

Kubernetes informe final

Índice

[Set-up del cluster](#)

[Creación del cluster](#)

[Despliegue de aplicaciones](#)

[Acceso a las aplicaciones](#)

[Configuración de ELK post instalación](#)

[Anexo](#)

[Funcionamiento interno de kubernetes](#)

[Capa ExternalCluster](#)

[Capa InternalCluster](#)

[La jerarquía](#)

[Funcionamiento de los distintos accés controllers](#)

[Ingress](#)

[NodePort](#)

[LoadBalancer](#)

[Proxy ClusterIP](#)

[Error de pantalla roja \(bad operation code\)](#)

Set-up del cluster

El cluster en cuestión está formado por 3 nodos que son hood03, hood05, hood06. En un principio hood03 se encontraba en estado de error pantalla roja, este error afectó a otros nodos y está detallado en el anexo. Cada nodo tiene instalado rancher OS y docker lo cual da pocos recursos pero ocupa poco espacio en los nodos proporcionando gran disponibilidad.

Para empezar se tuvo que realizar una limpieza de los residuos de rancher/rancher y RKE de el uso anterior que tuvieron los nodos. Para ello ejecutamos 2 scripts para limpiar los 2 tipos de archivos residuales diferentes.

Limpieza de procesos rancher/rancher:

```
#!/bin/sh
CLIST=$(docker ps -qa)
if [ "x"$CLIST == "x" ]; then
    echo "No containers exist - skipping container cleanup"
else
    docker rm -f $CLIST
fi
```

```

ILIST=$(docker images -a -q)
if [ "x"$ILIST == "x" ]; then
    echo "No images exist - skipping image cleanup"
else
    docker rmi $ILIST
fi

VLIST=$(docker volume ls -q)
if [ "x"$VLIST == "x" ]; then
    echo "No volumes exist - skipping volume cleanup"
else
    docker volume rm -f $VLIST
fi

```

Reiniciamos la máquina antes de ejecutar el segundo script:

```

#!/bin/sh
if [ $(id -u) -ne 0 ]; then
    echo "Must be run as root!"
    exit
fi

DLIST="/var/lib/etcd /etc/kubernetes /etc/cni /opt/cni /var/lib/cni /var/run/calico
/opt/rke"
for dir in $DLIST; do
    echo "Removing $dir"
    rm -rf $dir
done

```

Una vez limpios los nodos debemos establecer una persistencia de disco, ya que rancher solo nos permite aplicar cambios reiniciando la máquina, de esta forma no tenemos que añadir el nodo de nuevo al clúster cada vez que reiniciamos y los datos de las aplicaciones siguen intactos al reiniciar.

```

[root@hood03 ~]# fdisk /dev/sda
Welcome to fdisk (util-linux 2.31.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

```

The old ext4 signature will be removed by a write command.

Device does not contain a recognized partition table.
Created a new DOS disklabel with disk identifier 0xbd06088d.

```

Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-143305919, default 2048):
Last sector, +sectors or +size{K,M,G,T,P} (2048-143305919, default 143305919):

```

Created a new partition 1 of type 'Linux' and of size 68.3 GiB.

```

Command (m for help): w
The partition table has been altered.

```

Syncing disks.

Damos formato a la partición:

```
[root@hood03 rancher]# mkfs.ext4 -L RANCHER_STATE /dev/sda1
mke2fs 1.43.9 (8-Feb-2018)
Creating filesystem with 17912984 4k blocks and 4481024 inodes
Filesystem UUID: c1581567-62ba-421d-a69e-a9ff62207103
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632, 2654208,
    4096000, 7962624, 11239424
```

```
Allocating group tables: done
Writing inode tables: done
Creating journal (131072 blocks): done
Writing superblocks and filesystem accounting information: done
```

Para crear el cluster se instaló el front-end (rancher/rancher) en la maquina *rcruz01.pic.es*
Esto se puede hacer instalando la imagen de docker de rancher/rancher:

```
docker ps pull rancher/rancher
sudo docker run -d --restart=unless-stopped -p 80:80 -p 443:443 rancher/rancher
```

Creación del cluster

Para crear el cluster, accedemos mediante la interfaz grafica en *rcruz01.pic.es*
Encontramos la contraseña de acceso default en la página oficial de rancher, y la cambiamos por la que nos convenga.

