Kubernetes

3. Service & Ingress

Índice

Objetivos

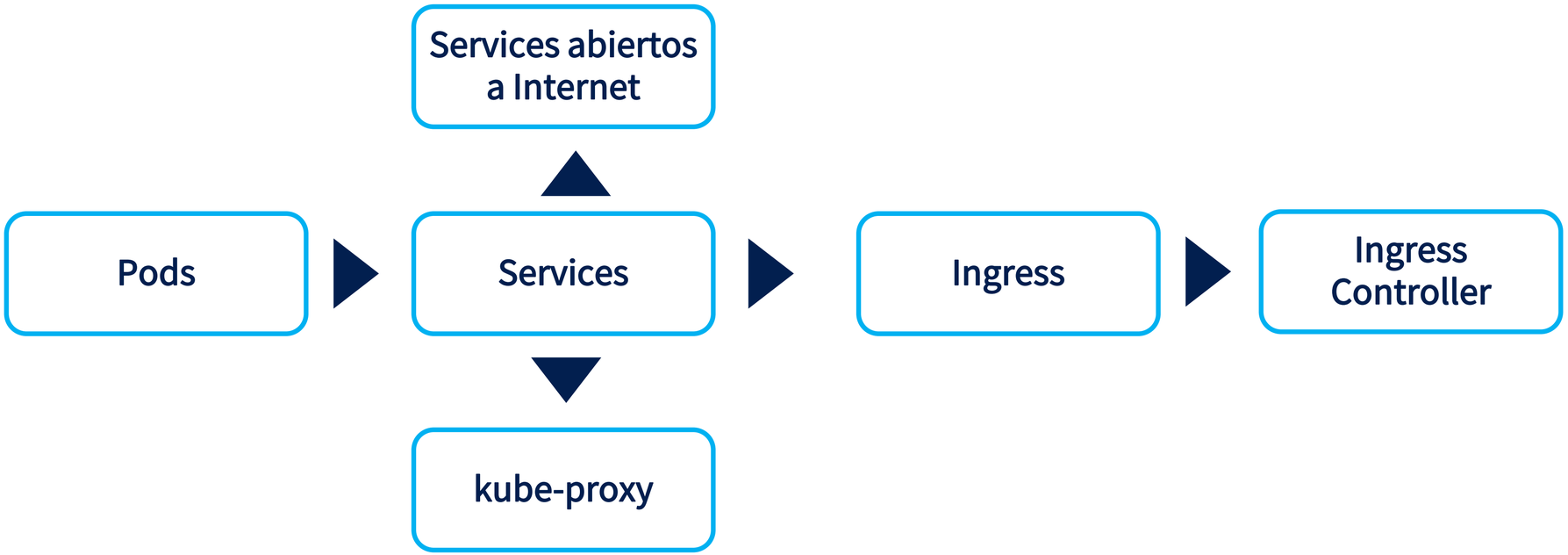
Conocer qué es un service

Ver cómo exponer servicios a Internet con services

Explorar kube-proxy

Conocer el objeto ingress y qué es un ingress controller

Mapa Conceptual

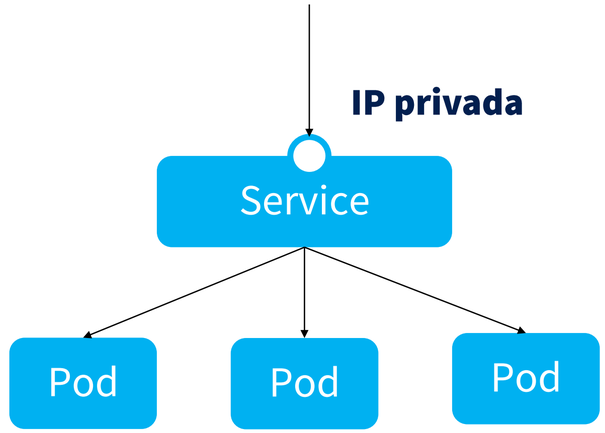


Services

En Kubernetes, un los pods que estén en estado Ready (por eso es importaService es un **balanceador de carga**, que reparte las peticiones entre todas las réplicas disponibles.

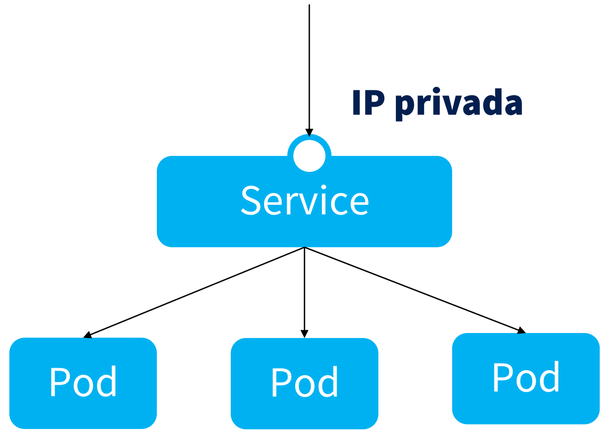
El service proporciona una IP privada como endpoint para este balanceador.

Solo direccionará peticiones hacia nte hacer uso de las ReadinessProbes que vimos al final del módulo 2).



Services

```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

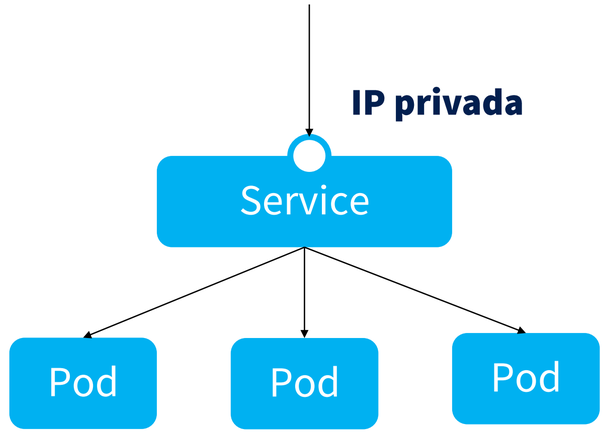


Services

apiVersion es v1 ya que el service es como el pod, un objeto básico en Kubernetes

Dentro del spec tenemos:

- selector: similar al campo selector que vimos en los deployments. Contiene una lista de labels con los que filtra los pods entre los que balanceará
- ports: Una lista de puertos que exponer y a los que conectarse.
  - protocol: TCP o UDP
  - port: El puerto que abrirá el service en la IP privada que crea
  - targetPort: El puerto de los pods al que reenviará las peticiones. Este puerto tiene que existir en la spec de los pods.



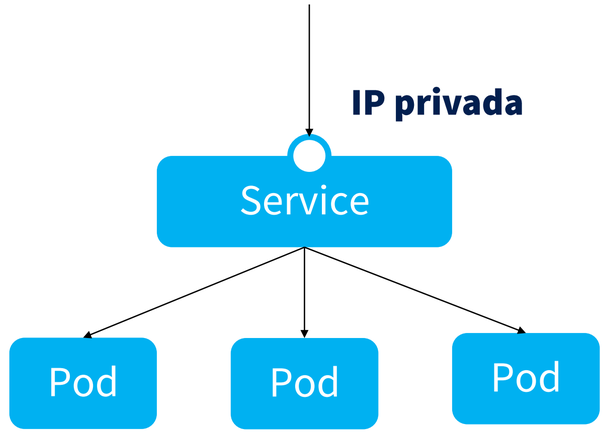
Services

Desplegamos este service en el entorno de prueba:

https://www.katacoda.com/courses/kubernetes/playground

Recuerda pulsar el botón launch.sh para lanzar el clúster de Kubernetes.

Necesitaremos crear un deployment para desplegar pods y el service



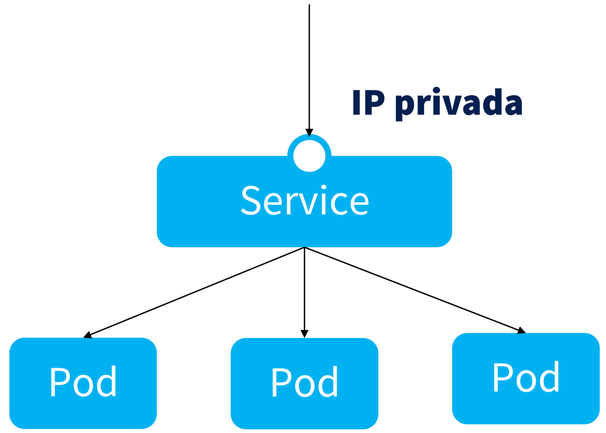
Services

Creamos los archivos para el deployment y el service

nano deployment.yaml
nano service.yaml

Y los enviamos a Kubernetes:

kubectl apply -f deployment.yaml service.yaml



Services

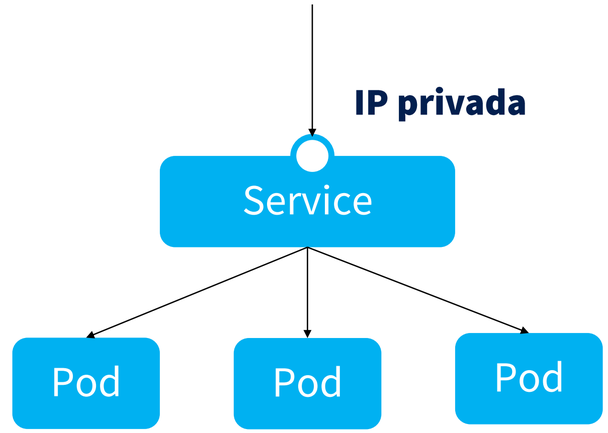
Con este comando podemos listar los services y obtener su IP

