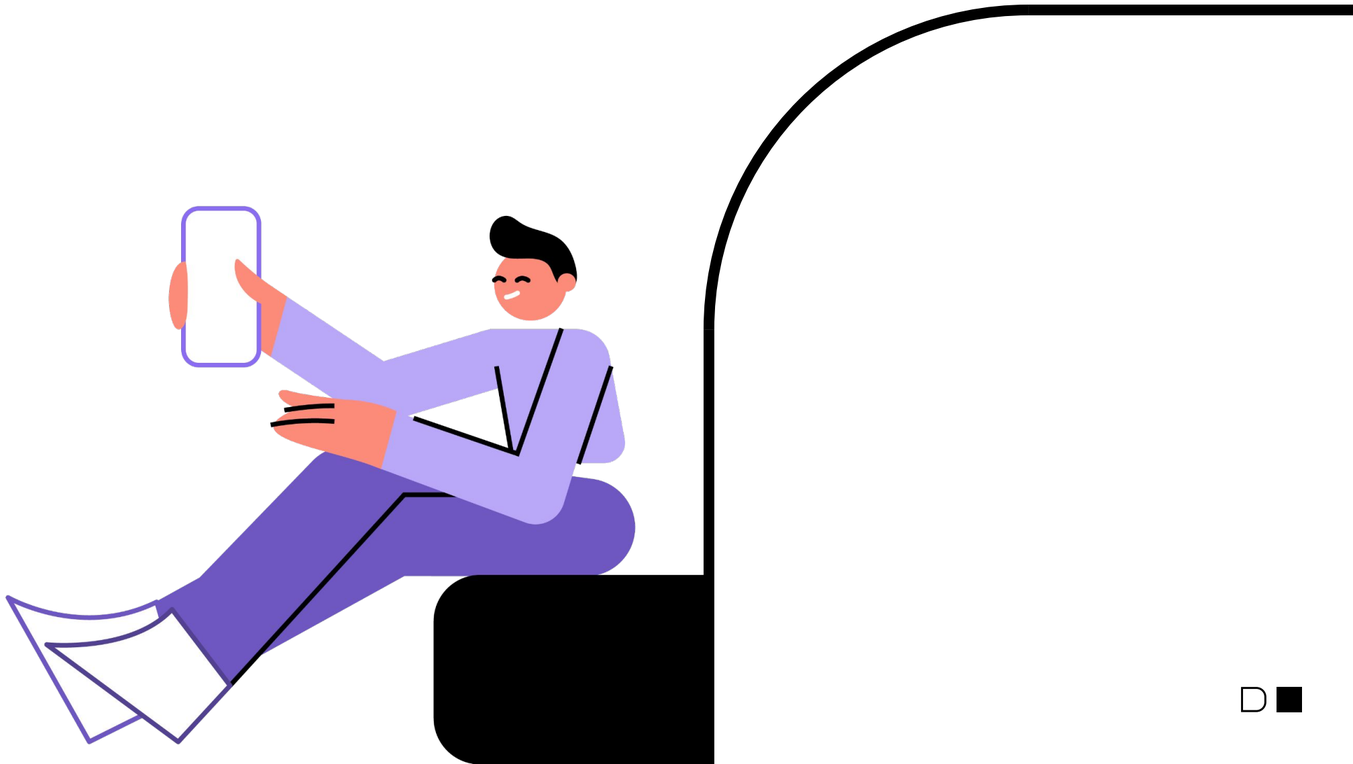
DIGITALHOUSE

Redes en Kubernetes

Certified Tech Developer
The Ultimate Degree

Índice



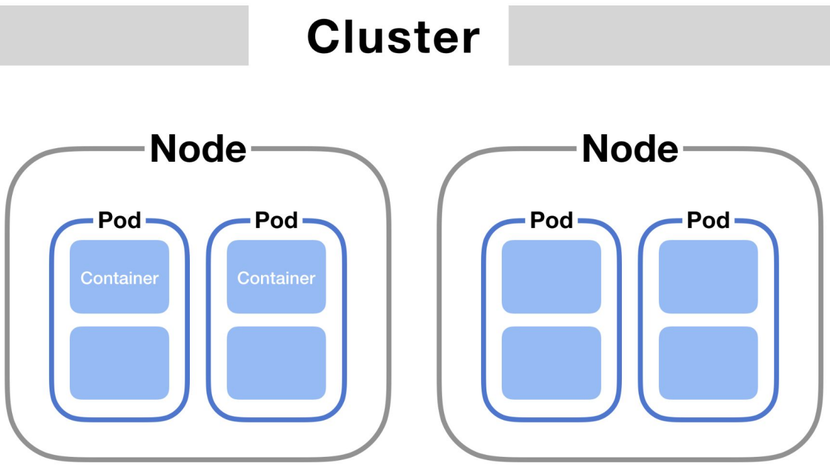
01 Fundamentos

Requisitos de red de Kubernetes

Antes de profundizar en los detalles sobre cómo fluyen los paquetes dentro de un clúster de Kubernetes, primero aclaremos los requisitos para una red de Kubernetes. El modelo de red de Kubernetes define un conjunto de reglas fundamentales:

- Un pod en el cluster debería poder comunicarse libremente con cualquier otro pod en forma directa.
- Cualquier programa que se ejecute en un nodo del cluster debe comunicarse con cualquier pod en el mismo nodo directamente.
- Cada pod tiene su propia dirección IP (IP por pod), y todos los demás pods, pueden acceder a él en esa misma dirección.

5 cosas esenciales que debemos comprender sobre las redes en **Kubernetes**



01 Cómo se comunican los contenedores en el mismo pod.

02 Cómo se comunican entre pods en el mismo nodo.

03 Cómo se comunican entre pods en diferentes nodos.

04 Comunicación entre pods y servicios.

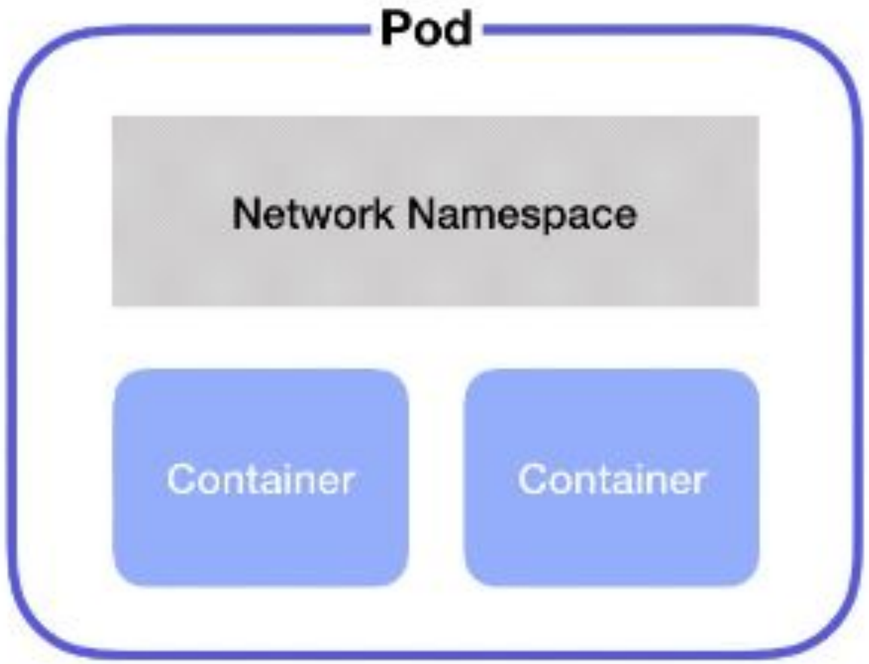
05 ¿Cómo funciona DNS? ¿Cómo descubrimos las direcciones IP?