Una vez en el apartado global, clicamos en “add cluster” y escogemos un cluster custom, ya que no tenemos ningún proveedor cloud. Lo configuramos a nuestra medida:

The screenshot shows the 'Cluster Options' configuration page in Rancher. The page title is 'Cluster Options' with a subtitle 'Customize Kubernetes options for the cluster'. There are two buttons at the top right: 'Edit as YAML' and 'Read from File'. The configuration is organized into several sections with radio buttons for enabling or disabling features:

- Windows Support (Experimental):** Disabled (selected).
- Kubernetes Version:** v1113-rancher1-1 (selected in a dropdown).
- Network Provider:** Canal (selected in a dropdown).
- Project Network Isolation:** Disabled (selected).
- Ngix Ingress:** Enabled (selected).
- Metrics Server Monitoring:** Enabled (selected).
- Pod Security Policy Support:** Disabled (selected).
- Docker version on nodes:** Require a supported Docker version (selected).
- Docker Root Directory:** /var/lib/docker (selected in a text input).
- Default Pod Security Policy:** None (selected).
- Node Port Range:** 30000-32767 (selected in a text input).
- Cloud Provider:** A message states 'If your cloud provider is not listed, please use the Custom option.' The 'Custom' option is selected among None, Amazon, Azure, and Custom.

Luego configuramos los permisos de los nodos, y se nos dará un comando con el cual podemos añadir nodos al cluster:

The screenshot shows the 'Node Options' configuration screen in Rancher. It has two main sections. The first section, labeled '1', is titled 'Node Options' and contains the instruction 'Choose what roles the node will have in the cluster'. Below this is a 'Node Role' section with three checkboxes: 'etcd', 'Control Plane', and 'Worker', all of which are checked. A link 'Show advanced options' is visible to the right. The second section, labeled '2', is titled 'Run this command on one or more existing machines already running a supported version of Docker.' and contains a long terminal command for running the rancher-agent. A copy icon is on the right side of the command box.

```
1 Node Options
Choose what roles the node will have in the cluster

Node Role
☒ etcd ☒ Control Plane ☒ Worker
Show advanced options

2 Run this command on one or more existing machines already running a supported version of Docker.

sudo docker run -d --privileged --restart=unless-stopped --net=host -v /etc/kubernetes:/etc/kubernetes -v
/var/run:/var/run rancher/rancher-agent:v2.1.0 --server https://rcruz01.pic.es --token
p5hqccrpth2fn4qkngkxbqz995mnngb86swk52bmtcrwdgfpkj8h --ca-checksum
e940506dc342a5694a09ef4584e49696bc3d6f6687b57966957ae23a957c8774 --etcd --controlplane --worker
```

Si ejecutamos el comando en cada uno de los nodos, estos se añadirán al cluster y los podremos ver en la interfaz gráfica. Tarda bastante tiempo en configurar el cluster, así que no hace falta alarmarse si tarda mucho.

Despliegue de aplicaciones

Una vez están todos los nodos activos vamos a la configuración global y en el apartado de “library”, activaremos todas las librerías para tener disponibles todas las aplicaciones posibles de forma que no tengamos que hacer una instalación manual a través de “kubectl”. Si alguna aplicación no se encuentra en la librería de rancher, esta se debe instalar manualmente tal y como indique tal aplicación.

Para instalar una aplicación primero seleccionamos el project donde queremos instalarlo. Podemos crear un project nuevo en “projects/namespaces”. Una vez en un proyecto iremos a “catalog apps → launch app”. Ahí buscaremos la aplicación que deseemos, en nuestro caso instalamos una plataforma ELK entera.

