Servicios

Antonio Espín Herranz

Contenidos

- Servicios
- Tipos de servicios

Servicios

Los servicios (Services) nos permiten acceder a las aplicaciones que hemos desplegado en el cluster.

- Un Service es una abstracción que **nos permite acceder a un conjunto de pods** (que se han creado a partir de un **Deployment**) que implementan una aplicación (Por ejemplo: acceder a un servidor web, a un servidor de base de datos, a un servicio que forma parte de una aplicación, ...).

- A cada Pod se le asigna una IP a la que no se puede acceder directamente, por lo tanto, necesitamos un **Service** que nos ofrece **una dirección virtual (CLUSTER-IP) y un nombre** que identifica al conjunto de Pods que representa, al cual nos podemos conectar.

Servicios

- La conexión al Service se puede realizar **desde otros Pods o desde el exterior** (mediante la generación aleatoria de un puerto).
  - Por ejemplo, si tenemos una aplicación formada por dos Services: servidor web y servidor de base de datos, tendremos que acceder desde el exterior al servidor web, y acceder al servidor de base de datos desde el servidor web.
  - En principio no será necesario acceder al servidor de base de datos desde el exterior.

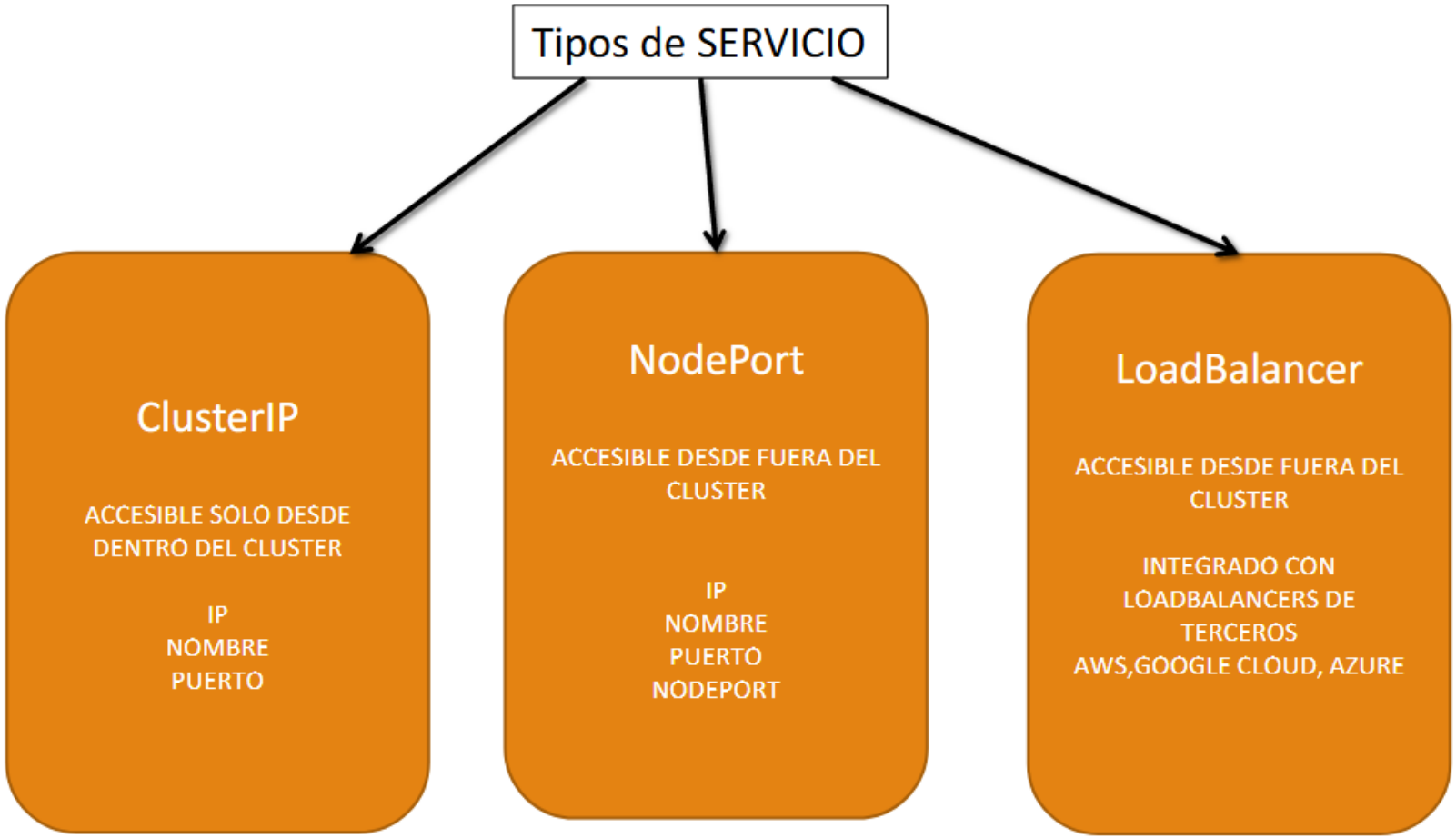
- Si el Deployment que hemos creado tiene más de un Pod asociado, el Service que representa el acceso a esta aplicación **balanceará la carga** entre los Pods con una política Round Robin.