02 Ruteos a través de Pods

Comunicación entre contenedores en el mismo **pod**

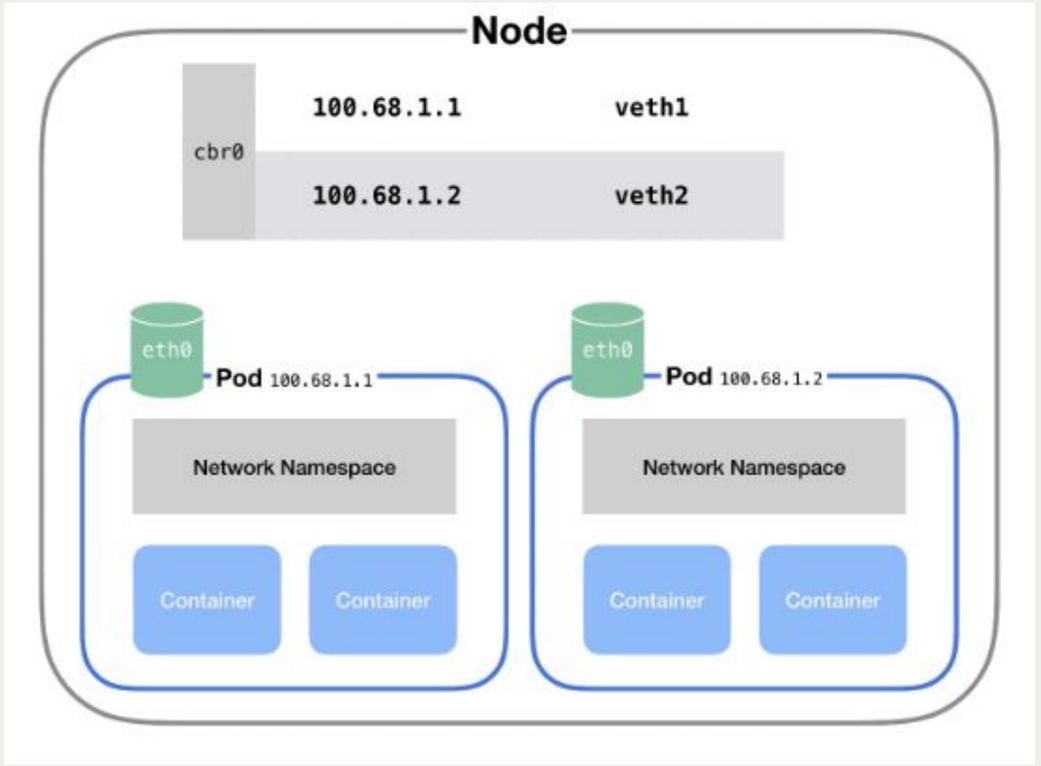
Primero, si tiene dos contenedores ejecutándose en el mismo módulo, ¿cómo se comunican entre sí?

Esto sucede a través de localhost y números de puerto. Al igual que cuando ejecutamos varias VMs en nuestra propia computadora portátil.



Comunicación entre pods en el mismo nodo

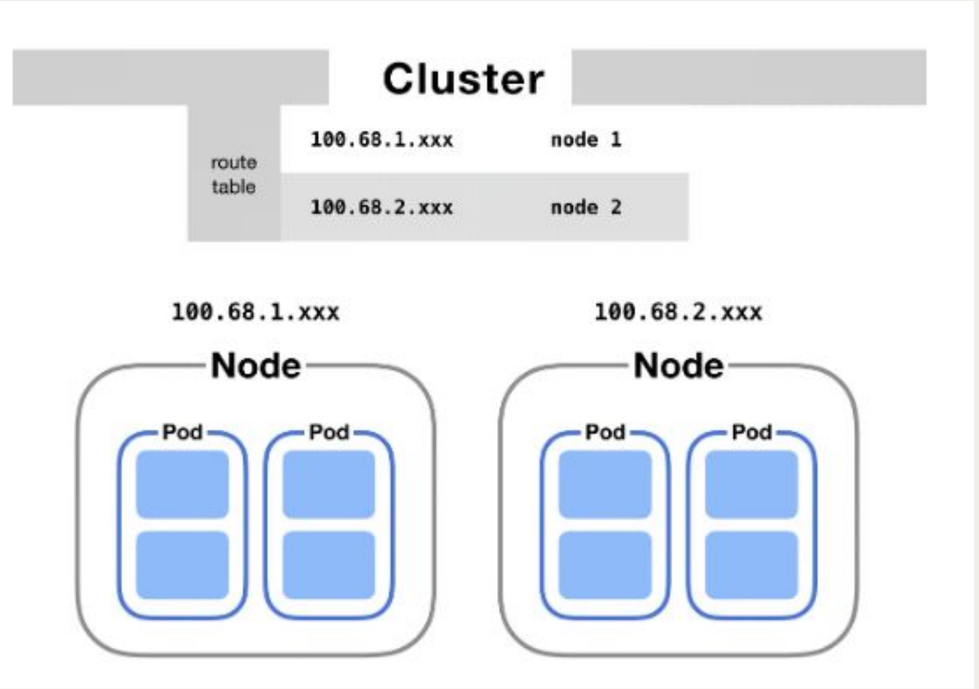
- Cada pod en un nodo tiene su propio espacio de nombres de red. Cada pod tiene su propia dirección IP.
- Y cada pod piensa que tiene un dispositivo ethernet totalmente normal llamado eth0 para realizar solicitudes de red. Pero Kubernetes está fingiendo: es solo una conexión ethernet virtual.
- El dispositivo eth0 de cada pod está realmente conectado a un dispositivo ethernet virtual en el nodo.
- Cuando un pod realiza una solicitud a la dirección IP de otro nodo, realiza esa solicitud a través de su propia interfaz eth0. Esto hace un túnel a la respectiva interfaz virtual del nodo.