Antes de lanzar la aplicación, debemos establecer una configuración, en la mayoría de aplicaciones las opciones de configuración vienen definidas en “detailed description”. Una vez sabemos que configuración queremos que tenga, lo más simple es ponerla en formato de YAML clickando en “edit as yaml”. También podemos subir directamente un archivo de configuración yaml para que lo lea.

Configuration Options

Helm templates accept a comma separated list of strings

Name *

Add a Description

Template Version

elastic-stack-hqjtf


110

Select a version of the template to deploy

This application will be deployed into the **elastic-stack-hqjtf** namespace [Customize](#)

Paste or Read answers in a yml/yaml format.

Edit as a Form

Read from File 

```
1 # You can find default configuration in PREVIEW section.
2 defaultImage: true
3 elasticsearch.image.repository: docker.elastic.co/elasticsearch/elasticsearch-oss
4 elasticsearch.image.tag: 6.4.1
5 kibana.image.repository: docker.elastic.co/kibana/kibana-oss
6 kibana.image.tag: 6.4.1
7 elasticsearch.master.heapSize: 512m
8 elasticsearch.master.service.type: NodePort
9 elasticsearch.master.persistence.enabled: 'false'
10 elasticsearch.master.persistence.storageClass: ""
11 elasticsearch.master.persistence.size: 10Gi
12 elasticsearch.master.replicas: 2
13 kibana.ingress.enabled: true
14 'kibana.ingress.hosts[0]': rancher-pic.pic.es
15 logstash.service.type: NodePort
```

Una vez configurado le damos a “launch” y la aplicación comenzará a desplegarse.

Muchas de las aplicaciones necesitan persistent volumes para funcionar ya que deben guardar datos físicamente y debemos proporcionar un volumen para ello.

Para crear el volumen, vamos a “cluster → storage → persistent volumes → add volume”

Como queremos crear un volumen que resida en el propio nodo donde está instalada la aplicación, debemos primero crear una label que nos permita seleccionar estos nodos como volumen persistente. Para ello vamos a la lista de nodos, editamos cada nodo y en labels ponemos “node = my-node”.

Vamos a añadir un volumen nuevo, por lo tanto seleccionamos un nombre, un tamaño, el path y en nuestro caso indicamos que se guardará en disco de los propios nodos. Por último, en “node selector” añadimos la premisa que hemos usado en los labels antes:

The screenshot shows the Rancher UI configuration for a new storage volume. The volume is named 'test'. The volume plugin is set to 'Local Node Disk' with a capacity of '10 GiB'. The path is '/mnt'. The access modes are configured with 'Single Node Read-Write', 'Many Nodes Read-Only', and 'Many Nodes Read-Write' all checked. The storage class is set to 'None'. The node affinity rule is configured with the key 'node', operator 'in list', and value 'my-node'.

Debemos crear tantos volúmenes como sea necesario dependiendo de la aplicación y con los tamaños adecuados para cada workload.

Una vez la aplicación tenga todos sus workloads en activo, ya podremos acceder a nuestra aplicación. Aun así, algunas aplicaciones pueden necesitar configuración posterior para interconectar sus módulos, como es el caso de ELK, donde debemos modificar el YAML de kibana y logstash para indicar el nombre del servicio elasticsearch al cual tienen que apuntar para funcionar.

Acceso a las aplicaciones

Hay distintas formas de acceder a las aplicaciones de kubernetes. La más apropiada sería usar LoadBalancer, pero no se puede utilizar si no se tiene proveedor de cloud. De forma que debemos utilizar ingress para las aplicaciones que hacen de front-end como kibana, ya que nos da un formato URL que es el más apropiado. Otras aplicaciones pueden usar el formato NodePort, que nos proporciona un puerto y una IP estática para el servicio.

