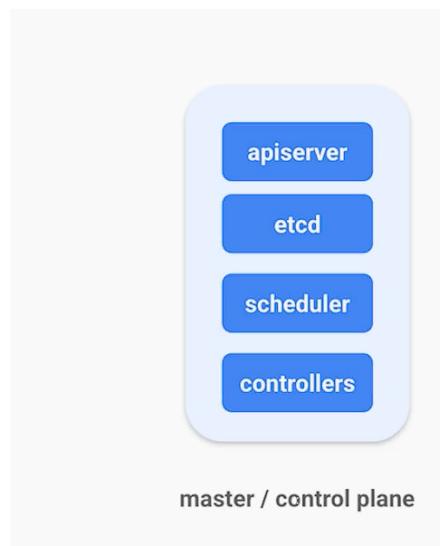
Todo es un recurso (objeto)

- kind (tipo)
- apiVersion
- metadata
- spec ← representación del estado deseado
- status ← representación del estado actual

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: nginx
  labels:
    app: nginx
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
```

Componentes del Master

- ▶ API server (kube-apiserver)
- ▶ etcd — key-value store distribuida y confiable
- ▶ Calendarizador (kube-scheduler)
- ▶ Controladores
 - Kube controller (kube-controller-manager)
 - Controlador de Replicación
 - Controlador de endpoints
 - Controlador de tokens y cuentas de servicio
 - Cloud controller (cloud-controller-manager)
- ▶ Add-ons
 - Kube DNS (kube-dns)
 - Web UI (dashboard)
 - Monitoreo
 - Logueo en todo el cluster

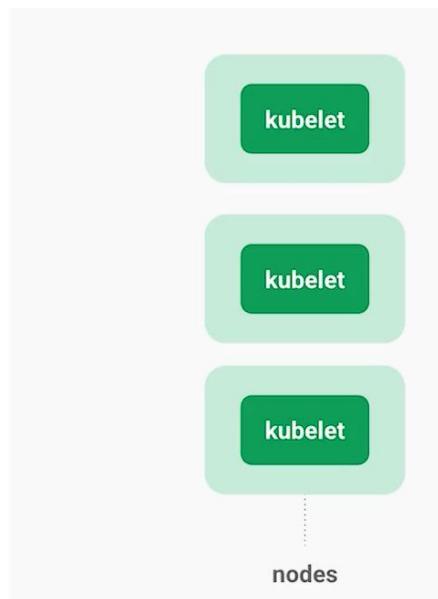


Componentes de los Nodos

- ▶ Kubelet (kubelet)
- ▶ Kube proxy (kube-proxy)
- ▶ Container runtime
 - Docker (containerd)
 - Rkt
- ▶ Monitoring/Logging
 - Fluentd

Los clientes usan la línea de comandos (CLI) de Kube Control (kubectl) para interactuar con el clúster

<https://kubernetes.io/docs/concepts/overview/components/>



Concepto clave: El Pod

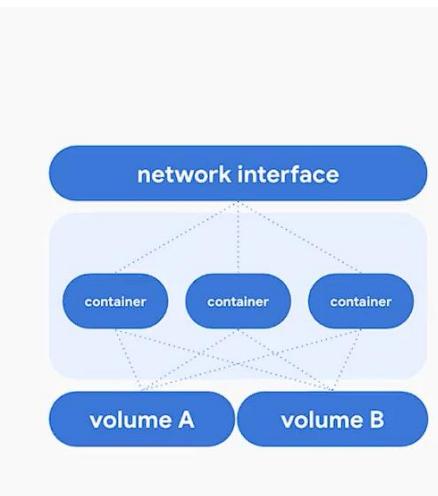
Unidad atómica de Kubernetes

Compuesto de uno o más contenedores con recursos de almacenamiento y red compartidos (e.j. servidor de apps y web server)

Los contenedores dentro de un pod comparten el mismo linux namespace, pero diferentes grupos de control

Kubernetes automatiza de buena manera la configuración de namespaces, cgroups, etc.

Unidad de despliegue (empaquetada)



El ciclo de vida de un Pod