Servicios

- En el cluster existirá un componente que nos ofrece un **servicio DNS**. Cada vez que creamos un Service se actualizará el DNS para resolver el nombre que hemos asignado al Service con la IP virtual (CLUSTER-IP) que se le ha asignado.

- El servicio es un intermediario entre un deployment (que puede tener n réplicas) y un cliente externo.

Tipos principales



Servicios

- Un servicio puede exponer un pod o un deployment, lo normal es exponer el deployment.

- Se puede hacer en modo comando o de forma declarativa.

- Ver ejemplo con los pasos

Crear servicio NodePort

- Vamos a exponer un deployment con un apache y después vamos a crear un servicio para poder acceder de forma externa desde el navegador.

- **# Crear un deployment de tipo apache**
- kubectl create deployment apache1 --image=httpd

- **# Consultarlo: vemos el pod, el deployment y el replica set.**
- kubectl get all

- **# Crear un servicio: exponemos el deployment**
- kubectl expose deploy apache1 --port=80 --type=NodePort

Crear un servicio NodePort

- **# Consultar el Servicio:**
- **# La externa IP aparecerá como <none>, es para entornos cloud**
- **# El Cluster-IP tendrá la IP asignada.**
- kubectl get service

```
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
apache1      NodePort    10.109.190.119   <none>        80:32612/TCP   21s
kubernetes   ClusterIP   10.96.0.1        <none>        443/TCP        4d6h
nginx        NodePort    10.111.99.171    <none>        80:30533/TCP   46h
```

- **# Vemos la dirección de minikube**
- minikube ip

Crear servicio NodePort

- **# En algunos Windows / Mac no coge bien la dirección IP de minikube**
- **# Lo podemos comprobar con el comando, hay veces que en vez de poner**
- **# la IP, pone: localhost o 127.0.0.1**
- minikube service list

NAMESPACE	NAME	TARGET PORT	URL
default	apache1	80	
default	kubernetes	No node port	
default	nginx	service-http/80	
kube-system	kube-dns	No node port	
kubernetes-dashboard	dashboard-metrics-scraper	No node port	
kubernetes-dashboard	kubernetes-dashboard	No node port	