```
kubectl get services
```

Una vez los pods hayan pasado a estado Running, podemos hacer curl a la IP del service

```
curl http://<IP>:80
```

Si hacemos varias peticiones y observamos los logs de los distintos pods, vemos que se han repartido entre los 3 pods.



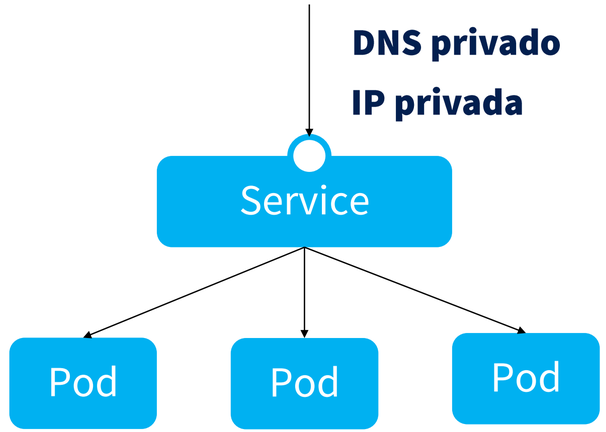
Services

Vemos que podemos utilizar la IP para conectarnos a los pods sin preocuparnos de conocer la IP específica de cada pod, que además cambia cada vez que se crea uno nuevo. De esta forma podemos conectarnos a nuestra aplicación con una IP estable.

Pero hay otra manera mejor de conectarnos con nuestro service.

Kubernetes nos da una dirección DNS fija para cada service desplegado en el clúster.

Con esa dirección, ni siquiera necesitamos conocer la IP del service. Sólo sabiendo su nombre podemos conectarnos a él.



Services

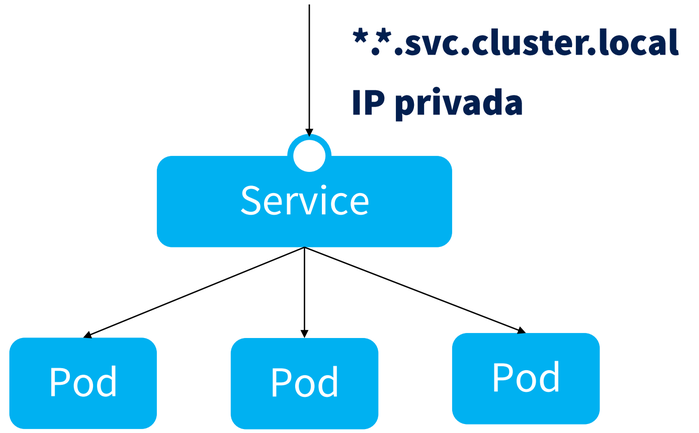
<Nombre del Service>.<Namespace>.svc.cluster.local

Por ejemplo:

nginx.default.svc.cluster.local

Esta dirección sólo funciona dentro de los pods. Kubernetes tiene un servidor DNS propio que crea estos nombres. Ese servidor DNS viene configurado por defecto en todos los pods.

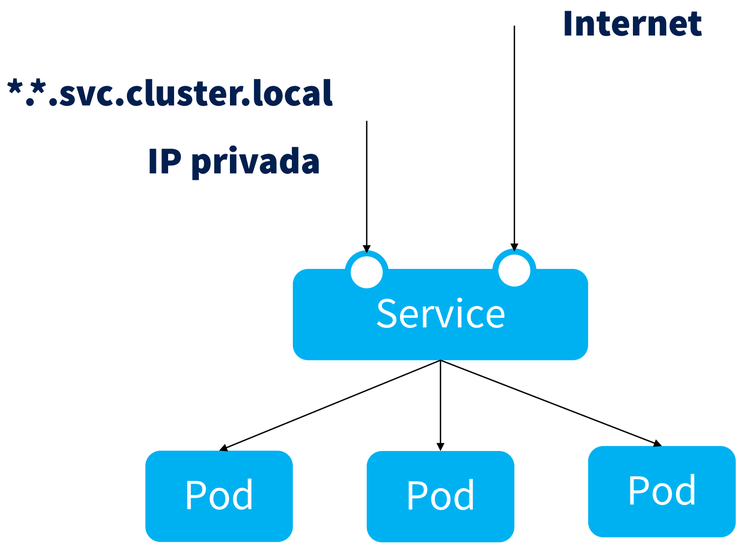
cluster.local es el dominio para los nombres internos de Kubernetes. Siempre que lo veamos, llevará a un servicio de nuestro clúster.



Services

Los services ofrecen más posibilidades que simplemente balancear entre las distintas réplicas de nuestra aplicación. Un service es la forma que tenemos de abrir la aplicación a Internet, fuera del clúster.

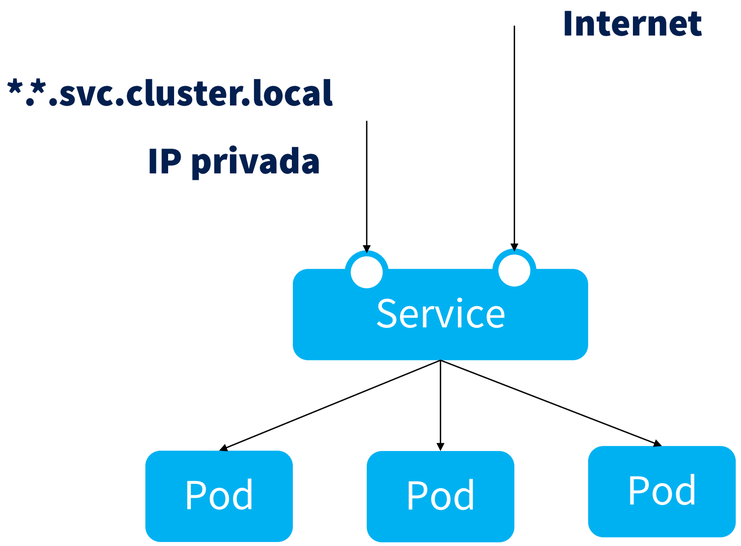
Hasta ahora sólo hemos visto servicios que sólo crean una IP privada. Pero hay distintos tipos de servicios que, aparte de esa IP privada, pueden abrir al conectividad a Internet.



Services

Podemos controlar qué clase de service estamos creando con el parámetro type:

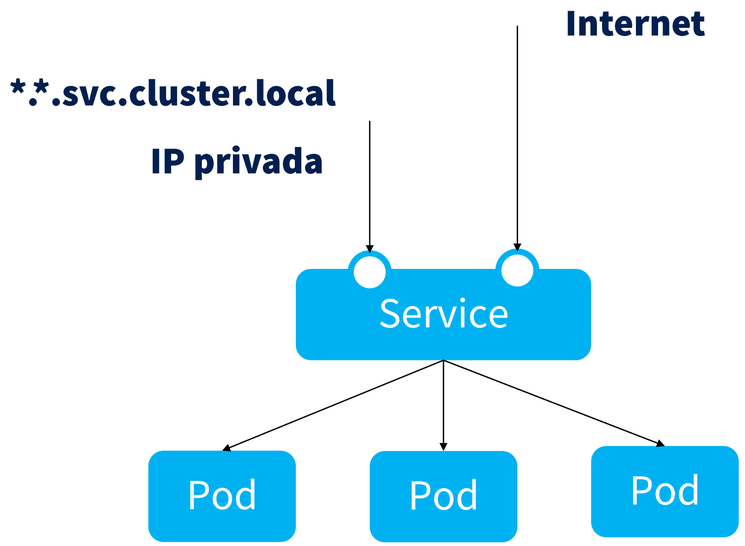
```
apiVersion: v1
kind: Service
metadata:
  name: nginx
spec:
  type: ClusterIP
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```



