



kubernetes

KUBERNETES

INTRODUCCIÓN A KUBERNETES

1

- Web:
 - <https://kubernetes.io>
- Documentación oficial:
 - <https://kubernetes.io/docs/home/>
- Instalación de herramientas:
 - <https://kubernetes.io/docs/tasks/tools/>

KUBERNETES



INTRODUCCIÓN

■ ¿Qué es Kubernetes?

- **Kubernetes** (a menudo abreviado como K8s) es una plataforma de orquestación de contenedores de código abierto, desarrollada originalmente por Google y ahora mantenida por la Cloud Native Computing Foundation (CNCF).
- Su propósito principal es automatizar el despliegue, escalado, y gestión de aplicaciones en contenedores.

■ ¿Qué hace Kubernetes?

- **Orquestación de contenedores:** Gestiona múltiples contenedores distribuidos a través de varios servidores (nodos).
- **Escalado automático:** Puede aumentar o reducir el número de instancias de tu aplicación según la demanda.
- **Autorecuperación:** Si un contenedor falla, Kubernetes lo reinicia o lo reemplaza automáticamente.
- **Despliegues continuos:** Permite hacer actualizaciones sin tiempos de inactividad mediante estrategias como rolling updates.
- **Balanceo de carga y descubrimiento de servicios:** Distribuye el tráfico entre los contenedores y permite que se comuniquen entre sí.
- **Gestión de configuración y secretos:** Permite separar la configuración de la lógica de la aplicación.

■ Arquitectura:

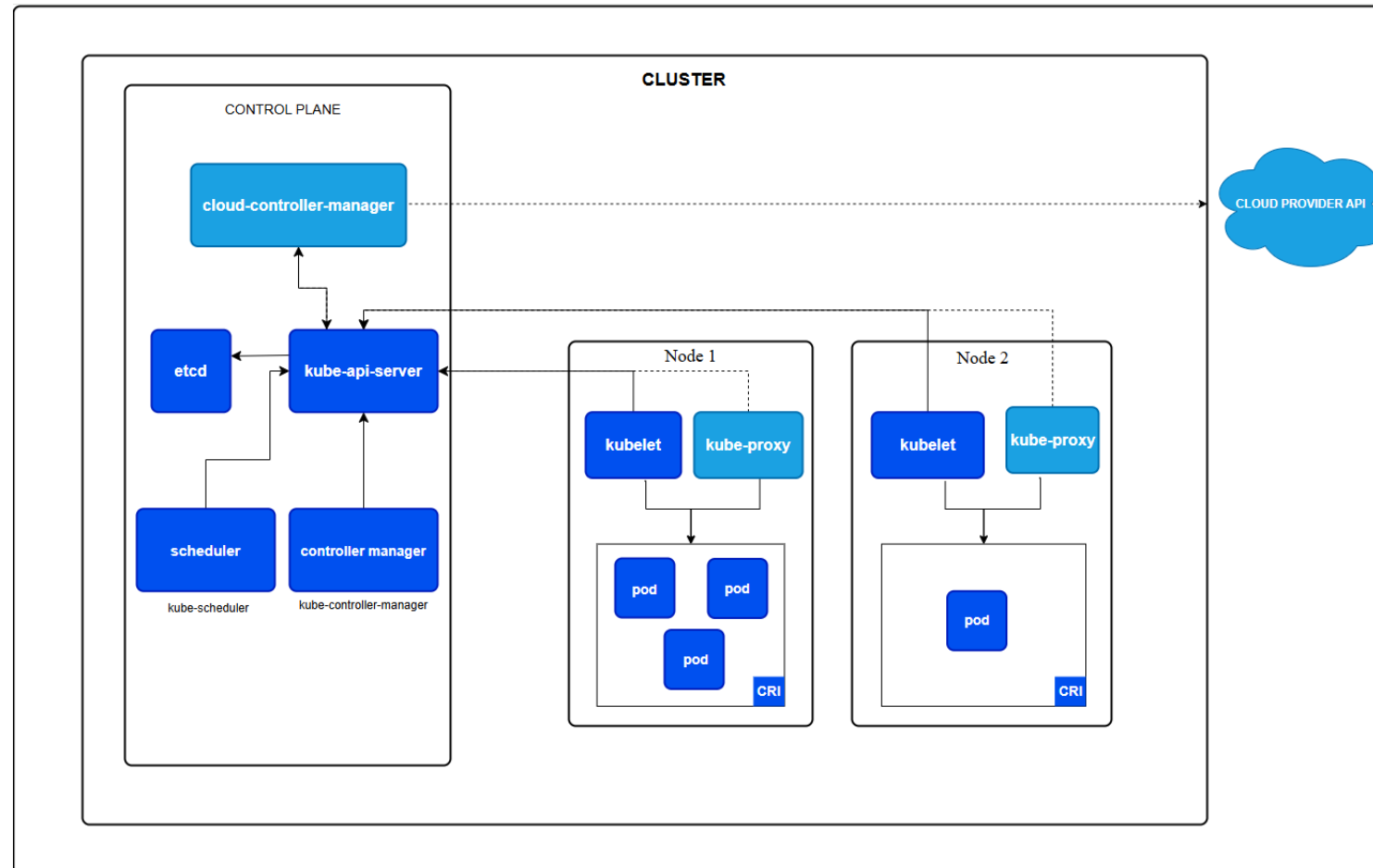
- La **arquitectura de Kubernetes** se basa en un **modelo cliente-servidor distribuido** compuesto por varios componentes que trabajan juntos para gestionar clústeres de contenedores. Está dividida principalmente en dos partes:
 - **Plano de control (Control Plane)**. Encargado de gestionar el estado del clúster y tomar decisiones globales (como programar pods, escalar aplicaciones, etc.).
 - Componentes:
 - **kube-apiserver** → Punto de entrada para todas las peticiones (CLI, UI, API). Expone la API de Kubernetes.
 - **etcd** → Base de datos distribuida donde se guarda todo el estado del clúster (clave-valor).
 - **kube-scheduler** → Decide en qué nodo correr cada pod nuevo (basado en recursos, afinidad, etc.).
 - **kube-controller-manager** → Ejecuta los controladores que vigilan el estado del clúster y reaccionan.
 - **cloud-controller-manager** → (Opcional) Interactúa con servicios de nube (balanceadores, volúmenes, etc.). Equivalente a kube-controller-manager en infraestructura en la nube.
 - **Nodos (Worker Nodes)**. Ejecutan las aplicaciones en contenedores. Cada nodo tiene los componentes necesarios para correr pods y comunicarse con el plano de control.