En este ejemplo, debería de aparece en URL

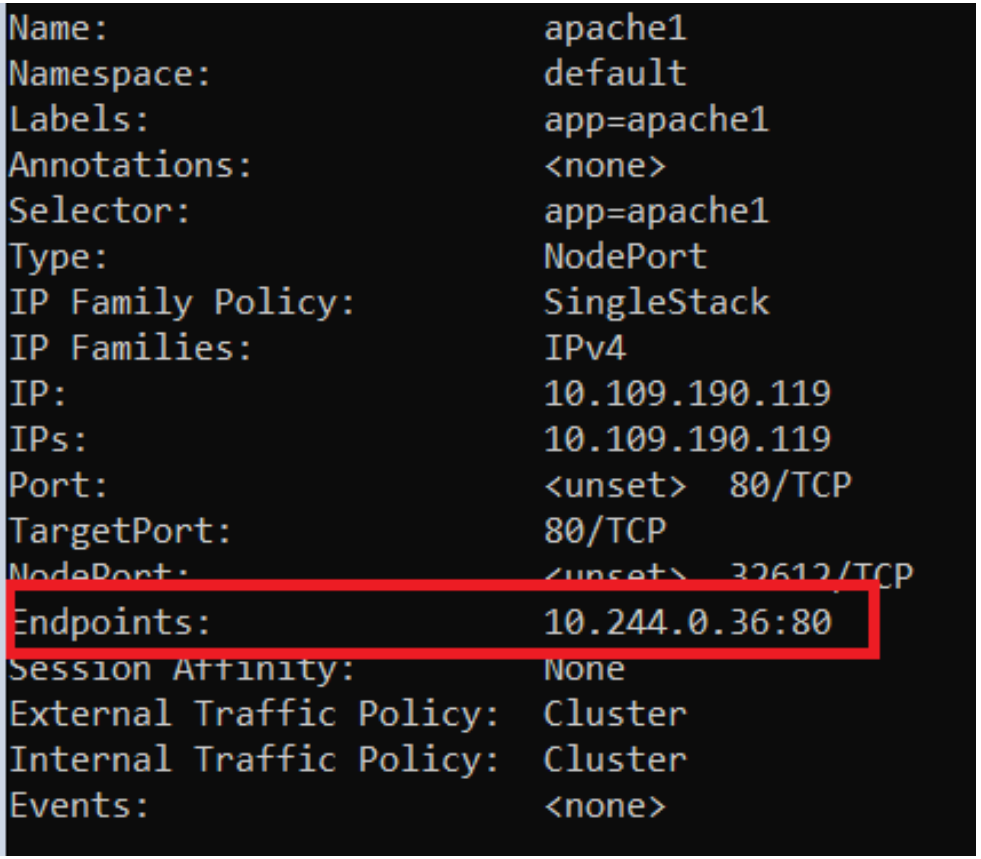
Crear servicio NodePort

- **# Este problema de no mostrar la URL lo suelen dar los Mac y los Windows, en Linux se da menos, podemos utilizar el siguiente comando:**
- Minikube service apache1
- **# y debería abrir un navegador →** http://127.0.0.1:56298/

Descripción del servicio

- Comando: kubectl describe service apache1
- (podemos poner también → kubectl describe svn apache1)

- El **EndPoint** representa el POD al que está apuntando el servicio.



Descripción del servicio

- Con el comando: kubectl get pod –o wide
- Podemos ver las direcciones **IP de los PODs**

```
NAME                                READY   STATUS    RESTARTS        AGE   IP            NODE       NOMINATED NODE   READINESS GATES
apache1-76bcd4fc76-m52rn            1/1     Running   0               27m   10.244.0.36   minikube   <none>           <none>
deployment-nginx-764c7c96bc-88xs8   1/1     Running   2 (7h30m ago)   46h   10.244.0.34   minikube   <none>           <none>
deployment-nginx-764c7c96bc-k7wm6   1/1     Running   2 (7h30m ago)   46h   10.244.0.28   minikube   <none>           <none>
pod-nginx                           1/1     Running   1 (7h30m ago)   9h    10.244.0.29   minikube   <none>           <none>
pod-nginx-2                         1/1     Running   1 (7h30m ago)   8h    10.244.0.35   minikube   <none>           <none>
```

De esta forma el servicio puede localizar el POD y por detrás lo que está utilizando son los selectores de las etiquetas

Selector: app=apache1 (en el servicio)

Mostrar las etiquetas del POD

- Para mostrar las etiquetas del POD y ver como coinciden con las del servicio, tenemos el comando (indicar el nombre del POD)
- Kubectl get pod apache1-76bcd4fc76-m52rn --show-labels

```
NAME                       READY   STATUS    RESTARTS   AGE   LABELS
apache1-76bcd4fc76-m52rn   1/1     Running   0          34m   app=apache1,pod-template-hash=76bcd4fc76
```

Kubernetes pone por defecto la etiqueta app=apache1

Prueba: escalar el deploy
¿Qué pasa con las etiquetas?

- Para escalar (añadir más réplicas) el deploy:
- Kubectl scale deploy apache1 --replicas=3

```
NAME                       READY   STATUS    RESTARTS   AGE   IP            NODE       NOMINATED NODE   READINESS GATES
apache1-76bcd4fc76-74d42   1/1     Running   0          38s   10.244.0.38   minikube   <none>           <none>
apache1-76bcd4fc76-9zfgr   1/1     Running   0          38s   10.244.0.37   minikube   <none>           <none>
apache1-76bcd4fc76-m52rn   1/1     Running   0          42m   10.244.0.36   minikube   <none>           <none>
```

- Ya tenemos 3 PODs cada uno con su IP.

Actualiza los EndPoint

- Consultamos la descripción del servicio:
- Kubectl describe service apache1

```
Name:                     apache1
Namespace:                default
Labels:                   app=apache1
Annotations:              <none>
Selector:                 app=apache1
Type:                     NodePort
IP Family Policy:         SingleStack
IP Families:              IPv4
IP:                       10.109.190.119
IPs:                      10.109.190.119
Port:                     <unset>  80/TCP
TargetPort:               80/TCP
NodePort:                 <unset>  32612/TCP
Endpoints:                10.244.0.36:80,10.244.0.37:80,10.244.0.38:80
Session Affinity:         None
External Traffic Policy:  Cluster
Internal Traffic Policy:  Cluster
Events:                   <none>
```

Actualiza los EndPoint

- Borrar un POD:
- Kubectl delete pod nombre_pod

- Creará uno nuevo, para mantener las replicas.

- Y luego actualizará las IPs de los POD y del Servicio.

Servicios Cluster IP

- Son sólo accesibles desde el propio clúster.
- Por defecto, los servicios se crean de este tipo.