Observen en el gráfico que las IPs de cada Pod están asociadas en la tabla más arriba a una interfaz veth1 y veth2. Este es el puente que utilizan los nodos para enviar y recibir tráfico de y hacia los pods.

Pero, ¿y si los pods están en nodos diferentes?

- A nivel de cluster, hay una tabla que asigna rangos de direcciones IP a varios nodos. A los pods en esos nodos se les habrán asignado direcciones IP de esos rangos.
- Por ejemplo, Kubernetes puede proporcionar direcciones de pods en el nodo 1 como 100.96.1.1, 100.96.1.2, etc. Y Kubernetes puede proporcionar direcciones de pods en el nodo 2 como 100.96.2.1, 100.96.2.2, etc.
- Luego, esta tabla se ocupará de que las direcciones IP que sean 100.96.1.xxx vayan al nodo 1, y las direcciones como 100.96.2.xxx vayan al nodo 2.



Una vez que hemos determinado a qué nodo enviar la solicitud, el proceso continúa aproximadamente igual que si los pods hubieran estado en el mismo nodo todo el tiempo.

Comunicación entre pods y services

Un último patrón de comunicación es importante en Kubernetes

- En Kubernetes, un service le permite asignar una sola dirección IP a un conjunto de pods. Realiza solicitudes a un punto final (nombre de dominio/dirección IP) y el service envía solicitudes a un pod en ese servicio.
- Esto sucede a través de kube-proxy, un pequeño proceso que Kubernetes ejecuta dentro de cada nodo.
- Este proceso asigna direcciones IP virtuales a un grupo de direcciones IP de pod reales.
- Una vez que kube-proxy haya asignado la IP virtual del service a una IP de pod real, la solicitud procede como en las secciones anteriores.

¿Cómo funciona el DNS?

¿Cómo funciona el DNS? ¿Cómo descubrimos las direcciones IP?

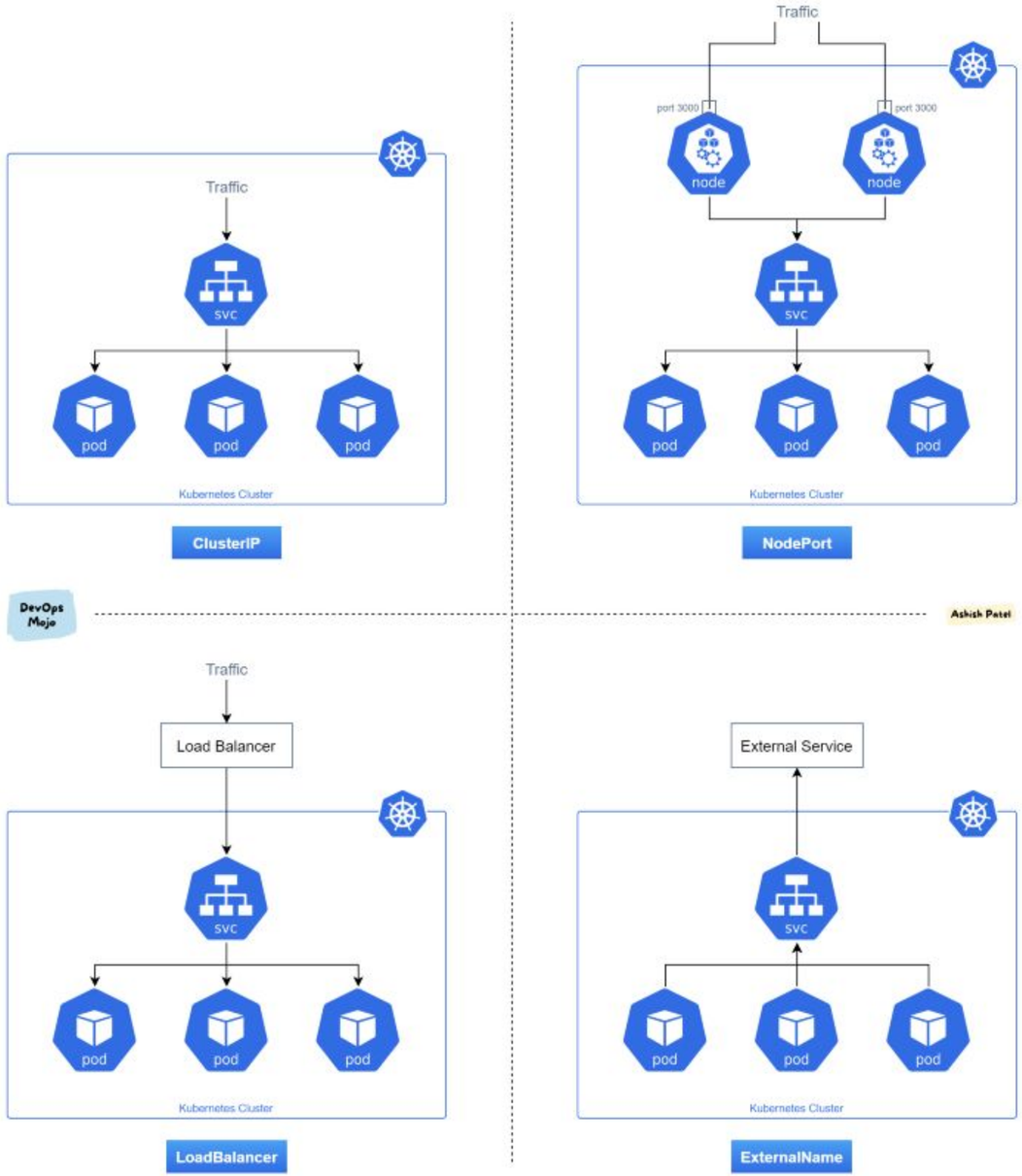
- Los clusters de Kubernetes tienen un servicio responsable de la resolución de DNS. A cada servicio en un cluster se le asigna un nombre de dominio como my-service.my-namespace.svc.cluster.local.
- Los pods reciben automáticamente un nombre DNS y también pueden especificar el suyo propio usando el nombre de host y las propiedades del subdominio en su configuración YAML.
- Entonces, cuando se realiza una solicitud a un servicio a través de su nombre de dominio, el servicio DNS la resuelve en la dirección IP del servicio.
- Luego, kube-proxy convierte la dirección IP de ese servicio en una dirección IP de pod. Después de eso, según si los pods están en el mismo nodo o en diferentes nodos, la solicitud sigue una de las rutas explicadas anteriormente.