■ Arquitectura:

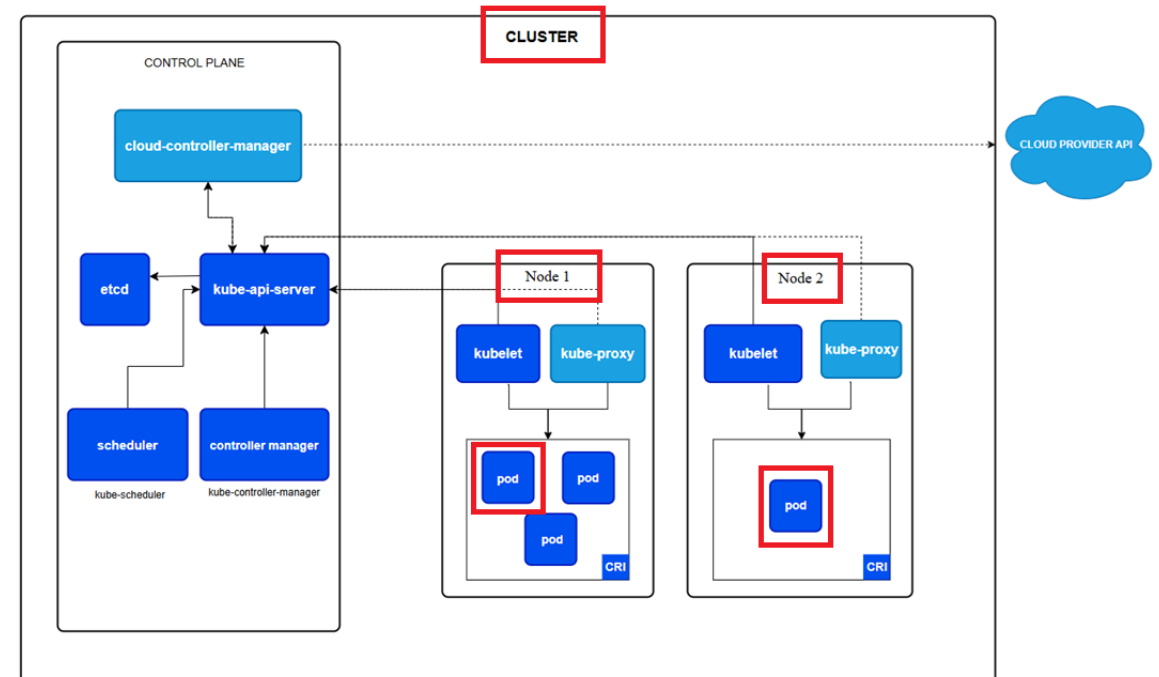
- La **arquitectura de Kubernetes** se basa en un **modelo cliente-servidor distribuido** compuesto por varios componentes que trabajan juntos para gestionar clústeres de contenedores. Está dividida principalmente en dos partes:
 - **Plano de control (Control Plane)**. Encargado de gestionar el estado del clúster y tomar decisiones globales (como programar pods, escalar aplicaciones, etc.).
 - **Nodos (Worker Nodes)**. Ejecutan las aplicaciones en contenedores. Cada nodo tiene los componentes necesarios para correr pods y comunicarse con el plano de control.
 - Componentes:
 - **kubelet** → Agente que corre en cada nodo y garantiza que los contenedores estén ejecutándose.
 - **kube-proxy** → Administra el enrutamiento de red y balanceo de carga para los servicios.
 - **Container Runtime Interface (CRI)** → Ejecuta los contenedores (por ejemplo: containerd, Docker, CRI-O).
 - **Pods**. La unidad mínima de ejecución en Kubernetes. Puede contener uno o varios contenedores. Normalmente un contenedor por pod. La IP la tiene el pod, no el container. Kubernetes gestiona pods, no contenedores.

- Arquitectura:

<https://kubernetes.io/docs/concepts/architecture/>



- Jerarquía básica en Kubernetes:
 - **Cluster:** Un conjunto de nodos gestionados por Kubernetes.
 - **Node** (nodo): Una máquina (física o virtual) que ejecuta Pods.
 - **Pod:** La unidad mínima de ejecución en Kubernetes. Puede contener uno o varios contenedores.



■ **Relación de conceptos:**

- Pod → Contenedor(es) que ejecutan tu aplicación
- Nodo → Máquina (física o virtual) que ejecuta Pods
- Clúster → Grupo de nodos gestionados por Kubernetes
- Service → Objeto lógico que enruta tráfico a Pods (no es una máquina)
- Deployment → Objeto que gestiona automáticamente la creación y escalado de Pods.
- Namespace → Espacio lógico para agrupar recursos dentro del clúster (como proyectos).
- Volume → Almacenamiento persistente para Pods (disco, red, etc.).
- ConfigMap / Secret → Almacenan configuración y secretos (claves, tokens, etc.) separados del código.

KUBERNETES



HOLA MUNDO

- <https://kubernetes.io/docs/tutorials/hello-minikube/>
- **kubectl:**
 - Es una herramienta de línea de comandos de Kubernetes para desplegar y gestionar aplicaciones en Kubernetes. Usando kubectl, puedes inspeccionar recursos del clúster; crear, eliminar, y actualizar componentes; explorar tu nuevo clúster y arrancar aplicaciones.
 - ¡Ya está instalado con Docker Desktop!
 - **COMPROBACIÓN: kubectl version --client**
 - Instalación: <https://kubernetes.io/docs/tasks/tools/>

■ **minikube:**

- Minikube es una herramienta que permite ejecutar un clúster de Kubernetes localmente en una sola máquina (una computadora personal o de desarrollo).
- Es ideal para aprender, experimentar o desarrollar aplicaciones que luego se desplegarán en un clúster real más grande.
- Características principales de Minikube:
 - Clúster de un solo nodo: Crea un único nodo maestro/worker que simula un clúster Kubernetes completo.
 - Multiplataforma: Funciona en Windows, macOS y Linux.
 - Simula entornos de nube: Te permite probar configuraciones similares a las de producción sin necesidad de usar un proveedor en la nube.
 - Ideal para desarrollo y pruebas: Ligero y rápido de iniciar/detener.
 - Incluye herramientas útiles: Como dashboard, soporte para Ingress, LoadBalancer local, etc.
- Minikube crea una máquina virtual (o contenedor, usando drivers como Docker, Hyper-V, VirtualBox, etc.) donde se instala Kubernetes, permitiéndote interactuar con el clúster mediante **kubectl**.