Para crear un ingress URL de una aplicación, primero debemos instalar un ingress controller, que será el que se encargue de gestionar los servicios dentro del cluster. En el caso de rancher tenemos ya instalado en el sistema por defecto NGINX que hace la función, pero la versión que está instalada por defecto solo nos permite tener una URL sin subdominios como <http://url/kibana>. De forma que para tener múltiples servicios en la misma URL debemos hacer una instalación manual de un ingress controller a través de kubectl. Una vez instalado el controlador, debemos crear la norma de ingress en la aplicación, esto se puede hacer en rancher en el apartado de “workloads → Load Balancing → add ingress”

o bien podemos indicar su creación en la configuración antes de lanzar la aplicación, tal y como se ha hecho en la instalación de ELK.

Para crear el aspecto por NodePort, simplemente lo especificamos antes del lanzamiento de la aplicación y se abrirá un puerto entre el 3000-3270 donde podremos acceder con una IP estática que se le asigna automáticamente. La IP corresponde a uno de los nodos, que no necesariamente contiene el servicio al que se enlaza, pero puede dar problemas si el nodo de la correspondiente IP se elimina o se cae.

Endpoint ↕	Protocol ↕
http://rancher-pic.pics/	HTTP
http://192.168.95.38:31430	TCP
	UDP

Configuración de ELK post instalación

Es necesario cambiar los YAML de las aplicaciones una vez instaladas, ya que los nombres de los servicios se generan después de la instalación. Por eso en los yaml de los workloads de kibana y logstash debemos poner el nombre de tal servicio:

```
spec:
  containers:
  - env:
    - name: ELASTICSEARCH_URL
      value: http://elastic-stack-elasticsearch-client:9200
```

```
- name: ELASTICSEARCH_HOST
  value: elastic-stack-elasticsearch-client
- name: ELASTICSEARCH_PORT
  value: "9200"
```

Una vez ya tenemos interconectada la plataforma ELK debemos configurar logstash para que envíe datos. Esto se hace a través del archivo de pipeline que encontramos en el apartado de “configmaps” de la aplicación. Lo configuramos de la misma forma que un archivo pipeline de logstash común, pero debemos tener en cuenta 2 cosas:

Tenemos que abrir los puertos de input en los servicios y en los workloads para que el container de logstash abra realmente ese puerto. Por ejemplo, queremos escuchar syslog a través del puerto 1514 UDP, para ello en los yaml de servicio y workload indicaremos que abra ese puerto en el container y que ese puerto será para UDP:

```
spec:
  clusterIP: 10.43.177.114
  externalTrafficPolicy: Cluster
  ports:|
  - name: udp
    nodePort: 31430
    port: 1514
    protocol: UDP
    targetPort: udp
```

```
ports:
- containerPort: 9600
  name: monitor
  protocol: TCP
- containerPort: 5044
  name: beats
  protocol: TCP
- containerPort: 1514
  name: udp
  protocol: UDP
```

Lo segundo que tenemos que tener en cuenta es que los datos que queremos enviar a logstash, deberán ser enviados al NodePort del servicio. En este caso enviamos el rsyslog a la dirección NodePort y esta lo redirige al UDP 1514.