Ejemplo

- Crear dos deployment:
- Kubectl create deployment redis-master --image=redis
- Para el cliente:
- Kubectl create deployment redis-cli --image=redis
- El servicio: Hay que exponer el master
- Kubectl expose deploy redis-master --port=6379 --type=ClusterIP
- No se sería necesario type (por defecto será ClusterIP)
- Sólo es accesible desde dentro.

Ejemplo

- Conectamos con el POD redis-cli:
- Kubectl exec redis-cli –numero –it -- bash
- Conectamos dentro del contenedor con el redis-master (el nombre del servicio actúa como el nombre de la máquina).
- root@.....:/data# redis-cli –h redis-master

```
D:\OneDrive\Escritorio\docker_kubernetes\repositorio\kubernetes\codigo_curso\crear_pod_nginx>kubectl exec redis-cli-6fb748d487-5jv99 -it -- bash
root@redis-cli-6fb748d487-5jv99:/data# redis-cli -h redis-master
redis-master:6379>
```

- Lanzar comando de la BD Redis: set k1 v1
- Probar a conectar desde el servidor y dentro poner: redis-cli
- Y recuperar la clave con get k1

Servicios

- Cuando tenemos más de un Pod ofreciendo el mismo servicio, realmente tenemos un clúster y es importante distinguir entre servicios sin estado (***stateless***) o con estado (***stateful***)).

- En un servicio sin estado (por ejemplo, un servidor web que sirva contenido estático), las peticiones son independientes y se pueden servir por diferentes nodos sin problema, aunque en el caso de un servidor web, deberíamos asegurarnos previamente de que el directorio con los datos es el mismo. Un servicio de este tipo lo podemos escalar con un despliegue sin problema.

- Por otra parte, si el servicio tiene estado (por ejemplo, un servidor de bases de datos), una petición puede depender de otra anterior, por lo que puede haber incoherencias si simplemente creamos un cluster de nodos iguales. En este tipo de servicios, es necesaria una configuración adicional que controle el estado y que haga que los datos que sirve cada Pod son coherentes entre sí.

Tipos de Servicios

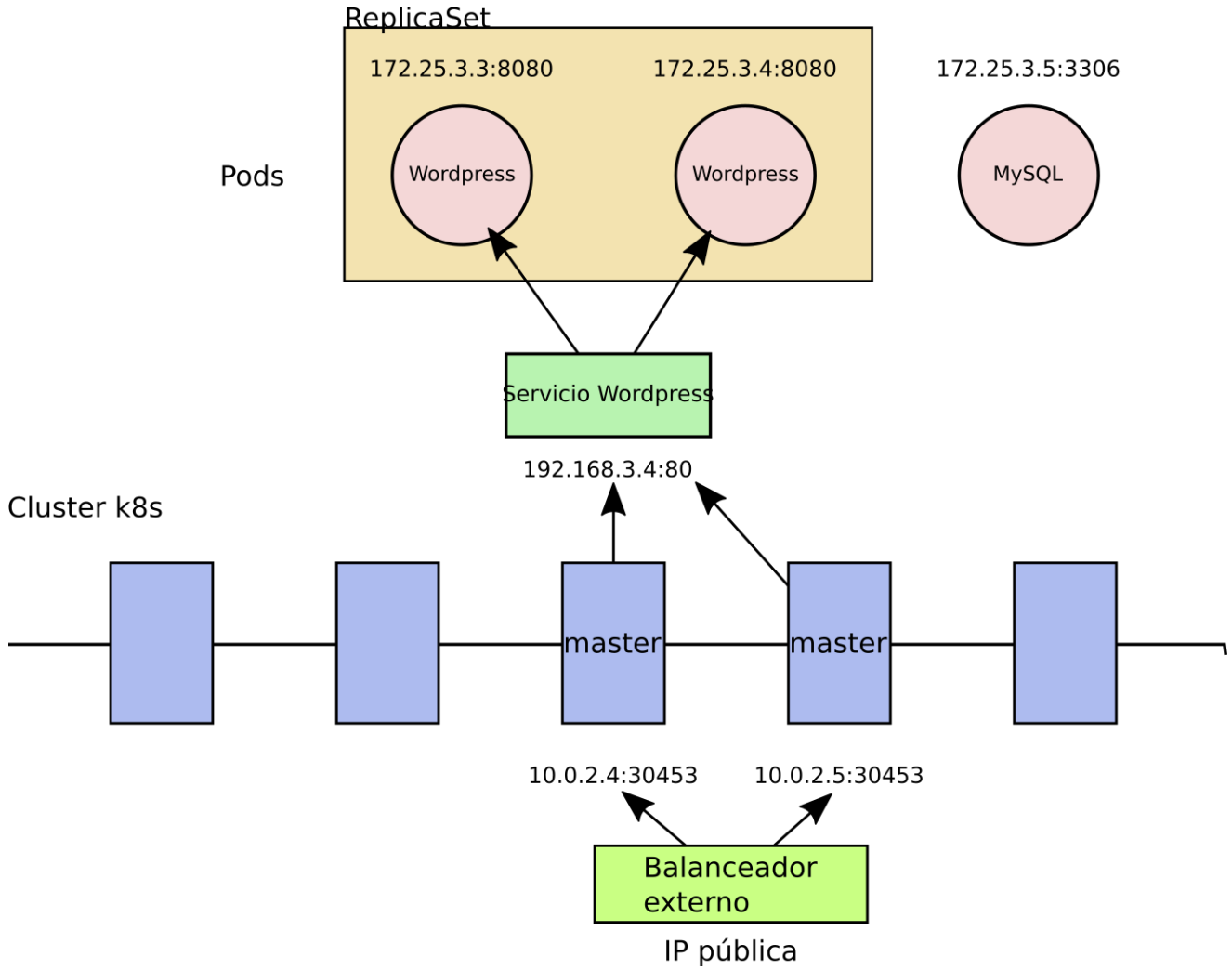
- **Cluster IP**
  - Solo se permite el acceso interno a un Service de este tipo. Es decir, si tenemos un despliegue con una aplicación a la que no es necesario acceder desde el exterior, crearemos un Service de este tipo para que otras aplicaciones puedan acceder a ella (por ejemplo, una base de datos). Es el tipo por defecto. Si deseamos seguir accediendo desde el exterior, para hacer pruebas durante la fase de desarrollo podemos seguir utilizando la instrucción kubectl port-forward.