■ minikube:

- Instalación: <https://minikube.sigs.k8s.io/docs/start>
- Cómmandos básicos:
 - `minikube start` → Inicia el clúster
 - `minikube stop` → Detiene el clúster
 - `minikube status` → Muestra el estado del clúster
 - `minikube dashboard` → Abre el dashboard web de Kubernetes
 - `minikube delete` → Elimina el clúster
 - `minikube profile list` → Muestra los cluster creados

■ minikube:

- Creación del cluster (Lo crea con Docker si está instalado):
 - **minikube start**

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.3775]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferna>minikube start
* minikube v1.35.0 en Microsoft Windows 11 Home 10.0.26100.3775 Build 26100.3775
* Controlador docker seleccionado automáticamente
* Using Docker Desktop driver with root privileges
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.46 ...
* Descargando Kubernetes v1.32.0 ...
  > preloaded-images-k8s-v18-v1...: 333.57 MiB / 333.57 MiB 100.00% 1.40 Mi
  > gcr.io/k8s-minikube/kicbase...: 500.31 MiB / 500.31 MiB 100.00% 1.14 Mi
* Creating docker container (CPUs=2, Memory=3900MB) ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Preparando Kubernetes v1.32.0 en Docker 27.4.1...
  - Generando certificados y llaves
  - Iniciando plano de control
  - Configurando reglas RBAC...
* Configurando CNI bridge CNI ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default

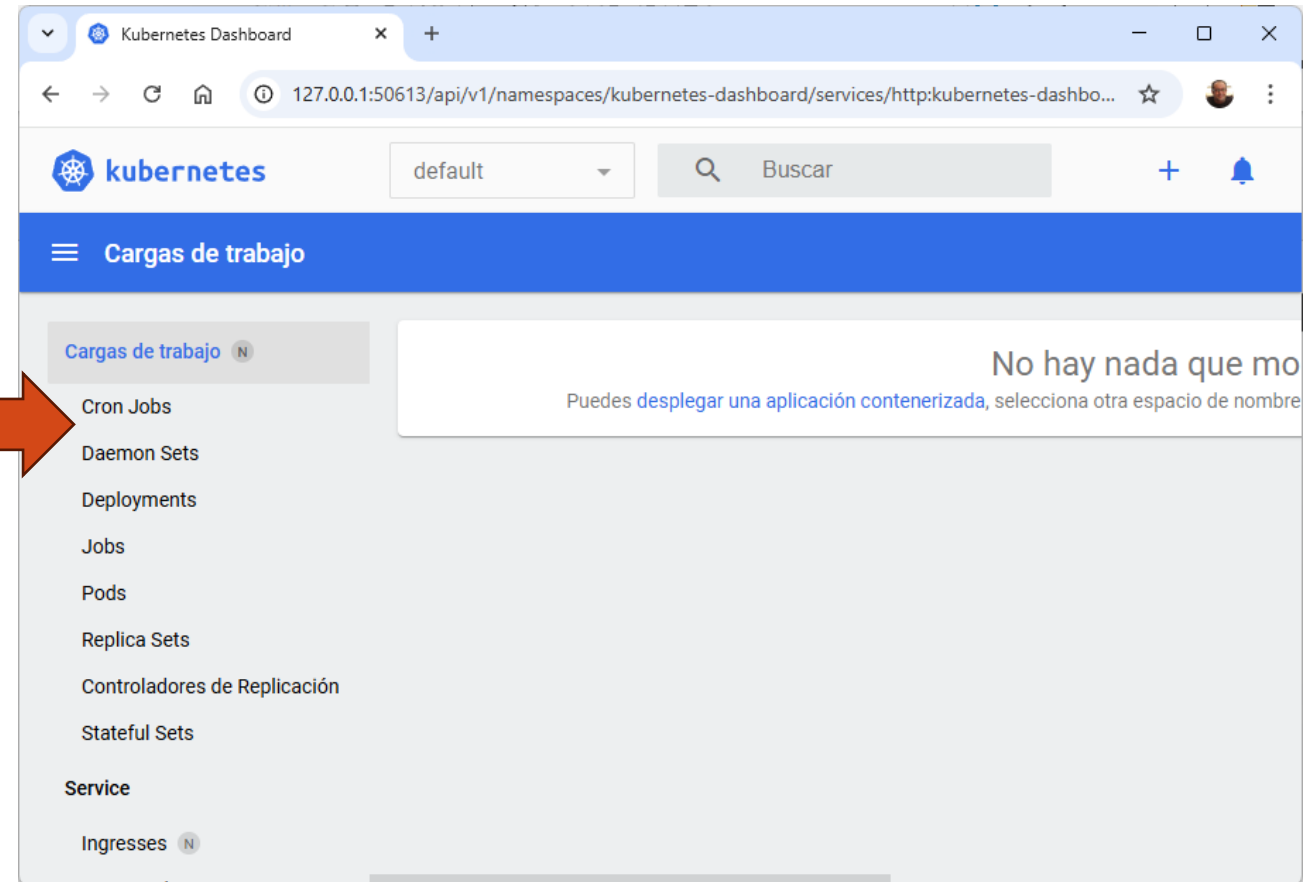
C:\Users\ferna>
```

KUBERNETES



- minikube:
 - Acceso al panel de control:
 - minikube dashboard

```
Símbolo del sistema - minikul x + v
C:\Users\ferna>minikube dashboard
* Verifying dashboard health ...
* Launching proxy ...
* Verifying proxy health ...
* Opening http://127.0.0.1:50613/api/v1/namespaces/kubernetes-dashboard/services/http:kubernetes-dashboard:/proxy/ in your default browser...
```



■ minikube:

- Consulta de los clusters creados:

- minikube profile list

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.3775]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferna>minikube profile list
```

Profile	VM Driver	Runtime	IP	Port	Version	Status	Nodes	Active Profile	Active Kubecontext
minikube	docker	docker	192.168.49.2	8443	v1.32.0	OK	1	*	*

```
C:\Users\ferna>
```


- minikube:
 - Creación de nuevo cluster:
 - `minikube start -p nombre-cluster`

```
Símbolo del sistema
Microsoft Windows [Versión 10.0.26100.3775]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferna>minikube start -p dev-cluster
* [dev-cluster] minikube v1.35.0 en Microsoft Windows 11 Home 10.0.26100.3775 Build 26100.3775
* Controlador docker seleccionado automáticamente
* Using Docker Desktop driver with root privileges
* Starting "dev-cluster" primary control-plane node in "dev-cluster" cluster
* Pulling base image v0.0.46 ...
* Creating docker container (CPUs=2, Memory=3900MB) ...
! Failing to connect to https://registry.k8s.io/ from inside the minikube container
* To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
* Preparando Kubernetes v1.32.0 en Docker 27.4.1...
  - Generando certificados y llaves
  - Iniciando plano de control
  - Configurando reglas RBAC...
* Configurando CNI bridge CNI ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Complementos habilitados: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "dev-cluster" cluster and "default" namespace by default

C:\Users\ferna>minikube profile-list
Error: unknown command "profile-list" for "minikube"
Run 'minikube --help' for usage.

C:\Users\ferna>minikube profile list
```