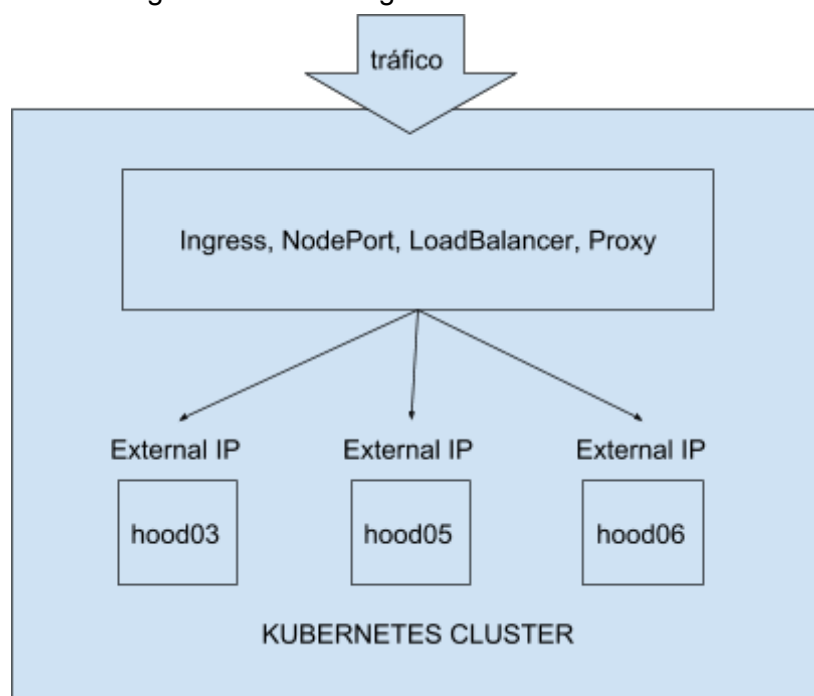
Anexo

Funcionamiento interno de kubernetes

Kubernetes tiene diferentes niveles de abstracción. Empezaremos explicando de fuera a dentro esta estructura.

Capa ExternalCluster

Esta capa es la visión es la que tenemos desde fuera del cluster. La forma que coje la estructura de esta capa es distinta dependiendo de qué tipo de acceso escogemos, pero una visión general sería la siguiente:



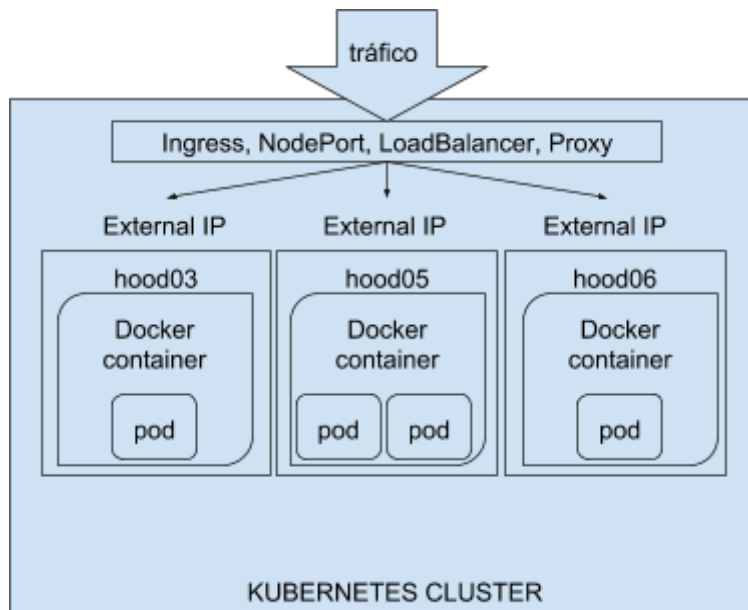
Podemos ver que el tráfico llega al cluster de kubernetes y los controladores de acceso son los que se encargan de recibir la petición y redirigirla al nodo donde se encuentra nuestra aplicación.

La comunicación entre estos controladores de acceso y los nodos, se realiza a través de una resolución "IP → servicio" que se explica más adelante.

Capa InternalCluster

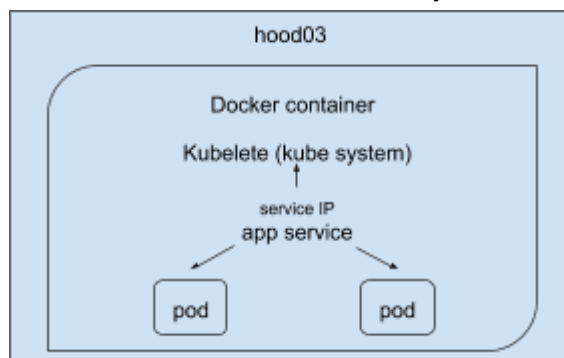
En el siguiente nivel de abstracción entramos dentro de cada nodo, en cada nodo encontramos un container, que en este caso se trata de un container de docker. El container de docker tiene instalado un entorno llamado Kubelete el qual tiene todos los daemons del sistema kubernetes que permiten el funcionamiento del sistema. Dentro del entorno se sitúan las aplicaciones dentro de lo que kubeletes llama pods, que consisten en un encapsulado asociado a una aplicación donde residen físicamente los archivos de dicha

aplicación. Estos pods son vistos por Kubelete como una unidad independiente de la aplicación que tenga dentro, y son manejados y replicados con tal de proporcionar escalabilidad y disponibilidad.