- **NodePort**
  - Abre un puerto, para que el Service sea accesible desde el exterior. Por defecto el puerto generado está en el rango de 30000:40000. Para acceder usamos la ip del servidor master del cluster y el puerto asignado.

- **LoadBalancer**
  - Este tipo sólo está soportado en servicios de cloud público (GKE, AKS o AWS). El proveedor asignará un recurso de balanceo de carga para el acceso a los Services. Si usamos un cloud privado como OpenStack, necesitaremos un plugin para configurar el funcionamiento.

Cluster IP

- Necesitamos que los Pods de Wordpress accedan al Pod del MySQL.
- La IP que ha tomado el Pod de MySQL (172.25.3.5) es inaccesible desde los Pods de Wordpress.
- Por lo tanto, hemos creado un Service de tipo ClusterIP, que ha obtenido una ip virtual (192.168.3.5) y expone el puerto de MySQL 3306.
- Esta IP sí es accesible desde los Pods de Wordpress.
- Al acceder a esta IP se balanceará la carga entre los Pods de MySQL (en el ejemplo sólo tenemos uno).
- Además, en el Wordpress no necesitamos configurar la IP virtual del Service que hemos creado, ya que disponemos de un servidor DNS que resuelve el nombre del Service mysql en la dirección virtual del Service (192.168.3.5). Por lo tanto, en la configuración de Wordpress pondremos el nombre mysql como host del servidor de base de datos al que debe acceder.