Profile	VM Driver	Runtime	IP	Port	Version	Status	Nodes	Active Profile	Active Kubecontext
dev-cluster	docker	docker	192.168.58.2	8443	v1.32.0	OK	1		*
minikube	docker	docker	192.168.49.2	8443	v1.32.0	OK	1	*	

```
C:\Users\ferna>
```



■ minikube:

- Detener un cluster (lo pone en estado **Stopped**):
 - `minikube stop -p nombre-cluster`

```
Símbolo del sistema × + v
```

Profile	VM Driver	Runtime	IP	Port	Version	Status	Nodes	Active Profile	Active Kubecontext
minikube	docker	docker	192.168.49.2	8443	v1.32.0	OK	1	*	*
nuevocluster	docker	docker	192.168.58.2	8443	v1.32.0	OK	1		

```
C:\Users\ferna>minikube stop -p nuevocluster
* Stopping node "nuevocluster" ...
* Apagando "nuevocluster" mediante SSH...
* 1 node stopped.

C:\Users\ferna>minikube profile list
```

Profile	VM Driver	Runtime	IP	Port	Version	Status	Nodes	Active Profile	Active Kubecontext
minikube	docker	docker	192.168.49.2	8443	v1.32.0	OK	1	*	*
nuevocluster	docker	docker	192.168.58.2	8443	v1.32.0	Stopped	1		

```
C:\Users\ferna>
```

KUBERNETES



kubernetes

- minikube:

- Eliminar un cluster:

- `minikube delete -p nombre-cluster`

```
Símbolo del sistema
C:\Users\ferna>minikube delete -p dev-cluster
* Eliminando "dev-cluster" en docker...
* Eliminando contenedor "dev-cluster" ...
* Eliminando C:\Users\ferna\.minikube\machines\dev-cluster...
* Removed all traces of the "dev-cluster" cluster.

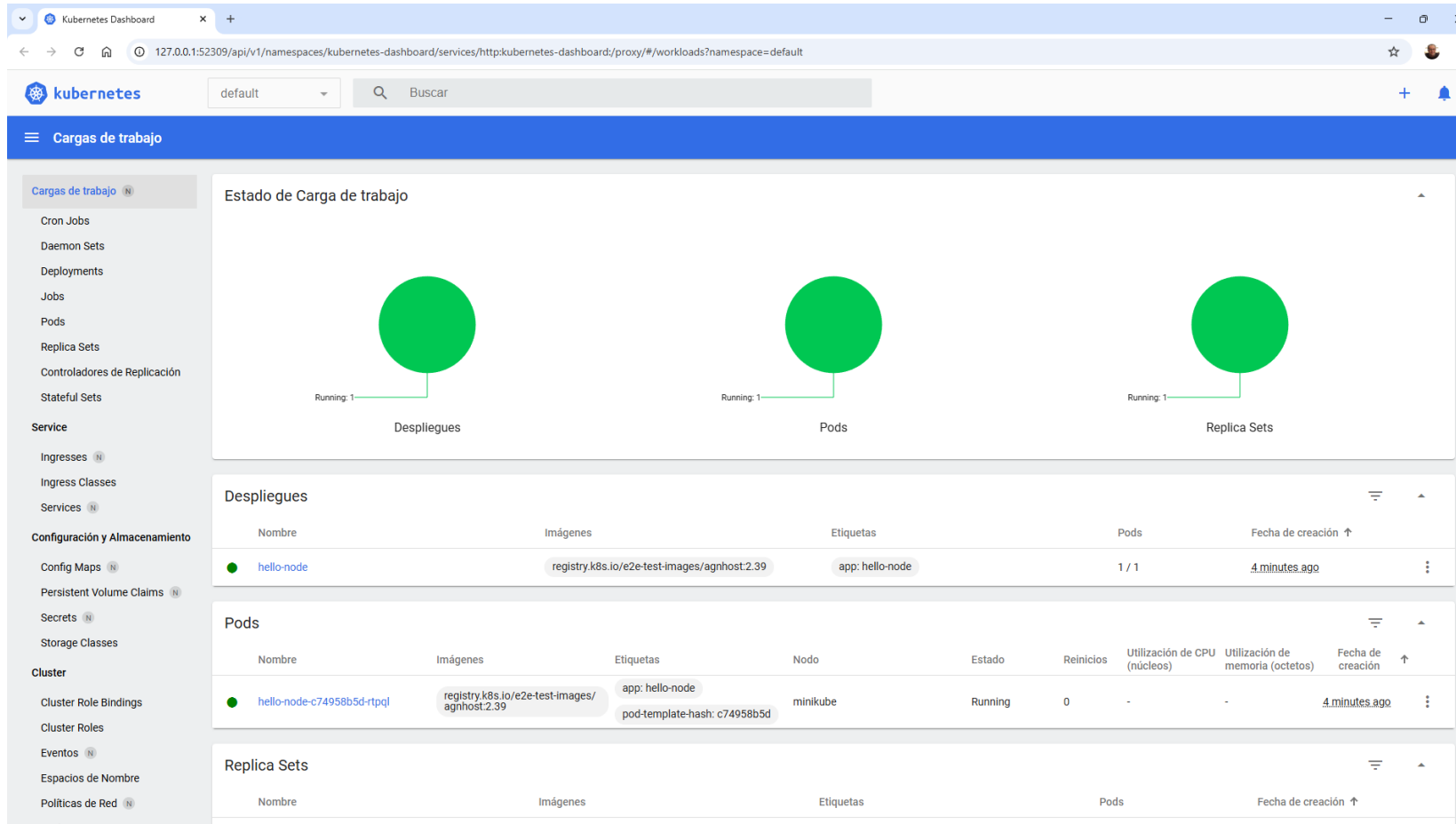
C:\Users\ferna>
```

■ kubectl:

■ Creación del **Deployment**:

- `kubectl create deployment hello-node --image=registry.k8s.io/e2e-test-images/agnhost:2.39 --/agnhost netexec --http-port=8080`
- Este comando lanza un deployment en Kubernetes que ejecuta un contenedor con la imagen `agnhost:2.39`, configurado para iniciar un servidor HTTP en el puerto 8080 usando el comando `netexec`.
- Desglose de la instrucción:
 - **kubectl create deployment hello-node:**
 - Crea un objeto de tipo Deployment en Kubernetes con el nombre `hello-node`.
 - **--image=registry.k8s.io/e2e-test-images/agnhost:2.39:**
 - Especifica la imagen del contenedor que se usará. Esta imagen (`agnhost`) es una herramienta de prueba utilizada por Kubernetes para simular distintos comportamientos de red y aplicaciones.
 - **-- /agnhost netexec --http-port=8080:**
 - Todo lo que aparece después de `--` se pasa como comando de entrada (entrypoint) al contenedor. Aquí se está ejecutando `/agnhost netexec`, que es un subcomando de la herramienta `agnhost`, el cual lanza un servidor que responde a conexiones de red.
 - **--http-port=8080:**
 - El servidor escucha en el puerto HTTP 8080.