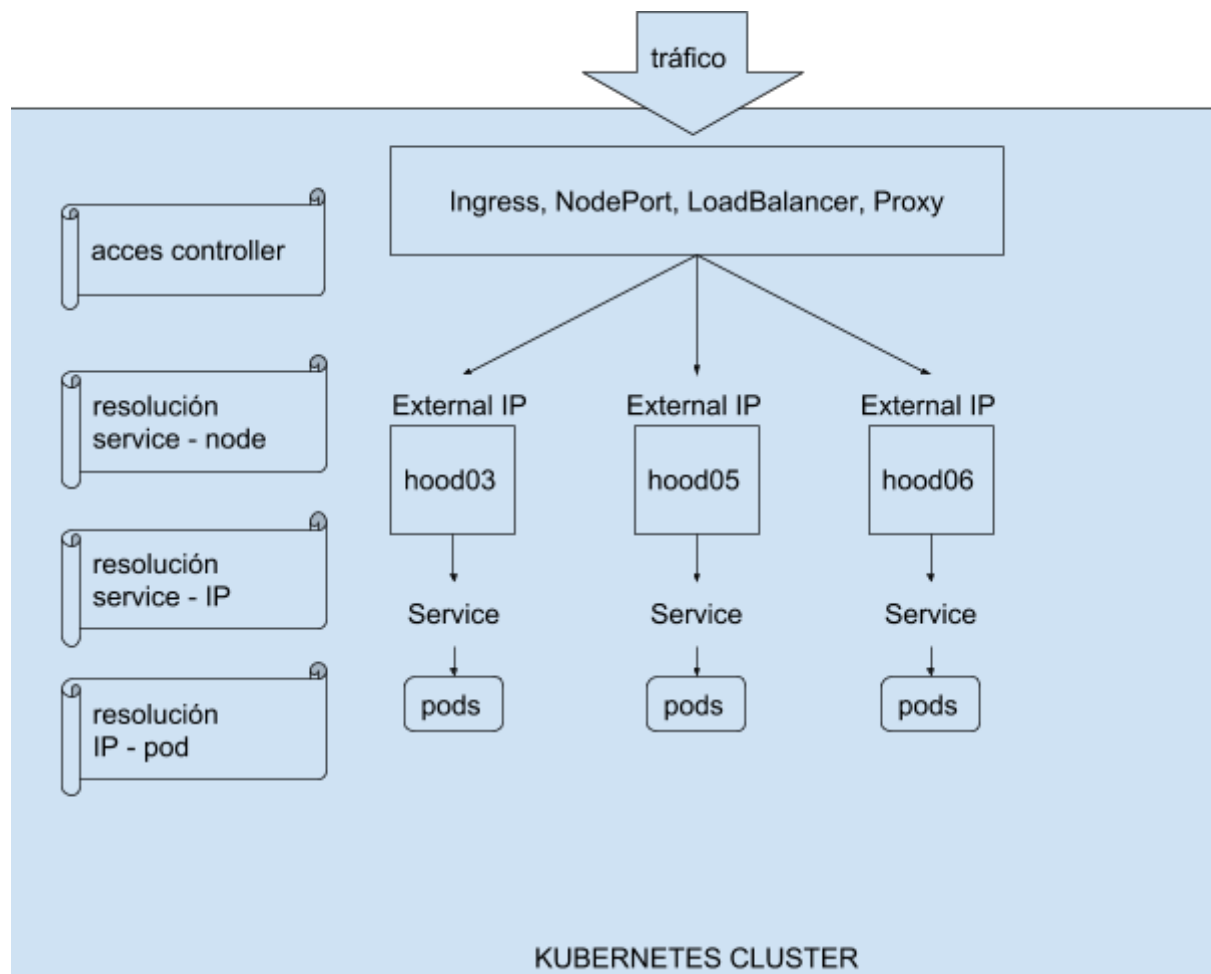


Si nos acercamos a ver como funciona por dentro la comunicación entre pods y kubelete, podemos ver que los pods son representados por un servicio, este servicio puede tener asociados varios pods que en conjunto forman una aplicación.