Services

type puede ser uno de estos 3 valores:

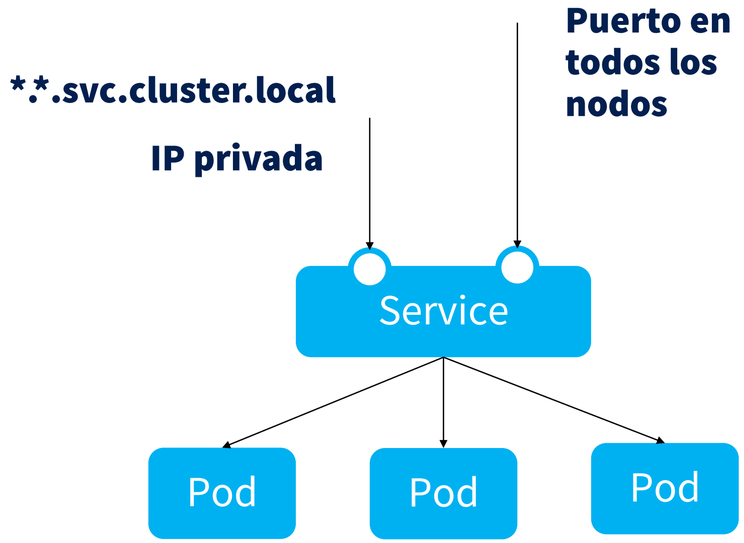
1. ClusterIP: es el valor por defecto. El que hemos visto hasta ahora. Sólo crea una IP privada
2. NodePort: nos permite exponer nuestro service fuera del clúster. Lo hace abriendo **un puerto en todos los nodos** del clúster. Todo el tráfico que llega a ese puerto en cualquier nodo, lo redirige hacia nuestros pods
3. LoadBalancer: Es un tipo especial, sólo disponible en algunos proveedores cloud, pero enormemente útil. Aparte de abrir un puerto como en NodePort, **crea un balanceador en nuestro cloud** automáticamente, que se dirige a ese puerto. De esta forma tenemos un método sencillo y seguro de exponer nuestras aplicaciones



Services

Veamos el tipo NodePort:

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



Services

Después de crear el service puedes ver su NodePort con

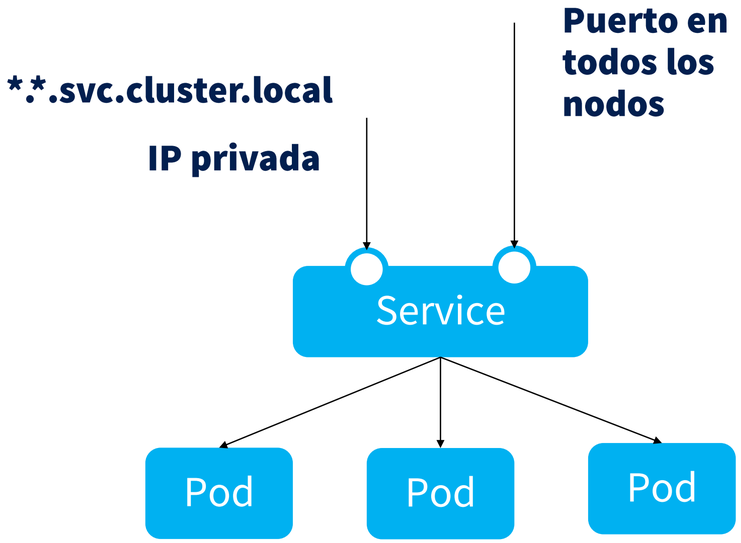
```
kubectl get services
```

Este puerto está abierto en todos los nodos, prueba en ambos a hacer un curl a si mismo a ese puerto

```
curl http://127.0.0.1:<NodePort>
```

Verás que te están respondiendo las réplicas de Nginx

No importa en qué nodo se estén ejecutando los contenedores. Kubernetes redireccionará el tráfico que llegue a cualquiera de ellos.



Services

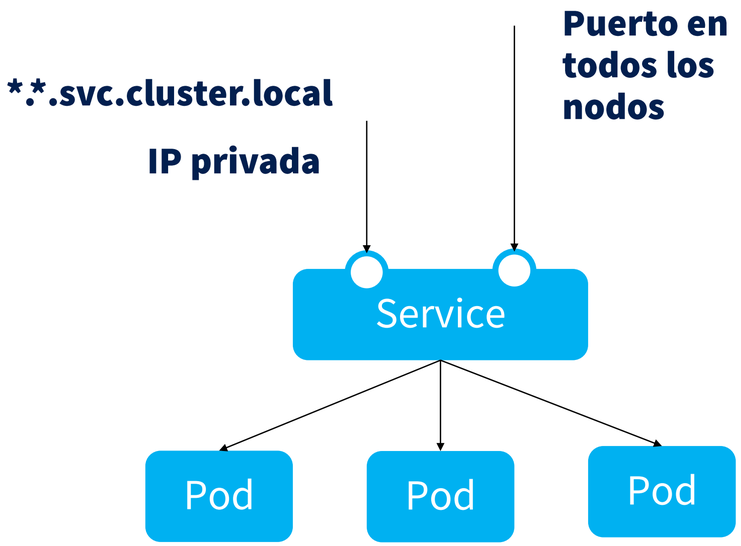
Esta es la manera más sencilla de abrir los pods a clientes que no están dentro de nuestro clúster.

El comportamiento del tráfico que llega a los NodePorts es exactamente igual que el que llega a la IP privada. Es balanceado de la misma manera entre todas las réplicas.

Vamos a ver otro comando de Kubernetes:

kubectl describe service nginx

Con él obtienes todos los detalles del service. Observa el campo réplicas, en el que efectivamente están las Ips de todos los pods entre los que está balanceando el tráfico.