■ minikube dashboard:



Cargas de trabajo

Estado de Carga de trabajo

Running: 1 Despliegues

Running: 1 Pods

Running: 1 Replica Sets

Despliegues

Nombre	Imágenes	Etiquetas	Pods	Fecha de creación ↑
hello-node	registry.k8s.io/e2e-test-images/agnhost:2.39	app: hello-node	1 / 1	4 minutes ago

Pods

Nombre	Imágenes	Etiquetas	Nodo	Estado	Reinicios	Utilización de CPU (núcleos)	Utilización de memoria (octetos)	Fecha de creación ↑
hello-node-c74958b5d-rtqgl	registry.k8s.io/e2e-test-images/agnhost:2.39	app: hello-node pod-template-hash: c74958b5d	minikube	Running	0	-	-	4 minutes ago

Replica Sets

Nombre	Imágenes	Etiquetas	Pods	Fecha de creación ↑
--------	----------	-----------	------	---------------------

KUBERNETES



kubernetes

■ kubectl:

■ Comandos:

- `kubectl get deployments` → Consulta de Deployments

```
Símbolo del sistema
C:\Users\ferna>kubectl get deployments
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
hello-node    1/1     1             1           8m25s
C:\Users\ferna>
```

- `kubectl get pods` → Consulta de pods

```
Símbolo del sistema
C:\Users\ferna>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-node-c74958b5d-rtpql         1/1     Running   0           9m52s
C:\Users\ferna>
```

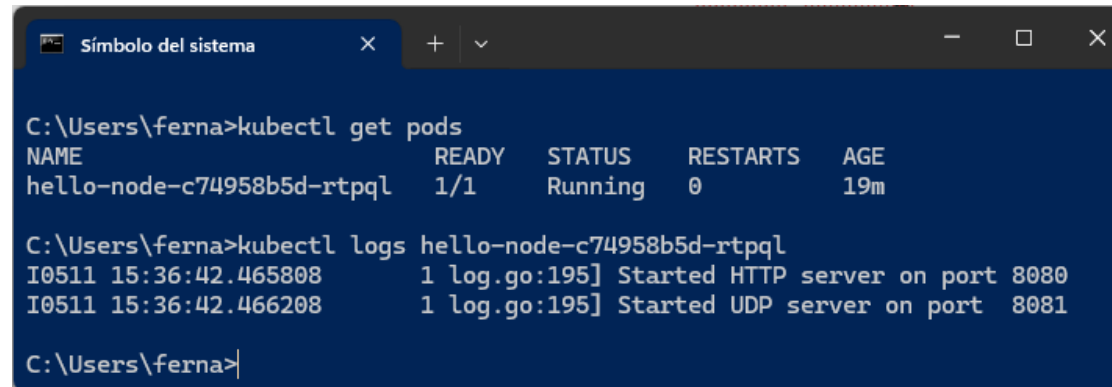
- `kubectl get events` → Consulta de eventos

```
Símbolo del sistema
C:\Users\ferna>kubectl get events
LAST SEEN   TYPE      REASON              OBJECT                               MESSAGE
10m         Normal    Scheduled            pod/hello-node-c74958b5d-rtpql       Successfully assigned default/hello-node-c74958b5d-rtpql to minikube
10m         Normal    Pulling              pod/hello-node-c74958b5d-rtpql       Pulling image "registry.k8s.io/e2e-test-images/agnhost:2.39"
8m42s       Normal    Pulled               pod/hello-node-c74958b5d-rtpql       Successfully pulled image "registry.k8s.io/e2e-test-images/agnhost:2.39" in 1m45.256s (1m45.256s including waiting). Image size: 126872991 bytes.
8m42s       Normal    Created              pod/hello-node-c74958b5d-rtpql       Created container: agnhost
8m42s       Normal    Started              pod/hello-node-c74958b5d-rtpql       Started container agnhost
10m         Normal    SuccessfulCreate     replicaset/hello-node-c74958b5d     Created pod: hello-node-c74958b5d-rtpql
10m         Normal    ScalingReplicaSet    deployment/hello-node                Scaled up replica set hello-node-c74958b5d from 0 to 1
```

■ kubectl:

■ Comandos:

- **kubectl config view** → Muestra el contenido del archivo de configuración de kubectl, que normalmente se encuentra en ~/.kube/config.
- **kubectl logs** → Muestra los logs del sistema.
- **Kubectl logs *id_pod*** → Muestra los logs del pod.

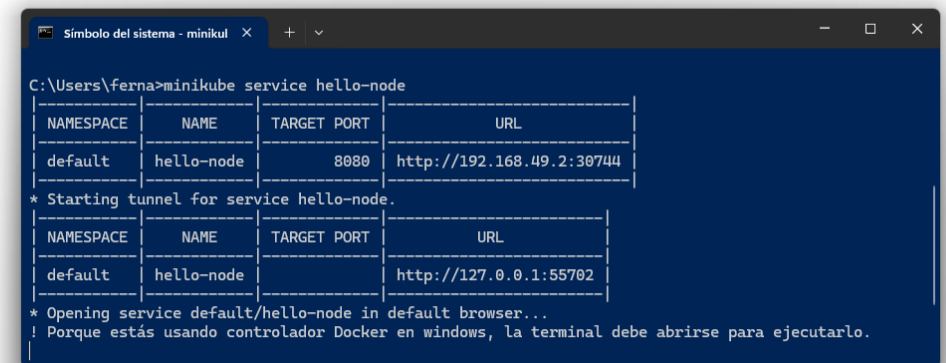
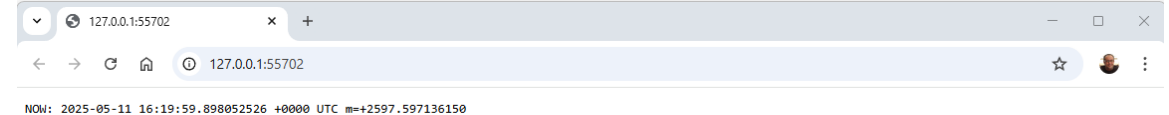


```
C:\Users\ferna>kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-node-c74958b5d-rtpql         1/1     Running   0           19m

C:\Users\ferna>kubectl logs hello-node-c74958b5d-rtpql
I0511 15:36:42.465808    1 log.go:195] Started HTTP server on port 8080
I0511 15:36:42.466208    1 log.go:195] Started UDP server on port 8081

C:\Users\ferna>
```