Los servicios tienen asociada una IP que utilizan para comunicarse con las peticiones exteriores a este nivel. Es kube-dns (uno de los daemons de kubelete) el que se encarga de resolver estos servicios a sus IP y de asociar estas IP a los pods.



La jerarquía



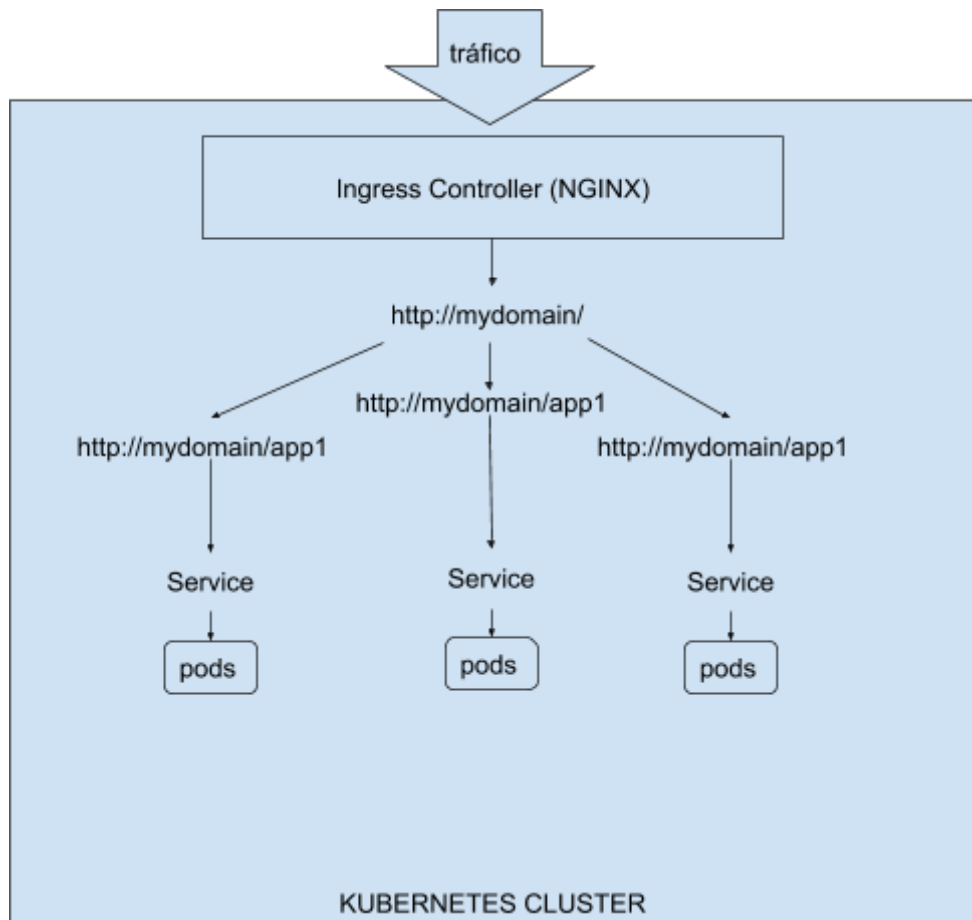
Cuando llega una petición al cluster para acceder a una aplicación, se sigue un protocolo que es el siguiente:

- La petición llega al accés controller, este envía a todos los nodos una señal de búsqueda de este aplicación. (Hay que tener en cuenta que en realidad la petición llega a todos los nodos ya que el acces controller está instalado en todos los nodos y no en un lugar externo al cluster).
- El acces controller consulta en los archivos de resolución en que nodos se encuentra este servicio, y la petición se envía al nodo donde este se encuentra.
- La petición llega al nodo y mediante kube-dns se busca la IP del servicio en la petición.
- Una vez encontrado el servicio se cargan los pods asociados y se abre la aplicación