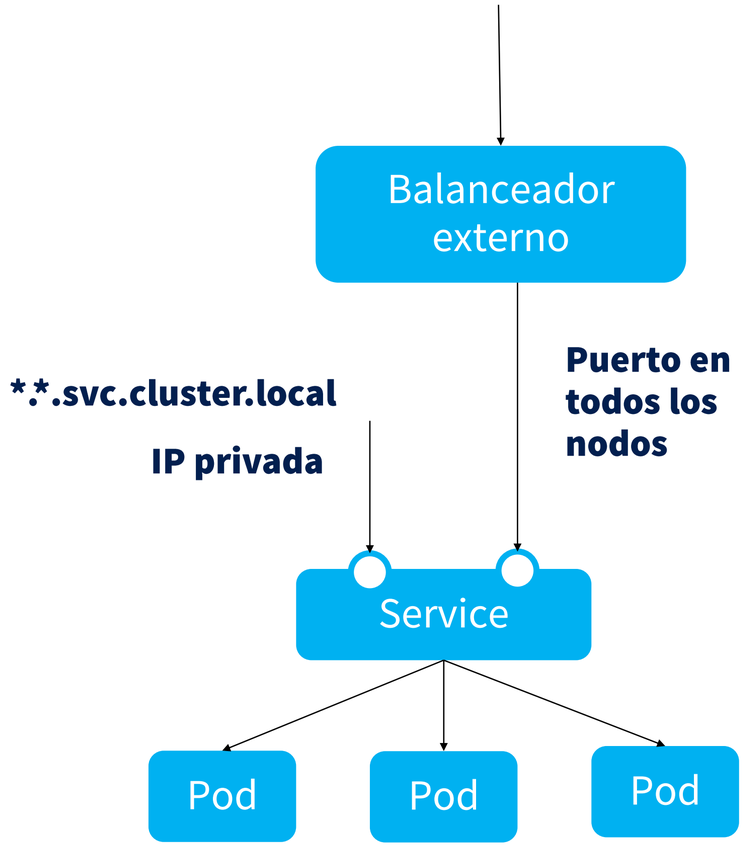
Services

El tipo LoadBalancer no es posible probarlo en este entorno de prueba. Sólo está disponible en algunos proveedores cloud.

Lo veremos en próximos módulos.

Simplemente es necesario saber que el tipo LoadBalancer también crea la IP privada y el abre el NodePort. Aparte de todas estas cosas crea el balanceador de carga externo, que se dirige al NodePort de cualquiera de los nodos.

Es más seguro ya que con él no necesitamos abrir los nodos a Internet, sino que estos pueden tener sólo IPs privadas y el balanceador externo es la única pieza abierta.



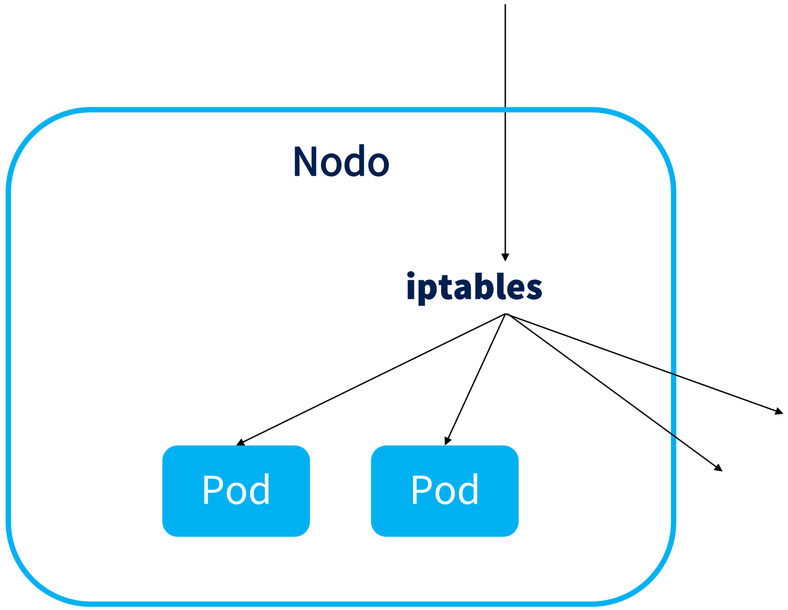
iptables

¿Cómo crea Kubernetes las IPs privadas de los services y abre los NodePorts en todos los nodos?

Todo este redireccionamiento y balanceo se realiza en el iptables de cada nodo.

iptables es el cortafuegos de Linux. Todo el tráfico que llega a un máquina Linux pasa primero por iptables, que puede permitir o no ese tráfico, modificarlo, redireccionarlo, etc.

Por suerte para nosotros, Kubernetes se encarga de configurar las reglas en iptables automáticamente. No necesitamos saber ni siquiera que existe.

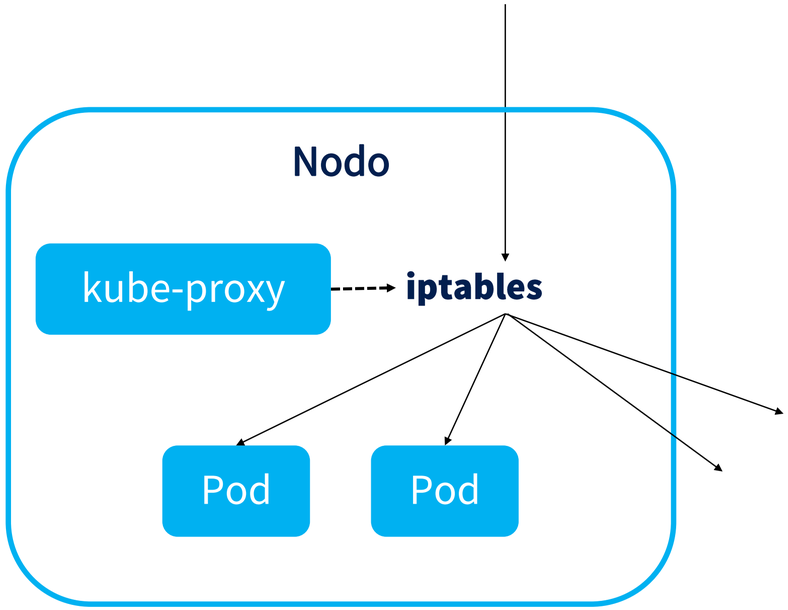


kube-proxy

kube-proxy es la pieza de Kubernetes que se encarga de configurar iptables en todos los nodos.

kube-proxy es un pod que debe estar levantado en todos los nodos para que el resto de pods que corren en el nodo puedan conectarse a services y NodePorts.

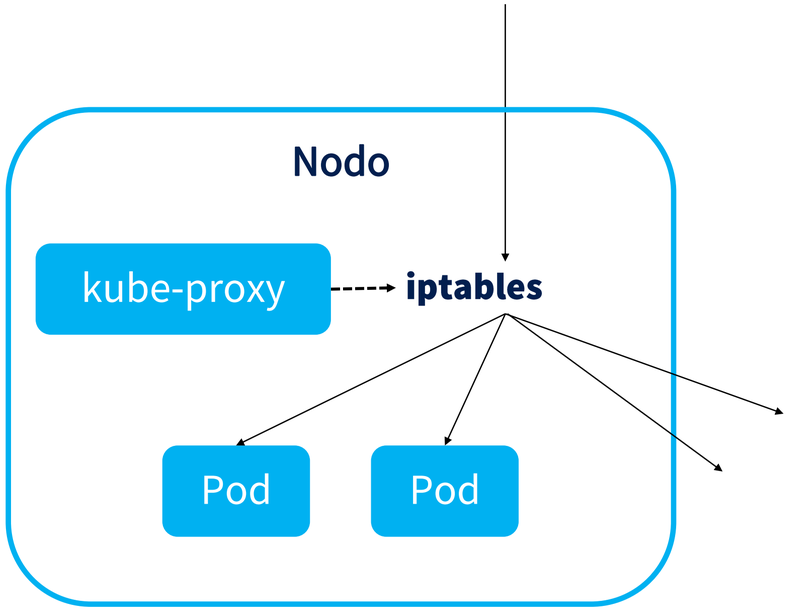
¿Un pod que debe levantarse obligatoriamente en todos los nodos? Efectivamente, es el caso perfecto para una carga de trabajo de tipo DaemonSet.



kube-proxy

kube-proxy configura automáticamente todo tipo de reglas en el iptables de cada nodo para abrir services en el clúster. Lo más importante es:

1. Reglas para las IPs privadas de los services: Crea una regla para que todo el tráfico que llegue destinado a una IP de un service, se redirija y balancee hacia las IPs de cada pod de ese service. Estén en el nodo que estén estos pods.
2. Reglas para los NodePorts: Abre el puerto del NodePort y crea una regla para que todo el tráfico que llegue por ese puerto, sea reenviado igualmente a los pods de ese service.