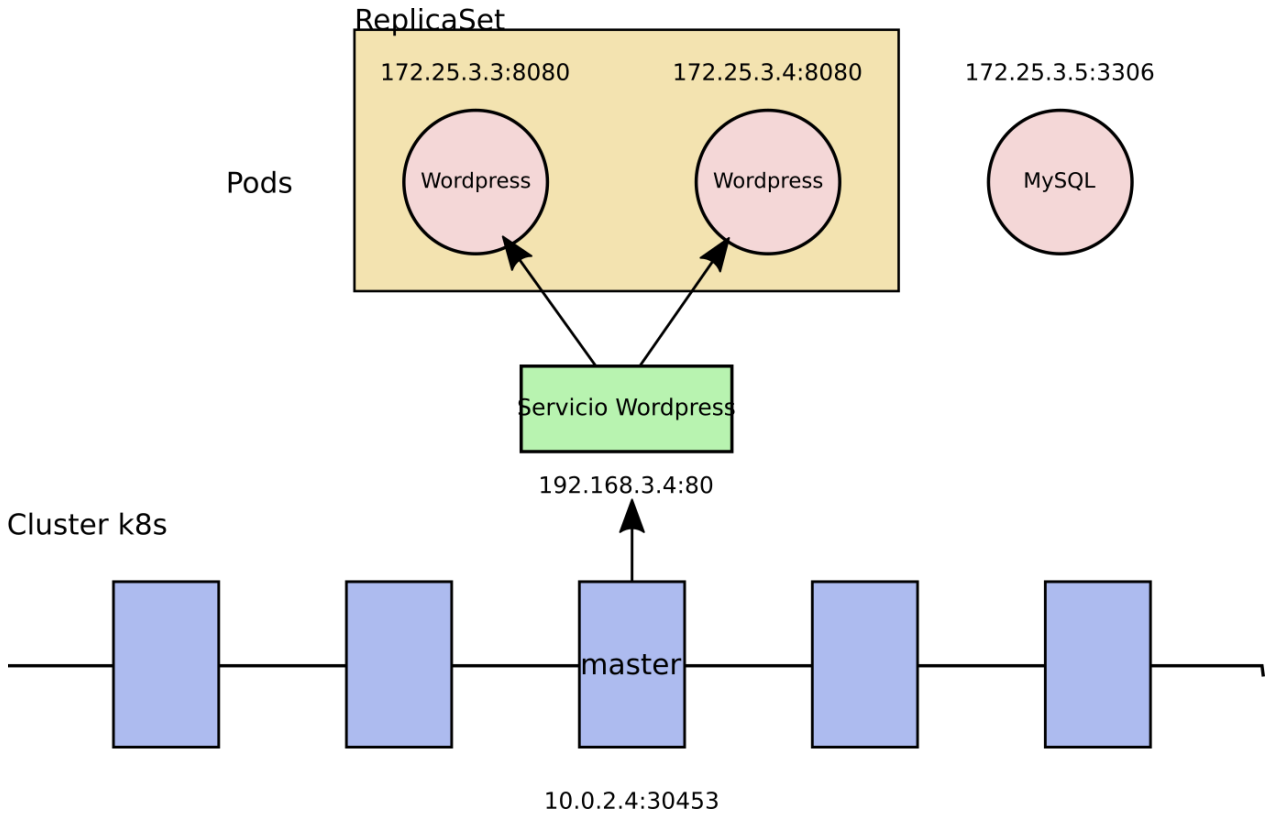
NodePort

- Necesitamos que los Pods de Wordpress sean accesibles desde el exterior, para que podamos acceder a la aplicación.

- La IP que han tomado los Pods de Wordpress (172.25.3.3, ...) no son accesibles desde el exterior. Además, comprobamos que estos Pods están ofreciendo el servicio en el puerto 8080.

- Por lo tanto, hemos creado un Service de tipo NodePort que ha obtenido una IP virtual (192.168.3.4) y expone el puerto 80.

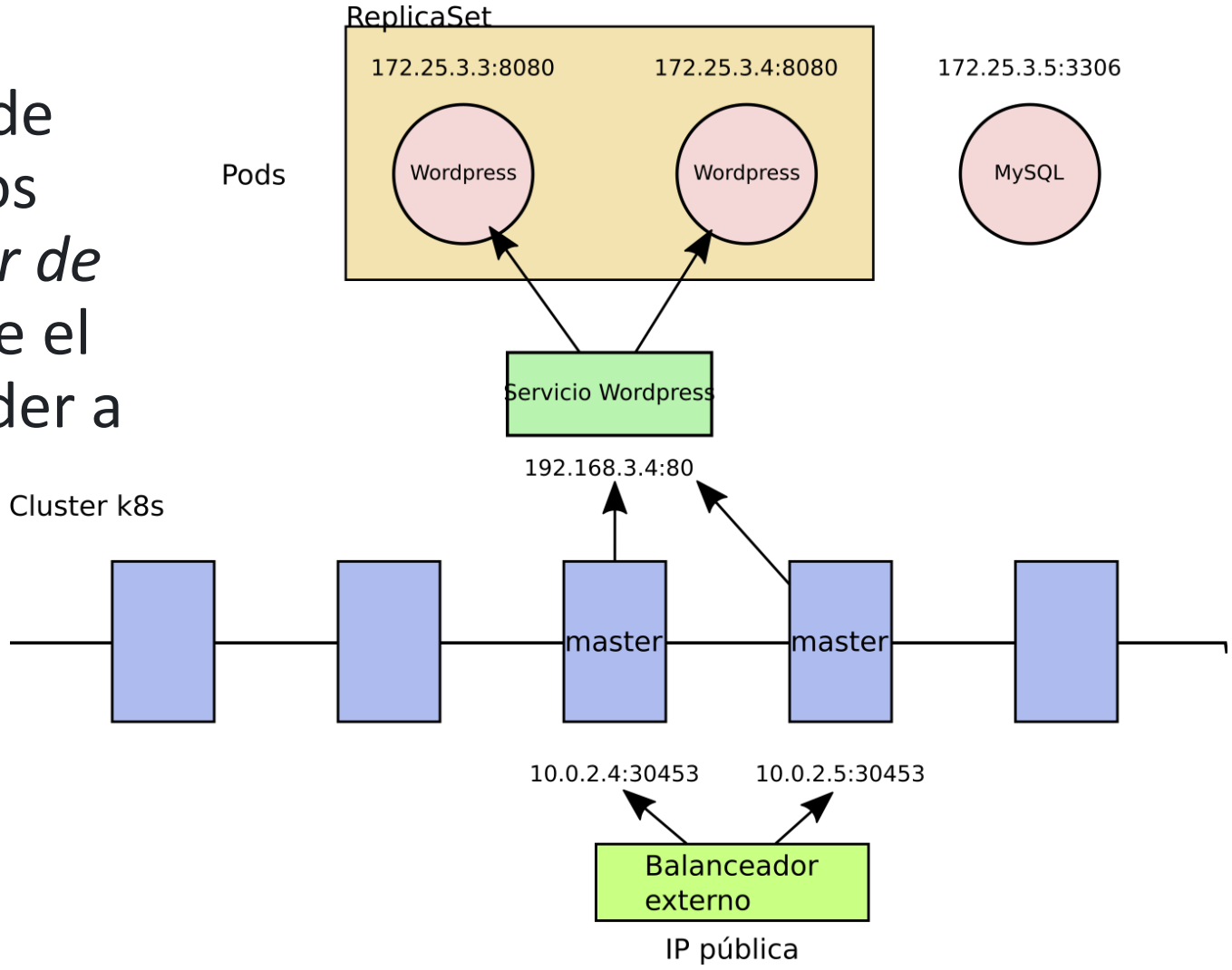
- Al acceder a esta IP al puerto 80 se balanceará la carga entre los Pods de Wordpress, accediendo a las IPs de los Pods de Wordpress al puerto 8080.