03

ClusterIP, NodePort, LoadBalancer y ExternalName

Tipos de **services**

Hay cuatro tipos de *services* de Kubernetes: ClusterIP, NodePort, LoadBalancer y ExternalName. La propiedad ***type*** en la especificación de *service* determina cómo se expone el servicio a la red.



ClusterIP

- ClusterIP es el tipo de servicio predeterminado y más común.
- Kubernetes asignará una dirección IP interna del cluster al servicio ClusterIP. Esto hace que el servicio solo sea accesible dentro del cluster.
- No puede realizar solicitudes de servicio (pods) desde fuera del clúster.
- Opcionalmente, puede configurar la IP del cluster en el archivo de definición del servicio.

Casos de uso

Comunicación entre servicios dentro del cluster. Por ejemplo, la comunicación entre los componentes front-end y back-end de su aplicación.

NodePort

- Es una extensión del servicio ClusterIP. Se crea automáticamente un servicio ClusterIP, al que se enruta el servicio NodePort.
- Expone el servicio fuera del clúster al agregar un puerto para todo el custer encima de ClusterIP.
- NodePort expone el servicio en la IP de cada nodo en un puerto estático (el NodePort). Cada nodo representa ese puerto en su servicio. Entonces, el tráfico externo tiene acceso al puerto fijo en cada nodo. Significa que cualquier solicitud a su cluster en ese puerto se reenvía al servicio.
- Puede ponerse en contacto con el Servicio NodePort, desde fuera del cluster, solicitando <NodeIP>:<NodePort>.
- El puerto del nodo debe estar en el rango de 30000–32767. La asignación manual de un puerto al servicio es opcional. Si no está definido, Kubernetes asignará uno automáticamente.

Casos de uso

Cuando deseen habilitar la conectividad externa a su servicio.

LoadBalancer

- Es una extensión del servicio NodePort. Los servicios NodePort y ClusterIP, a los que se enruta el equilibrador de carga externo, se crean automáticamente.
- Integra NodePort con balanceadores de carga basados en la nube. Expone el Servicio externamente utilizando el balanceador de carga de un proveedor de la nube.
- Cada proveedor de la nube (AWS, Azure, GCP, etc.) tiene su propia implementación de equilibrador de carga nativo. El proveedor de la nube creará un equilibrador de carga, que luego enrutará automáticamente las solicitudes a su servicio de Kubernetes.
- El tráfico del balanceador de carga externo se dirige a los pods de backend. El proveedor de la nube decide cómo se equilibra la carga. La creación real del equilibrador de carga ocurre de forma asíncrona.