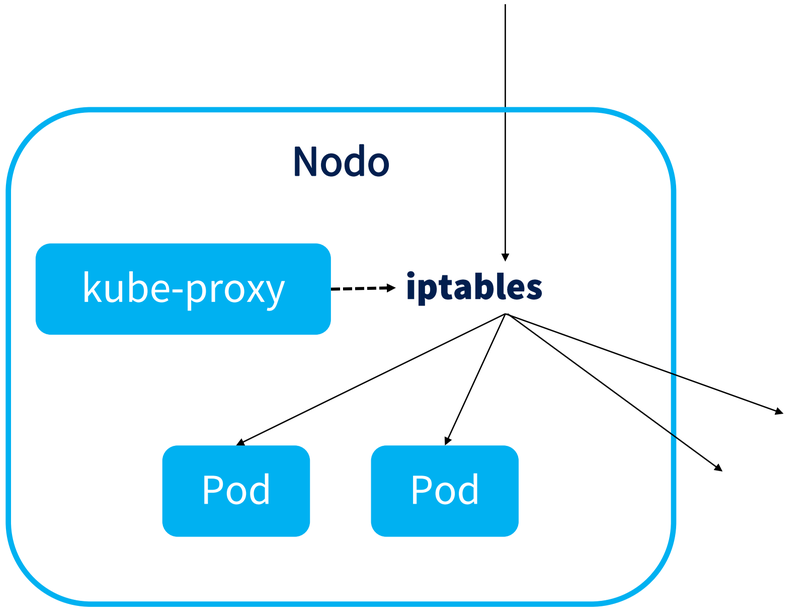


Namespaces

En el clúster del entorno de prueba hay un DaemonSet que ha levantado el kube-proxy en todos los nodos.

No lo hemos visto cuando hemos desplegado nuestro DaemonSet en el anterior módulo, porque kube-proxy está en un namespace distinto.

Los Namespaces en Kubernetes son una forma de separar nuestras aplicaciones para aislarlas mejor. Son algo simbólico, no implica que los pods de distintos Namespaces se levanten en nodos distintos. Es una forma de organizar los objetos en Kubernetes.



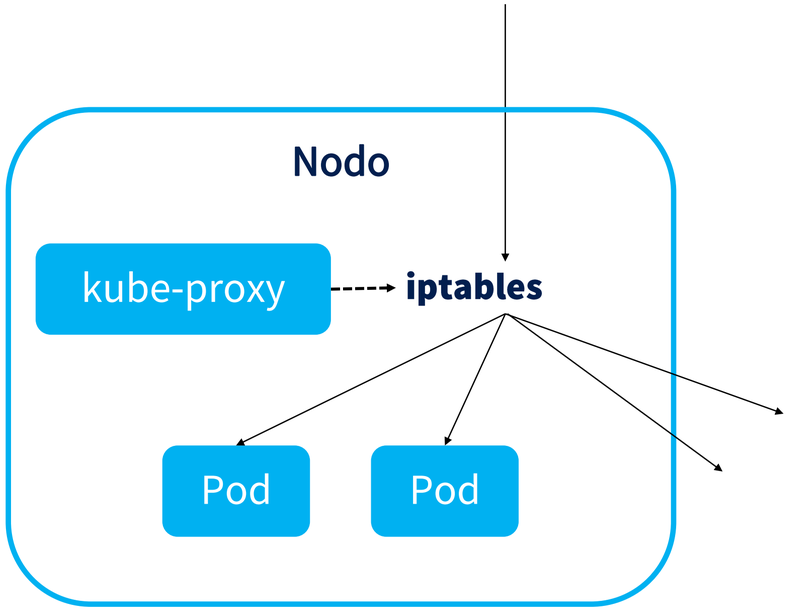
Namespaces

Todo este tiempo hemos estado utilizando el namespace default, en el que se crean todos los objetos si no especificamos su namespace.

Pero existe otro namespace en nuestro clúster: kube-system. En él están todos los objetos que el clúster necesita para funcionar. Entre ellos, un DaemonSet con kube-proxy.

Puedes listar todos los Namespaces con:

```
kubectl get namespaces
```



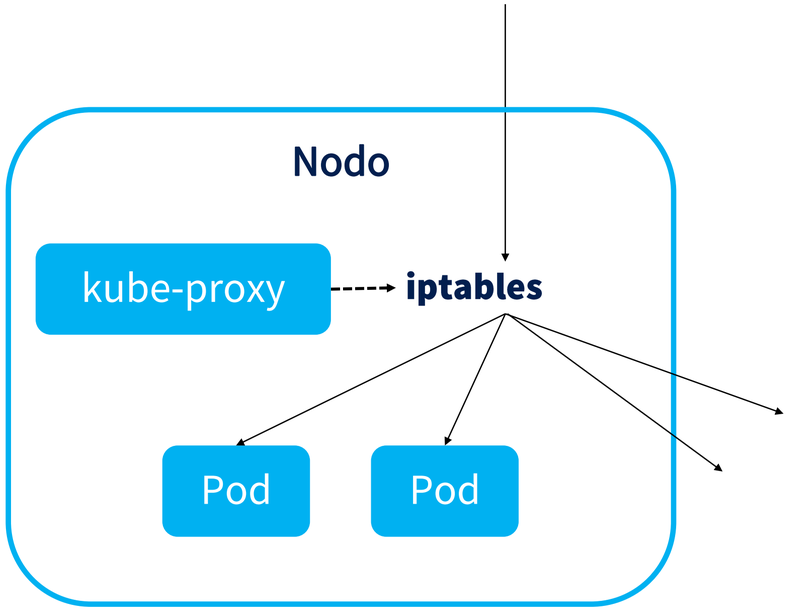
kube-proxy

Todos los comandos que hemos visto son válidos en todos los Namespaces. Para especificar en qué namespace queremos que se ejecute, sólo hay que añadir la opción -n

```
kubectl get daemonsets -n kube-system
```

De esa forma listamos todos los daemonset que hay en el namespace kube-system. Entre ellos, vemos efectivamente el kube-proxy, que tiene 2 pods, uno en cada nodo.

Veremos todos los demás objetos de kube-system a lo largo del curso.

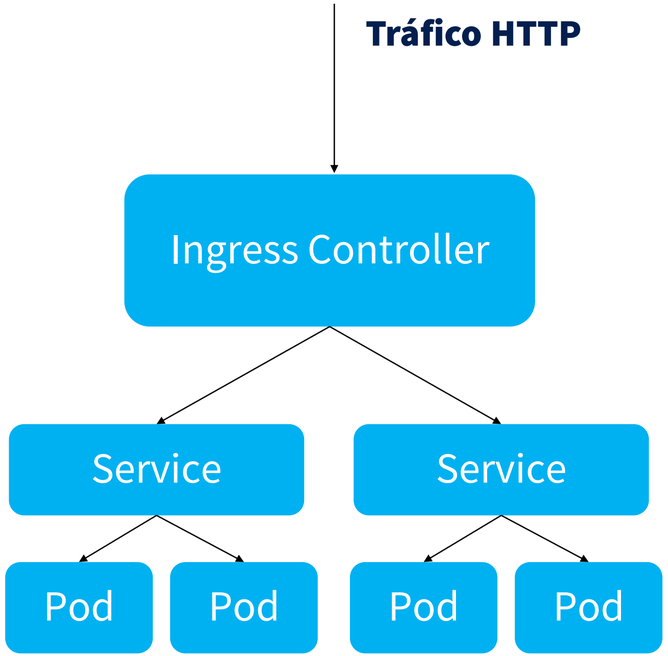


Ingress

Los Ingress son otro tipo de objeto de red de Kubernetes.

Kubernetes está pensado para desplegar muchas aplicaciones distintas en el mismo clúster. Abrir todas esas aplicaciones a Internet mediante NodePorts o LoadBalancers sería una tarea inmensa.