- En este punto, el Pod sólo es accesible desde dentro del cluster
→ para hacer accesible el contenedor desde el exterior se debe exponer el Pod como un **Service**.
- **kubectl:**
 - **kubectl expose** deployment hello-node --type=LoadBalancer --port=8080
 - **kubectl get services**
 - **minikube service hello-node**

A screenshot of a terminal window titled "Símbolo del sistema - minikul". The prompt is "C:\Users\ferna>minikube service hello-node". The output shows two tables. The first table lists the service details for 'hello-node' in the 'default' namespace, showing it is exposed on port 8080 with a target port of 8080 and a URL of 'http://192.168.49.2:30744'. The second table shows the tunnel details for the same service, with a target port of 8080 and a URL of 'http://127.0.0.1:55702'. Below the tables, there are instructions to start the tunnel and open the service in a default browser.

```
C:\Users\ferna>minikube service hello-node
```

NAMESPACE	NAME	TARGET PORT	URL
default	hello-node	8080	http://192.168.49.2:30744

```
* Starting tunnel for service hello-node.
```

NAMESPACE	NAME	TARGET PORT	URL
default	hello-node	8080	http://127.0.0.1:55702

```
* Opening service default/hello-node in default browser...
! Porque estás usando controlador Docker en windows, la terminal debe abrirse para ejecutarlo.
```


- Addons (extensiones):
 - Los **addons de Minikube** son **componentes o servicios adicionales** que puedes activar fácilmente en tu clúster local de Kubernetes para **agregar funcionalidades comunes** sin tener que configurarlas manualmente.
 - Ejemplos:
 - Interfaces gráficas (como el Dashboard)
 - Monitoreo y métricas (metrics-server)
 - Balanceadores de carga e Ingress controllers
 - DNS, registros de contenedores locales, certificados TLS, etc.
 - `minikube addons list` → Obtener el listado de extensiones disponibles.
 - `minikube addons enable <nombre-del-addon>` → Habilitar *addon*.
 - `minikube addons disable <nombre-del-addon>` → Deshabilitar el *addon*.

- Addons (extensiones) populares:
 - dashboard → UI web para visualizar y administrar el clúster
 - metrics-server → Recolecta métricas de CPU/RAM de los pods
 - ingress → Habilita un controlador Ingress (NGINX)
 - registry → Registro Docker local dentro de Minikube
 - default-storageclass → Provee almacenamiento dinámico local
 - volumesnapshots → Soporte para snapshots de volúmenes
- Ejemplo:
 - minikube addons enable dashboard
 - minikube dashboard

- Addons (extensiones) :
- Ejemplo **metrics-server**:
 - minikube addons enable metrics-server
 - `kubectl get pod,svc -n kube-system`

```
C:\Users\ferna>kubectl get pod,svc -n kube-system
```

NAME	READY	STATUS	RESTARTS	AGE
pod/coredns-668d6bf9bc-d6775	1/1	Running	0	30m
pod/etcd-minikube	1/1	Running	0	30m
pod/kube-apiserver-minikube	1/1	Running	0	30m
pod/kube-controller-manager-minikube	1/1	Running	0	30m
pod/kube-proxy-bwkqg	1/1	Running	0	30m
pod/kube-scheduler-minikube	1/1	Running	0	30m
pod/metrics-server-7fbb699795-5vkp5	1/1	Running	0	3m6s
pod/storage-provisioner	1/1	Running	1 (29m ago)	30m

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/kube-dns	ClusterIP	10.96.0.10	<none>	53/UDP,53/TCP,9153/TCP	30m
service/metrics-server	ClusterIP	10.109.131.165	<none>	443/TCP	3m6s

```
C:\Users\ferna>
```

- Addons (extensiones) :
- Ejemplo **metrics-server** (Información proporcionada):
 - Pods:
 - NAME → Nombre único del pod
 - READY → Número de contenedores listos vs. totales en el pod (ej. 1/1 = OK)
 - STATUS → Estado actual del pod (Running, CrashLoopBackOff, Pending, etc.)
 - RESTARTS → Cuántas veces se ha reiniciado ese pod (por fallos u otros motivos)
 - AGE → Tiempo desde que fue creado
 - Services:
 - NAME → Nombre del servicio
 - TYPE → Tipo de servicio: ClusterIP, NodePort, LoadBalancer, etc.
 - CLUSTER-IP → IP interna asignada dentro del clúster
 - EXTERNAL-IP → IP pública (si aplica). En Minikube normalmente es <none>
 - PORT(S) → Puertos que el servicio expone
 - AGE → Cuánto tiempo lleva creado

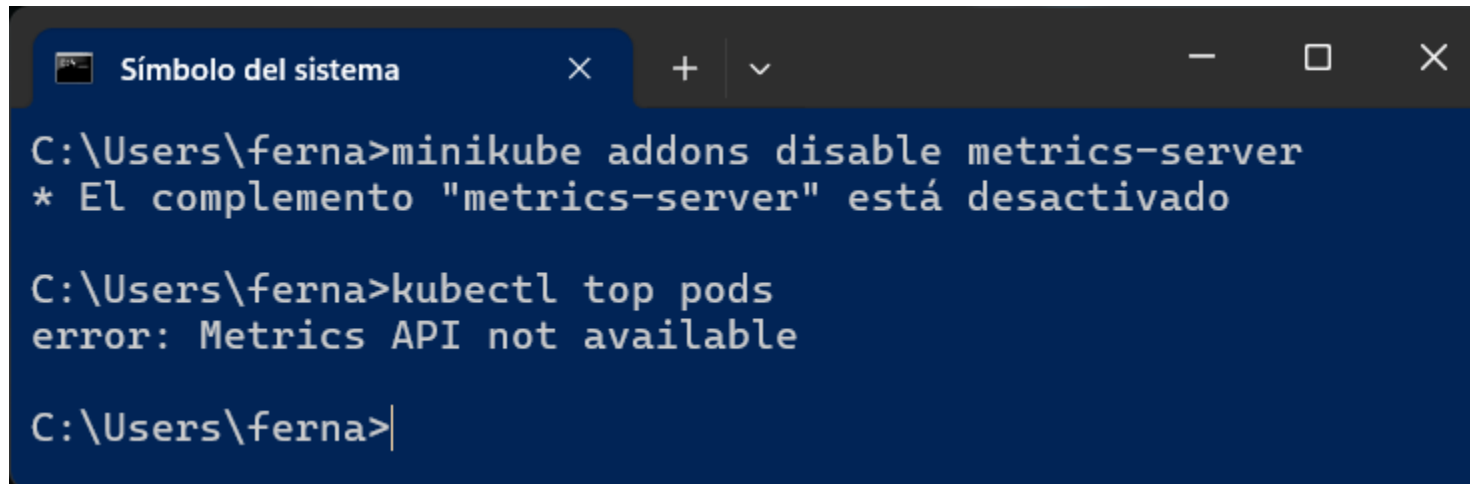
- Addons (extensiones) :
- Ejemplo **metrics-server**:
 - `kubectl top node` → Muestra estadística de uso de recursos de los nodos.
 - `kubectl top pods` → Muestra estadística de uso de recursos de los pods.

```
Símbolo del sistema
C:\Users\ferna>kubectl top node
NAME          CPU(cores)   CPU(%)   MEMORY(bytes)   MEMORY(%)
minikube      179m         1%       1016Mi          13%

C:\Users\ferna>kubectl top pods
NAME          CPU(cores)   MEMORY(bytes)
hello-node-5fd7c866d5-z95gf  1m           31Mi

C:\Users\ferna>
```