El Service NodePort ha asignado un puerto de acceso aleatorio (entre el 30000 - 40000) que nos permite acceder a la aplicación mediante la IP del nodo master. En el ejemplo si accedemos a 10.0.2.4:30453 estaremos accediendo al Service que nos permitirá acceder a la aplicación.

LoadBalancer

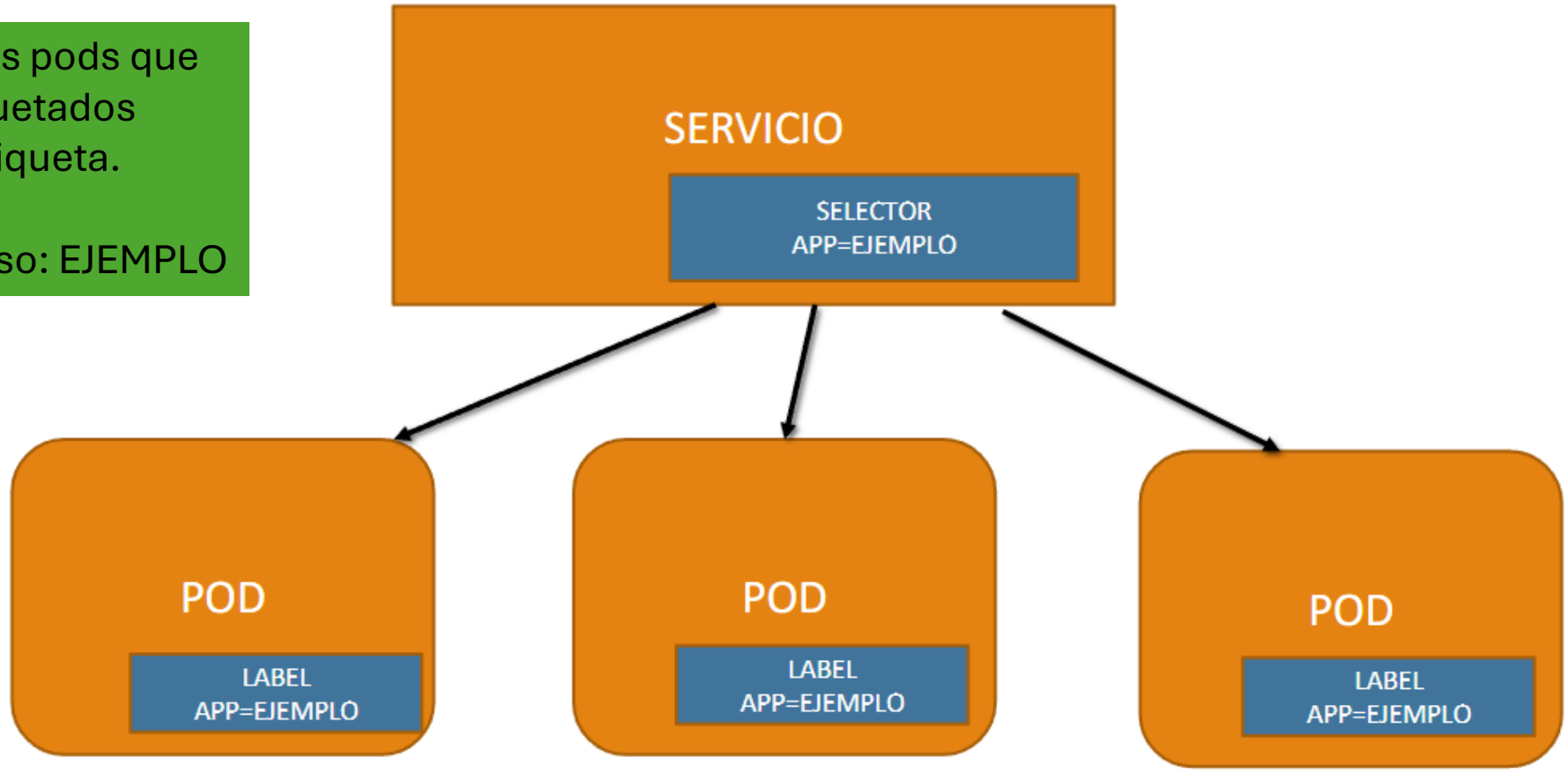
- El cloud de infraestructura donde tengamos instalado el cluster nos ofrecerá un recurso *balanceador de carga* con una IP accesible desde el exterior que nos permitirá acceder a la aplicación directamente.