Para eso están los ingress. Gracias a ellos podemos tener un único punto de entrada al clúster. Ese punto de entrada único es el Ingress Controller.



Ingress

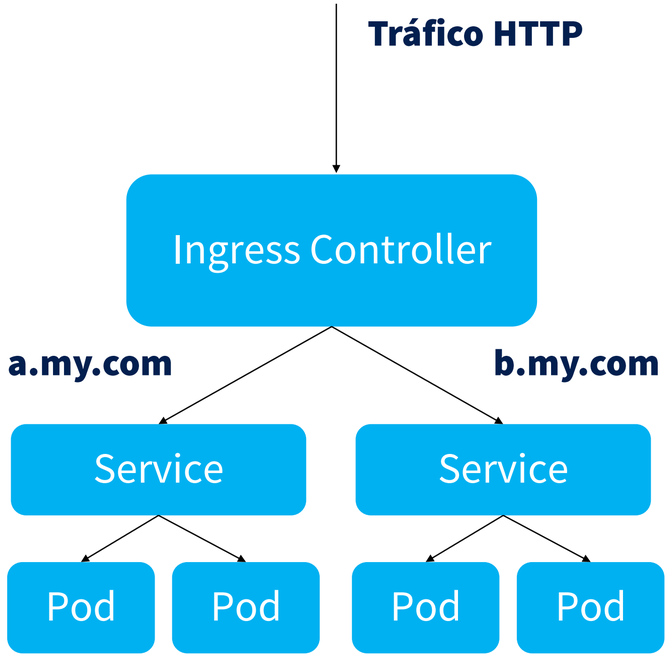
Un Ingress Controller es simplemente un router HTTP, que dirige cada petición al service correcto.

Los ingress son las reglas para configurar ese enrutado.

Por ejemplo podemos enrutar dependiendo del dominio de esa petición, del subdominio, de los headers, etc.

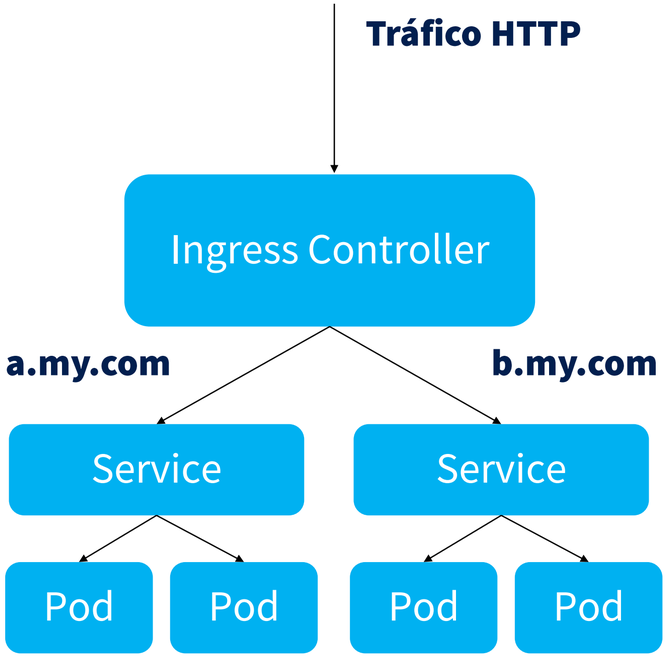
Entonces tenemos un único servicio expuesto a Internet, el ingress controller (el ingress controller es también uno o varios pods con su service).

Y por cada aplicación del clúster, tendremos un objeto Ingress que configure el enrutado de peticiones hacia ella.



Ingress

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: my-nginx
spec:
  rules:
  - host: nginx.my.com
    http:
      paths:
      - backend:
          serviceName: nginx
          servicePort: 80
```



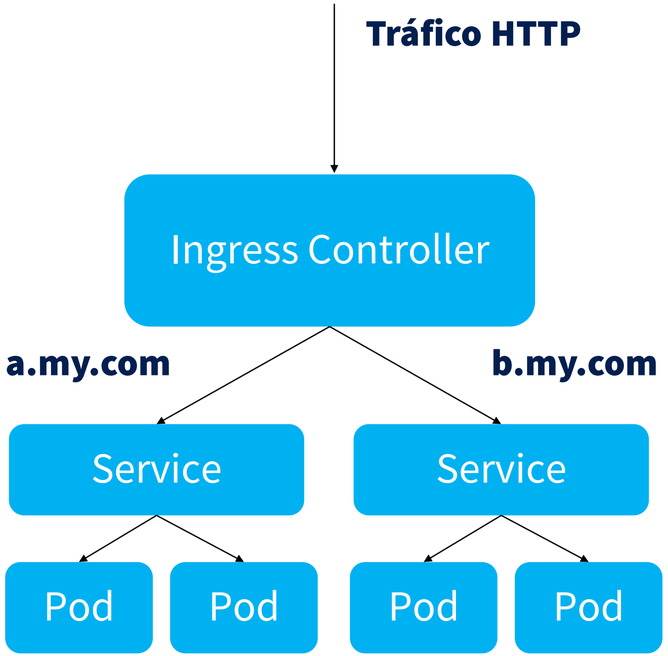
Ingress

Este ingress configuraría el ingress controller para que todas las peticiones dirigidas al host nginx.my.com se direccionaran hacia nuestro service.

Es posible añadir tantas reglas dentro del campo rules como queramos.

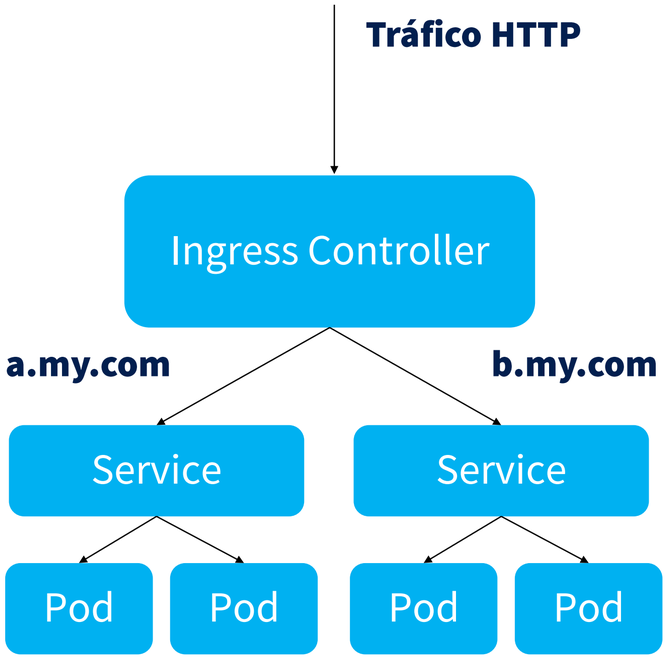
Además dentro de cada regla podemos filtrar por host como en el YAML anterior, pero también podemos diferencias distintos paths dentro del mismo host.

Por ejemplo nginx.my.com/v1 y nginx.my.com/v2 pueden dirigirse a services distintos



Ingress