Casos de uso

Cuando utiliza un proveedor de nube para alojar su cluster de Kubernetes. E*ste tipo de servicio suele depender en gran medida del proveedor de la nube.*

ExternalName

- Los servicios de tipo ExternalName asignan un servicio a un nombre DNS, no a un selector típico como my-service. Estos servicios se especifican con el parámetro `spec.externalName`. Asigna el Servicio al contenido del campo externalName (por ejemplo, foo.bar.example.com), devolviendo un registro CNAME con su valor. Además, no se establece delegación de ningún tipo.

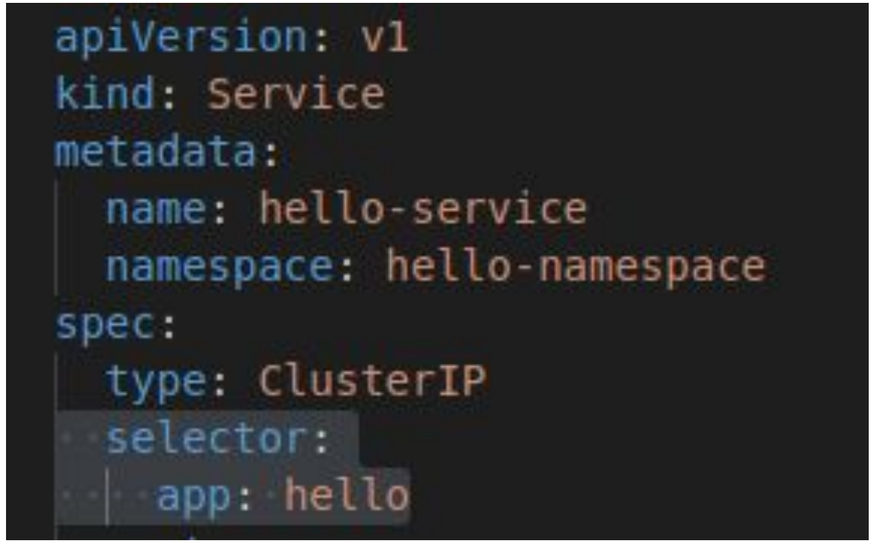
Casos de uso

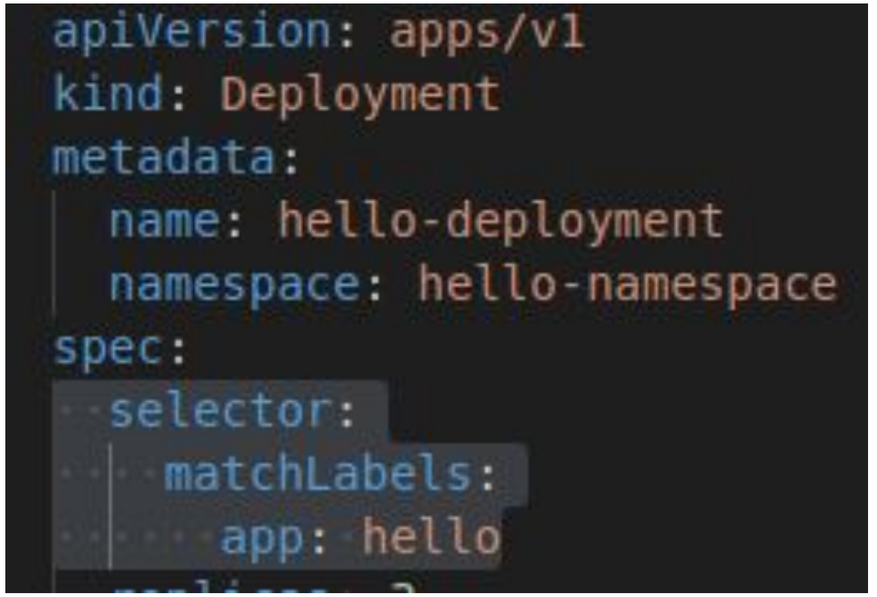
Se usa comúnmente para crear un servicio dentro de Kubernetes que represente un almacén de datos externo, como una base de datos que se ejecuta externamente a Kubernetes.

Se puede utilizar el servicio ExternalName (como un servicio local) cuando los pods de un espacio de nombres se comunican con un servicio en otro espacio de nombres.