- Addons (extensiones) :
- Ejemplo **metrics-server**:
 - Desactivación:
 - minikube addons disable metrics-server



```
C:\Users\ferna>minikube addons disable metrics-server
* El complemento "metrics-server" está desactivado

C:\Users\ferna>kubectl top pods
error: Metrics API not available

C:\Users\ferna>
```

- Clean Up:
 - Eliminación de los recursos del cluster:
 - `kubectl delete service hello-node`
 - `kubectl delete deployment hello-node`
 - Detención del cluster:
 - `minikube stop`
 - Eliminación del cluster (detiene el cluster y elimina los recursos):
 - `minikube delete`
 - `minikube delete -p nombre_cluster`
 - Para obtener información de los clusters:
 - `kubectl config get-contexts`

Desplegar objetos en kubernetes

- Desplegar contenedores en kubernetes, pod
 - El Pod es la unidad más pequeña de ejecución en Kubernetes. Un pod puede contener uno o varios contenedores (normalmente uno). Al desplegar un contenedor en Kubernetes, lo haces dentro de un Pod.
- RC (ReplicationController)
 - ReplicationController fue la primera forma de asegurar que un número determinado de réplicas de un Pod estén siempre en ejecución. Hoy en día ha sido reemplazado por ReplicaSet, aunque todavía funciona.
- Deployments/RS
 - Un **Deployment** gestiona actualizaciones declarativas para Pods y ReplicaSets. Al crear un Deployment, este a su vez crea un ReplicaSet, que luego gestiona los Pods. Se especifica en un fichero **deployment.yaml**
- Services
 - Los **Services** exponen una aplicación que corre en un conjunto de Pods. Proveen una IP estable y nombre DNS interno. Se especifica en un fichero **service.yaml**
- Gestion de Namespaces
 - Los **Namespaces** permiten dividir recursos del clúster entre múltiples usuarios o equipos.

- Despliegue declarativos en kubernetes
 - La forma recomendada de trabajar en Kubernetes es mediante archivos **YAML declarativos**. Esto permite versionar y automatizar la infraestructura como código.
- Aplicaciones multicontenedor en kubernetes
 - Un Pod puede contener múltiples contenedores que comparten red y almacenamiento. Esto es útil para patrones como **sidecar** o **adapter**.
- Rolling Update
 - Con Deployments, Kubernetes hace actualizaciones **rolling** automáticamente: reemplaza Pods antiguos gradualmente por nuevos sin downtime.
- Autoscaling
 - Kubernetes permite escalar horizontalmente los Pods en función del uso de recursos (CPU, memoria, etc.) mediante el **Horizontal Pod Autoscaler (HPA)**.
- Secretos
 - Los **Secrets** permiten almacenar información sensible como contraseñas, tokens y claves SSH.

■ Ejemplo de despliegue:

- Disponer de una imagen Docker en Docker Hub para desplegar.
- Iniciar Docker Desktop.
- Ejecutar el cluster: `minikube start`
- Confirmar que el cluster está creado: `minikube profile list`
- Acceder al panel de contro: `minikube dashboard`

■ Ejemplo de despliegue:

- Fichero deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-app
  labels:
    app: mi-app
spec:
  replicas: 10
  selector:
    matchLabels:
      app: mi-app
  template:
    metadata:
      labels:
        app: mi-app
    spec:
      containers:
        - name: mi-app
          image: fuya/mi-imagen-para-kubernetes
          ports:
            - containerPort: 8081
```

Nombre del deploy

Número de pods

Imagen docker

Puerto interno del contenedor en el pod



■ Ejemplo de despliegue:

- Fichero deployment.yaml:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-app
  labels:
    app: mi-app
spec:
  replicas: 10
  selector:
    matchLabels:
      app: mi-app
  template:
    metadata:
      labels:
        app: mi-app
    spec:
      containers:
        - name: mi-app
          image: fuya/mi-imagen-para-kubernetes
          ports:
            - containerPort: 8081
```

La especificación los contenedores (containers) también admite determinar recursos mínimos y máximos de memoria y CPU

resources:

requests:

memory: "256Mi"

cpu: "250m"

limits:

memory: "512Mi"

cpu: "500m"

▪ Ejemplo de despliegue:

- Fichero service.yaml:

```
apiVersion: v1
kind: Service
metadata:
  name: mi-app-service
spec:
  selector:
    app: mi-app
  ports:
    - protocol: TCP
      port: 80 # Puerto expuesto por el servicio
      targetPort: 8081 # Puerto en el contenedor
  type: LoadBalancer
```

Nombre del deployment

Nombre del servicio

- Expone el servicio al exterior del clúster.
- Crea una IP pública o DNS (en entornos cloud).
- Redirige las solicitudes entrantes en el puerto 80 al puerto 8081 del contenedor dentro de los Pods seleccionados.

Indica a Kubernetes que debe exponer la aplicación al exterior del clúster mediante un balanceador de carga externo.

■ Ejemplo de despliegue:

- Realizar despliegue:
 - Deployment: Crea los **pods** y el **Deployment** con las especificaciones definidas.
 - `kubectl apply -f deployment.yaml`
 - Service: Crea el servicio que expone la aplicación, ya sea como ClusterIP, NodePort o LoadBalancer.
 - `kubectl apply -f service.yaml`
- Uso de los servicios:
 - NOTA: En Minikube, el tipo LoadBalancer no crea una IP pública automáticamente como lo haría en la nube.
 - `minikube tunnel`
 - `kubectl get service mi-app-service`

```
Símbolo del sistema
C:\Users\ferna>curl http://127.0.0.1:80/saludo
{"id":1,"mensaje":"Hello, World!"}
C:\Users\ferna>
```

```
Símbolo del sistema
C:\Users\ferna>kubectl get services mi-app-service
NAME          TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
mi-app-service LoadBalancer  10.108.183.135 127.0.0.1      80:30583/TCP     17m
```

■ Ejemplo de despliegue:

- Eliminar service y deployment:
 - Consulta:
 - `kubectl get services`
 - `kubectl get deployment`
 - Eliminación:
 - `kubectl delete service nombre-service`
 - `kubectl delete deployment nombre-app`