```
rules:
  - host: nginx.my.com
    http:
      paths:
      - path: /v1
        backend:
          serviceName: nginx
          servicePort: 80
      - path: /v2
        backend:
          serviceName: other-nginx
          servicePort: 80
```

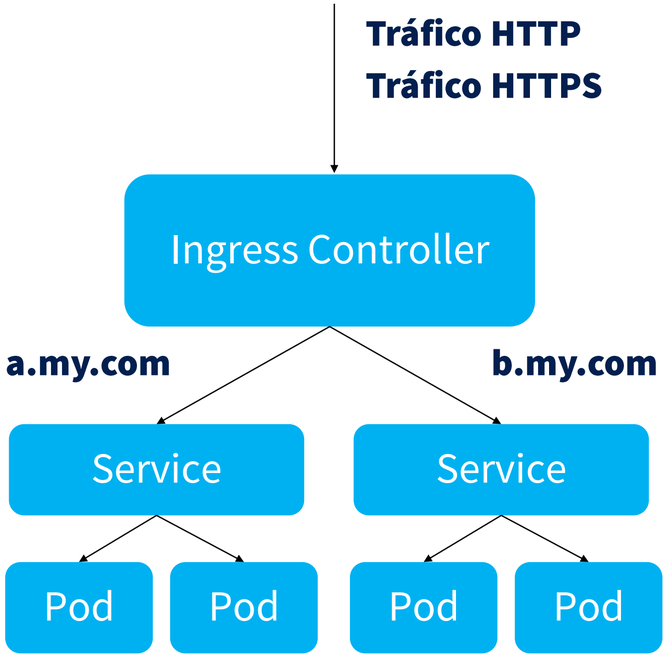


Ingress

Por último, es posible, y de hecho muy recomendable, añadir en el ingress certificados TLS para cada dominio, de forma que el ingress controller haría la terminación TLS de forma automática para todo el tráfico HTTPS. Absteniendo a nuestros pods de realizar esa tarea.

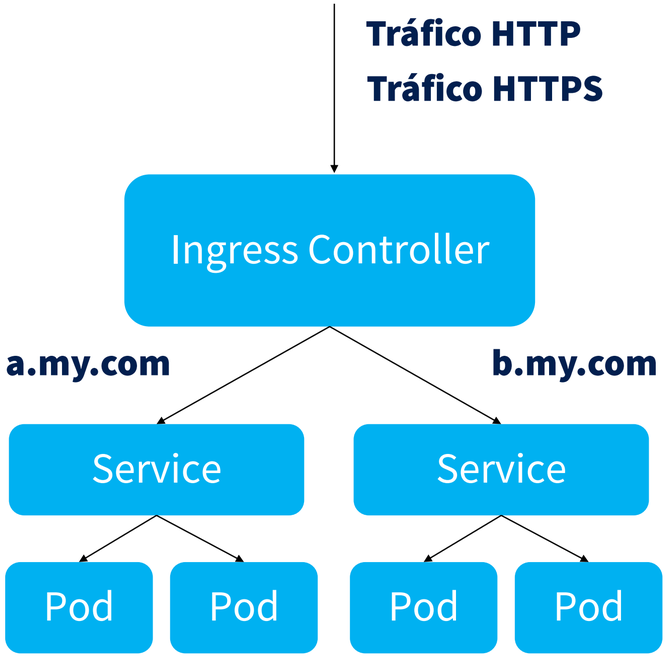
En el spec sólo necesitamos poner los nombres de dominio que deseamos que tengan terminación TLS, junto con el nombre de un objeto secret en el que se encuentra almacenado ese certificado.

Veremos qué son los secrets a fondo en el siguiente módulo.



Ingress

```
spec:
  tls:
  - hosts:
    - nginx.my.com
    secretName: nginx-certificate-tls
  rules:
  - host: nginx.my.com
    http:
      paths:
      - backend:
          serviceName: nginx
          servicePort: 80
```

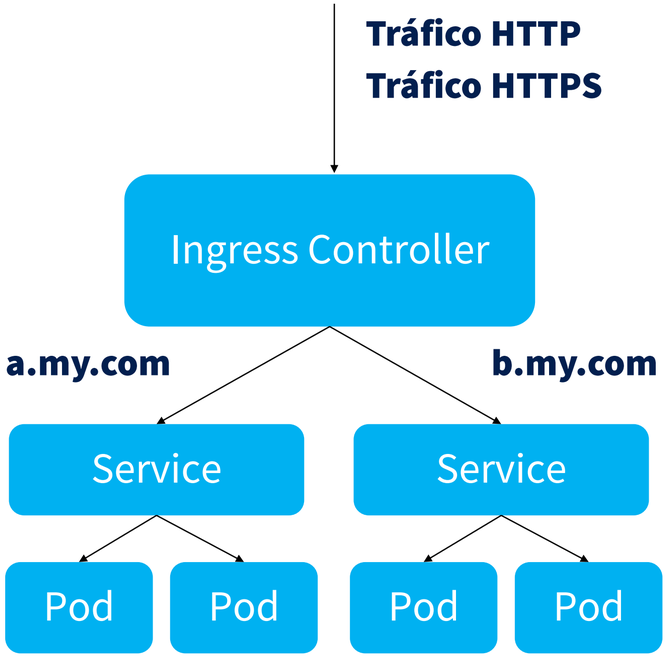


Ingress Controller

El ingress controller no es una pieza que venga por defecto en Kubernetes. Es algo que debemos desplegar en el clúster nosotros mismos.

Existen varios ingress controller para Kubernetes basados en distintas tecnologías. Se despliegan como cualquier otro deployment o service que hayamos visto hasta ahora.

En la documentación de cada uno podemos encontrar los objetos YAML necesarios para desplegarlos. Éstos tendrán un Deployment, un service para abrir el ingress controller a Internet, y otros objetos de configuración que aún no hemos visto. Pero se desplegarán exactamente igual que cualquier YAML.

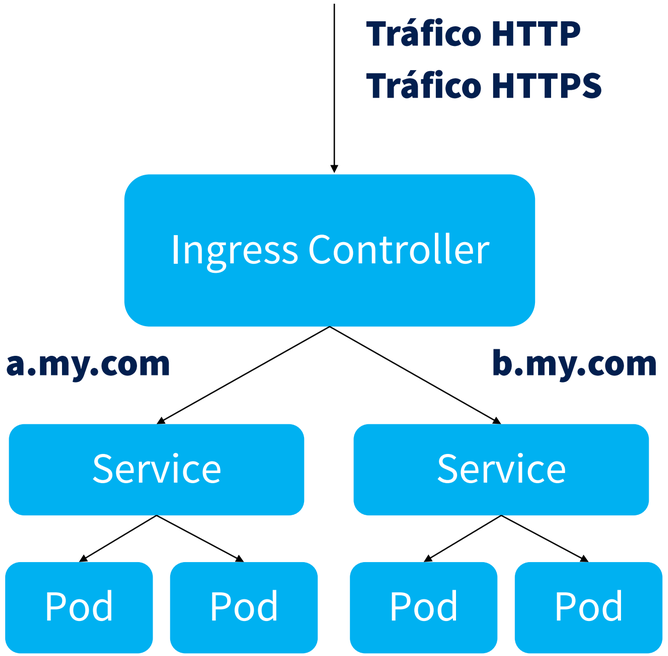


Ingress Controller

Los ingress controller más utilizados son:

- Nginx controller: basado en Nginx y mantenido dentro del proyecto oficial de Kubernetes
- Ambassador
- Traefik
- Gloo
- HA Proxy controller
- …

Además existen controllers específicos para cada proveedor cloud.



Ingress Controller

Vamos a desplegar el Nginx Ingress Controller. En la documentación oficial podéis encontrar instrucciones para desplegarlo:

https://kubernetes.github.io/ingress-nginx/deploy/

Vemos que para desplegar el ingress controller sólo necesitamos ejecutar:

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/nginx-0.30.0/deploy/static/mandatory.yaml

Deployment Ingress Controller

Ingress Controller

El comando puede parecer raro pero es un kubectl apply como el que hemos usado hasta ahora. Pero en lugar de pasarle un archivo YAML local, le pasamos la dirección a un YAML remoto que se encuentra en GitHub. Kubernetes lo descargará y lo aplicará.

Si abres la URL en el navegador verás que contiene muchos objetos distintos. Entre ellos, el primero es un objeto namespace, ya que el Nginx Ingress Controller se despliega en su propio namespace.

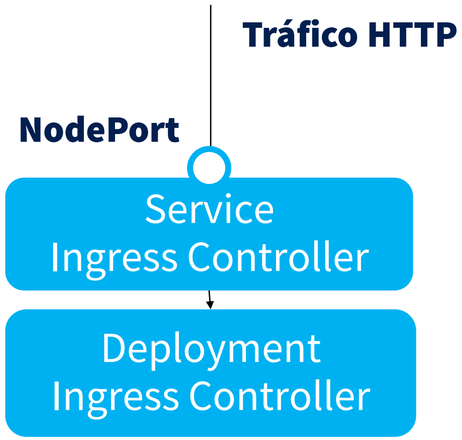
Y el último es un objeto Deployment que ya conocemos con la imagen de docker del ingress controller.

Los demás son necesarios pero no importantes por ahora.

Deployment
Ingress Controller

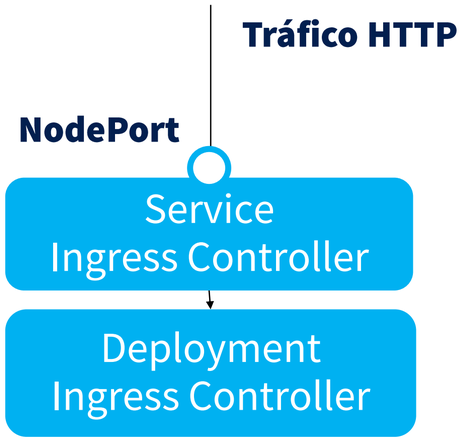
Ingress Controller

En el YAML no está el objeto service, ya que este depende de cómo quieras abrir el ingress controller al mundo. Nosotros lo haremos con un service NodePort.



Ingress Controller