Parámetro Selector

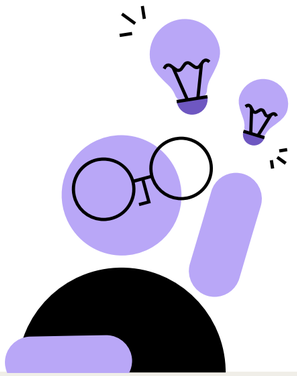
El envío de peticiones desde los services hacia los Pods se controla mediante el parámetro ‘selector’. Tendrá que contener un valor de tipo clave-valor que debe ser el mismo tanto en el service como en los pods que reciben el tráfico. En la imagen que se observa arriba, el service contiene el selector **app**: **hello**. Debajo, en el Deployment dentro del bloque ‘spec’, se encuentra el selector **app**: **hello** contenida dentro de ‘matchLabels’.





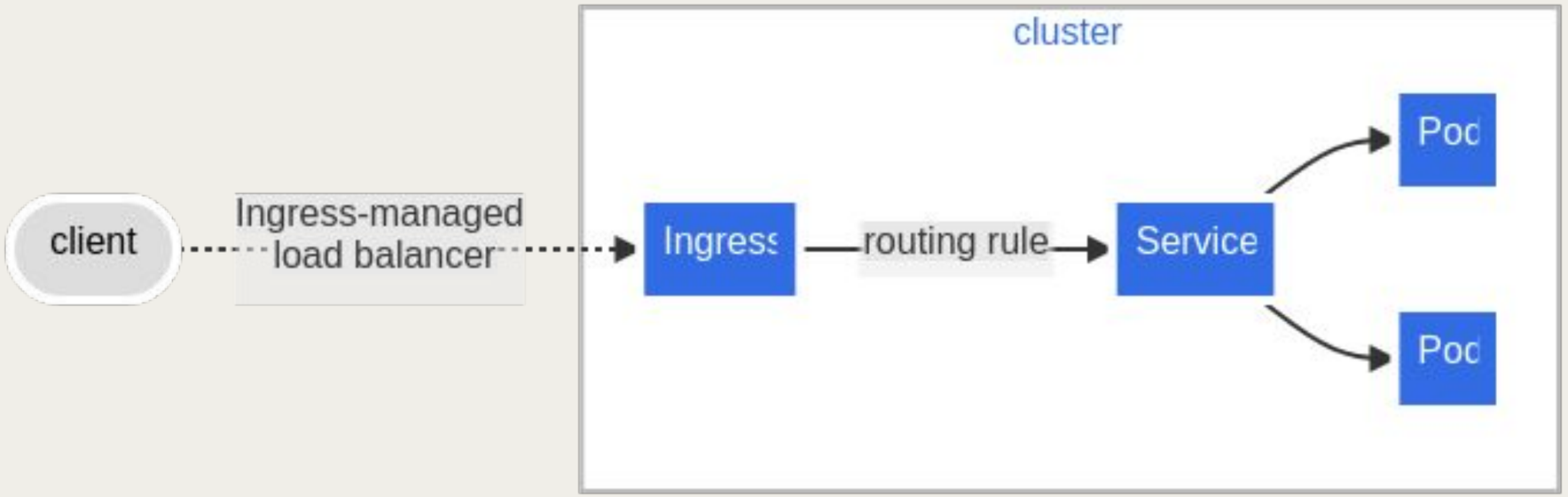
01 Ingress

¿Qué es?



Ingress expone las rutas HTTP y HTTPS desde fuera del cluster a los servicios dentro del cluster.
El enrutamiento del tráfico está controlado por reglas definidas en el recurso Ingress.