Uso de selectores

Localiza los pods que estén etiquetados con esa etiqueta.

En este caso: EJEMPLO



Ejemplo: el deploy

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deployment-nginx
  labels:
    app: nginx
spec:
  revisionHistoryLimit: 2
  strategy:
    type: RollingUpdate
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx
        name: contendor-nginx
        ports:
        - name: http
          containerPort: 80
```

Kubectl apply –f fichero.yaml

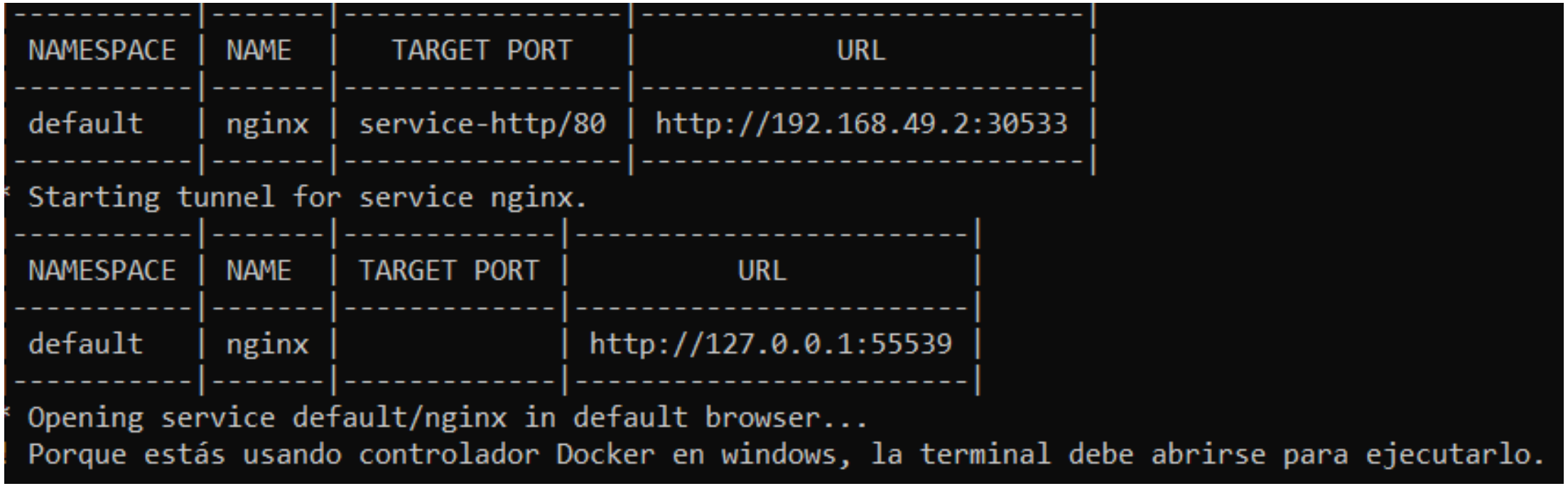
El servicio

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: NodePort
  ports:
  - name: service-http
    port: 80
    targetPort: http
  selector:
    app: nginx
```

Kubectl apply –f servicio.yaml

Acceso al servicio

- **minikube service nginx**



http://127.0.0.1:55539/