Almacenamiento con kubernetes

- Volúmenes en kubernetes
 - Un volumen en Kubernetes es un espacio de almacenamiento que puede ser usado por uno o varios contenedores en un pod para persistir datos más allá del ciclo de vida de un contenedor.
 - El almacenamiento local (dentro del contenedor) es efímero y se pierde si el contenedor muere o se reinicia.
 - Un volumen permite mantener datos aunque el contenedor se recree.
- (Algunos) Tipos de volúmenes:
 - **emptyDir**. Es un volumen temporal que se crea cuando un pod se inicia y existe mientras el pod esté vivo; es ideal para almacenar datos efímeros o compartir información entre contenedores del mismo pod, pero sus datos se pierden si el pod termina o se reinicia
 - **hostPath**. permite montar un directorio o archivo específico del nodo donde corre el pod directamente dentro del contenedor, lo que puede ser útil para acceder a recursos locales del nodo, aunque su uso debe ser cuidadoso ya que puede afectar la portabilidad y seguridad.
 - **NFS**. Es un volumen que monta un sistema de archivos compartido en red, permitiendo que múltiples pods, incluso en diferentes nodos, accedan concurrentemente a los mismos datos persistentes, ideal para aplicaciones que requieren almacenamiento compartido, aunque depende de un servidor NFS externo o desplegado en el clúster.
- Asignar un volumen persistente a un pod

- Asignar un volumen persistente a un pod
 - Se realiza a través de un **PersistentVolumeClaim (PVC)**.
 - Paso 1: Crear un PersistentVolumeClaim (PVC)

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mi-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

- Asignar un volumen persistente a un pod
 - Paso 2: Crear un Pod que use el PVC

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-con-pvc
spec:
  containers:
  - name: app
    image: nginx
    volumeMounts:
    - name: volumen-persistente
      mountPath: /data
  volumes:
  - name: volumen-persistente
    persistentVolumeClaim:
      claimName: mi-pvc
```

Advanced Topics

- **Service Discovery**
 - Service Discovery (descubrimiento de servicios) en Kubernetes es el mecanismo que permite que las aplicaciones o microservicios se encuentren y se comuniquen entre sí dentro del clúster, sin necesidad de conocer de antemano las direcciones IP concretas de otros servicios.
 - En Kubernetes, los Pods son efímeros: pueden morir, ser reiniciados o escalados dinámicamente. Esto significa que sus IPs pueden cambiar frecuentemente, por lo que no es práctico depender de direcciones IP fijas. Aquí es donde entra Service Discovery.
 - Se pueden implementar de dos maneras:
 - **DNS-based Service Discovery.** Kubernetes incluye un componente llamado **CoreDNS**, que actúa como un servidor DNS interno.
 - **Environment Variables.** Kubernetes también inyecta variables de entorno en los Pods cuando se crean, conteniendo información de los servicios disponibles (como IP y puerto). Este método es más limitado y no se actualiza dinámicamente si el servicio cambia después de creado el Pod.

- Ingress Controller
 - Un Ingress Controller es un componente que gestiona el acceso externo (HTTP/HTTPS) a los servicios dentro de un clúster de Kubernetes. El Ingress Controller es el componente que hace que eso funcione de verdad. Kubernetes no incluye uno por defecto, por lo tanto, se debe instalar.
- Affinity / Anti-Affinity
 - Son reglas que le dicen al scheduler de Kubernetes cómo y dónde colocar los pods en los nodos del clúster, para mejorar la disponibilidad, rendimiento, o cumplimiento de políticas.
 - **Affinity**. Le indica al scheduler que prefiera o exija colocar pods juntos, basándose en etiquetas de nodos o pods.
 - **Anti-Affinity**. Es lo contrario: le dice al scheduler que evite colocar pods juntos.
- NodeSelector using Labels
 - Es una forma sencilla y básica de **asignar pods a nodos específicos** en un clúster Kubernetes, usando etiquetas que estén puestas en esos nodos.

- Healthchecks

- Kubernetes usa dos tipos principales de healthchecks para verificar el estado de los contenedores dentro de los pods:
 - **Liveness Probe** (Comprobación de vida). Indica si el contenedor está vivo o no. Si falla, Kubernetes asume que el contenedor está colgado o en mal estado y lo reinicia. Útil para detectar situaciones donde la aplicación está bloqueada o no responde.
 - **Readiness Probe** (Comprobación de disponibilidad). Indica si el contenedor está listo para recibir tráfico. Si falla, Kubernetes deja de enviar tráfico al pod (no lo incluye en el Service). Útil para evitar enviar tráfico a pods que aún están arrancando o están temporalmente ocupados.

- ConfigMap

- Un ConfigMap es un objeto de Kubernetes que permite almacenar configuraciones no confidenciales en forma de pares clave-valor.
- Estas configuraciones pueden luego ser consumidas por los Pods para que las aplicaciones puedan obtener valores de configuración sin tener que hardcodearlos dentro de la imagen del contenedor.
- Separan la configuración del código.
- Facilitan cambios en configuración sin tener que reconstruir la imagen del contenedor.
- Permiten compartir configuraciones entre varios pods o servicios.

- **Horizontal Pod Autoscaler**

- El Horizontal Pod Autoscaler es un recurso de Kubernetes que ajusta automáticamente el número de réplicas de un Deployment, ReplicaSet o StatefulSet en función de métricas observadas, como el uso de CPU o cualquier métrica personalizada.
- Escala hacia arriba o hacia abajo la cantidad de pods según la demanda real.
- Mejora la eficiencia de recursos.
- Mantiene el rendimiento de la aplicación sin intervención manual.

- **Prometheus**

- Es un sistema de monitoreo y recolección de métricas.
- Recoge métricas de aplicaciones y sistemas mediante scraping (consultas periódicas HTTP).
- Almacena esas métricas en una base de datos de series temporales.
- Permite hacer consultas sobre las métricas con su lenguaje propio, PromQL.
- Muy popular para monitoreo de Kubernetes porque se integra bien con él.

- **Exporters**

- Son pequeños programas que exponen métricas en formato que Prometheus puede entender.
- Funcionan como “traductores” de métricas específicas de sistemas, aplicaciones o hardware.
- Ejemplos comunes:
 - node-exporter: métricas del nodo (CPU, memoria, disco, red).
 - kube-state-metrics: métricas del estado del clúster Kubernetes.
 - cadvisor: métricas de contenedores. Se despliegan como pods o servicios dentro del clúster.

- **Grafana**

- Plataforma para visualizar métricas y datos.
- Se conecta a Prometheus (y otros orígenes de datos).
- Permite crear dashboards personalizados y alertas.
- Muy usada para analizar métricas en tiempo real y tendencias históricas.