```
apiVersion: v1
kind: Service
metadata:
  name: ingress-nginx
  namespace: ingress-nginx
spec:
  type: NodePort
  ports:
    - name: http
      port: 80
      targetPort: 80
      protocol: TCP
selector:
    app.kubernetes.io/name: ingress-nginx
```



Ingress Controller

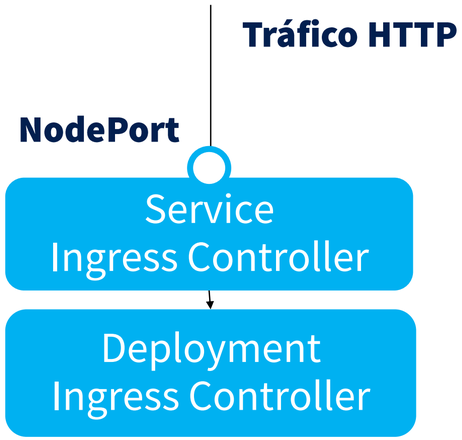
Creamos el service y ejecutamos este comando en el namespace ingress-nginx para ver su NodePort

kubectl -n ingress nginx get services

Podemos hacer curl a ese puerto con

curl http://127.0.0.1/<NodePort>

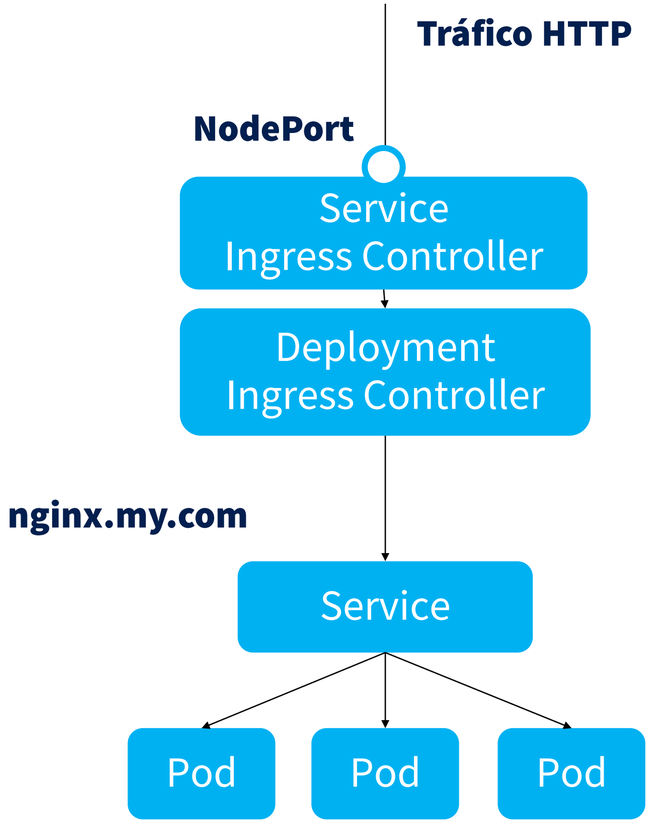
Nos devolverá un error 404, que es el mensaje por defecto del ingress controller cuando la petición no coincide con ninguna regla de ningún ingress.



Ingress Controller

Vamos a crear un ingress para nuestro Nginx. Recuerda que si no están desplegados el Deployment y el Service que hemos visto anteriormente, no va a funcionar.

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: my-nginx
spec:
  rules:
  - host: nginx.my.com
    http:
      paths:
      - backend:
          serviceName: nginx
          servicePort: 80
```



Ingress Controller

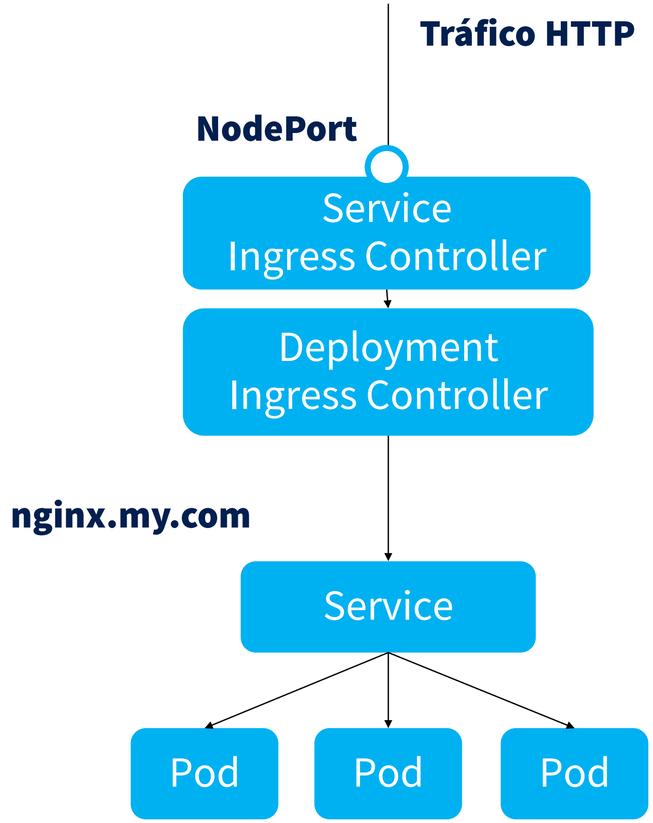
Ahora si hacemos de nuevo una petición al NodePort, y esta está dirigida a nginx.my.com, llegará a nuestros pods.

Como no controlamos el dominio nginx.my.com ni lo podemos dirigir a ninguno de nuestros nodos, tendremos que simularlo.

Podemos hacerlo añadiendo el header HTTP “Host: nginx.my.com” a nuestra petición:

curl -H "Host: nginx.my.com" 127.0.0.1:31238

Ya no recibiremos el error Not Found, sino que ¡la respuesta es la que ya conocemos de nuestros pods!



¡Fin del módulo!

En este módulo hemos visto:

- Qué es un service, el primer objeto de red que conocemos. Actúa como un balanceador entre las replicas de nuestra aplicación.
- Como crear services de tipo NodePort o LoadBalancer para abrirlos a Internet
- Cómo funcionan los services por dentro, gracias a kube-proxy
- Qué son los Namespaces en Kubernetes. Hemos visto que el kube-proxy se despliega en el namespace kube-system.
- Qué es un ingress controller, que actúa como punto de entrada para todo el tráfico HTTP de nuestro clúster
- Cómo crear objetos ingress para configurar reglas en el ingress controller, y que el tráfico HTTP llegue a nuestros pods.

Extras

Páginas de la documentación oficial:

Services
DNS for services
Namespaces
kube-proxy
Ingress
Ingress controller
Nginx ingress controller

¡Gracias!