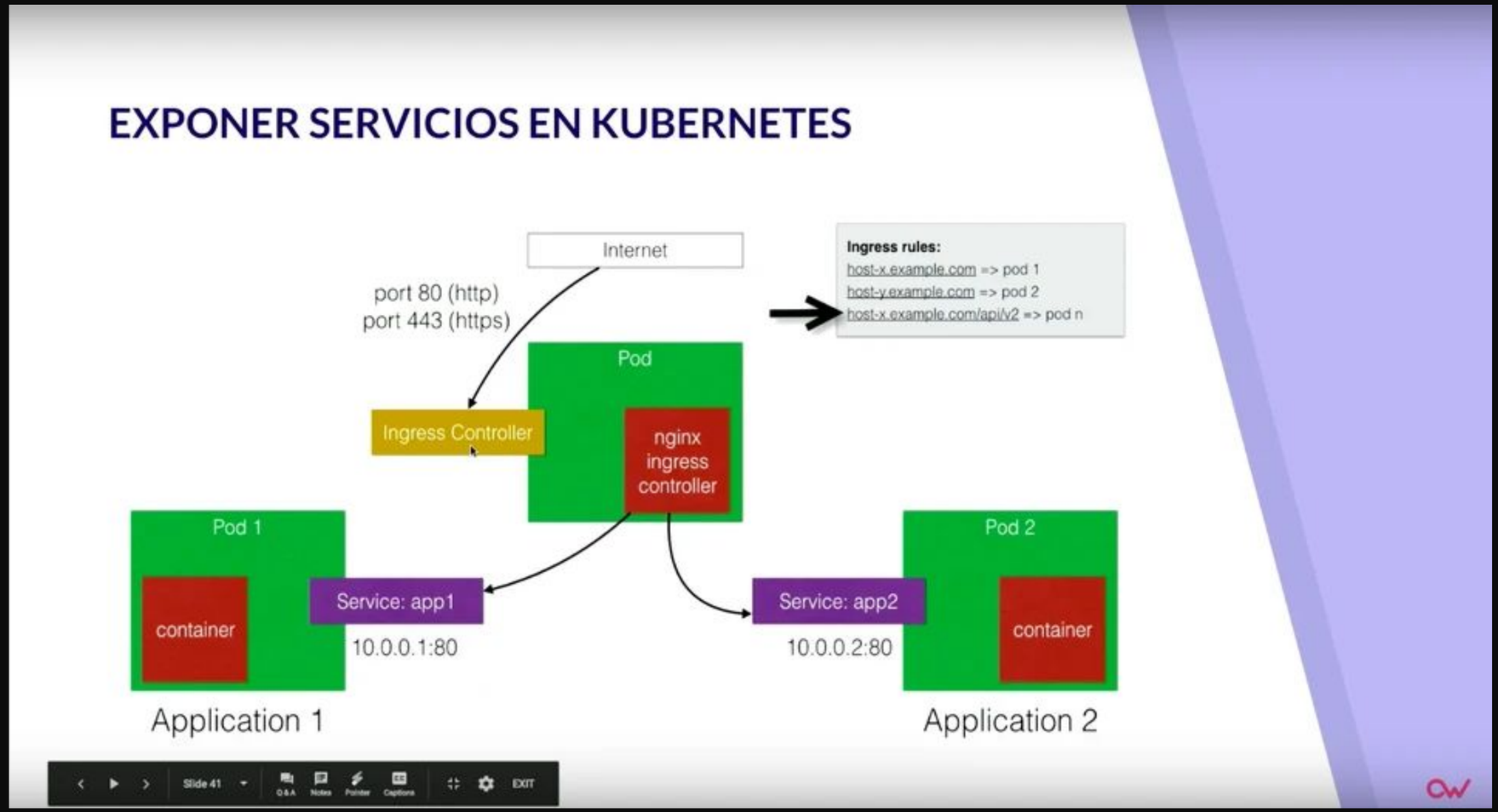
Aquí hay un ejemplo donde un Ingress envía todo su tráfico a un Servicio:



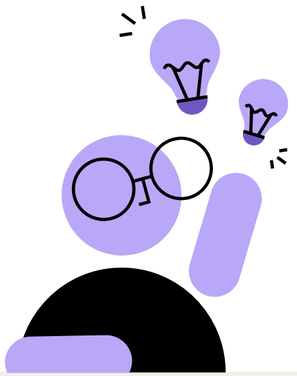
Ingress controller

- Como ya hemos visto, nuestros servicios corriendo dentro de un cluster pueden estar accesibles de manera privada dentro del mismo, o puede que necesitemos exponerlos a internet.
- Para exponerlos a internet hay varias maneras de hacerlo, se puede exponer un puerto en las máquinas del cluster o con un Load Balancer. Una manera muy potente y sencilla es hacerlo utilizando Ingress Controllers.
- Un Ingress Controller es un proxy, en este caso veremos un ejemplo con nginx, que nos va a permitir, utilizando el domain de la request que está llegando al Ingress Controller, redirigir esa request a distintos pods dentro de nuestro cluster.

Ingress controller



Ingress controller



En este caso vemos que tenemos un Ingress Controller, de tipo nginx, escuchando internet en el puerto 80 y el puerto 443 con certificado HTTPS, y es el que va a recibir todo el tráfico público que entre en el cluster.

En función del dominio con el que se hace la petición, vamos a definir reglas dentro del nginx controller que redirijan a un pod u a otro según el host del que provengan.

DIGITALHOUSE

¡Muchas gracias!