Proceso total: aplicación → servicio → nodo → pods

Funcionamiento de los distintos accés controllers

Ingress

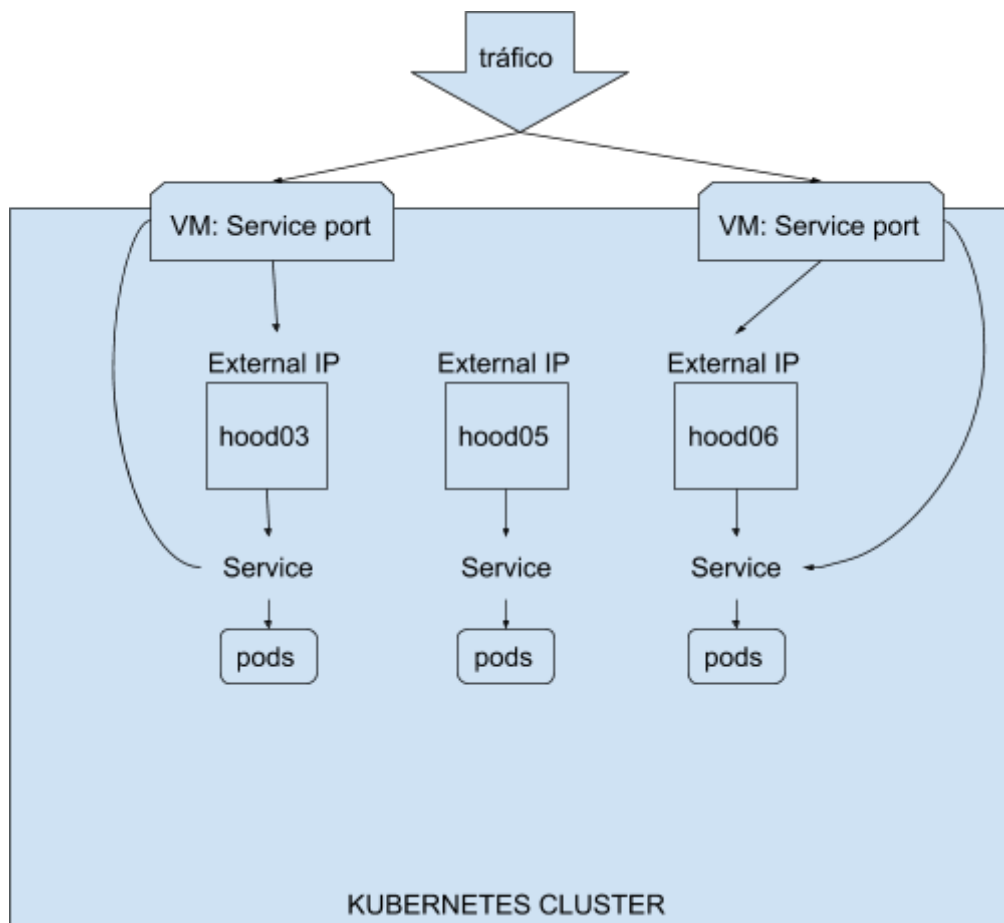


El método ingress, es el más usado cuando no se tiene un LoadBalancer, este método requiere un ingress controller, que comúnmente suele ser "NGINX-ingress-Controller". Ingress nos permite establecer el acceso a nuestros servicios mediante una URL y si se configura adecuadamente podemos crear un subdominio para cada aplicación.

Para ello debemos instalar en cada nodo el NGINX y crear un "ingress rule" para la aplicación que queramos utilizar. Este "ingress rule" es lo que indica la conexión entre servicio y puerto con la URL que especificamos.

NOTA: hace falta crear una resolución de DNS para la URL en las máquinas que usamos para acceder el servicio. Indicando así la IP externa de los nodos que dan servicio en esa URL.

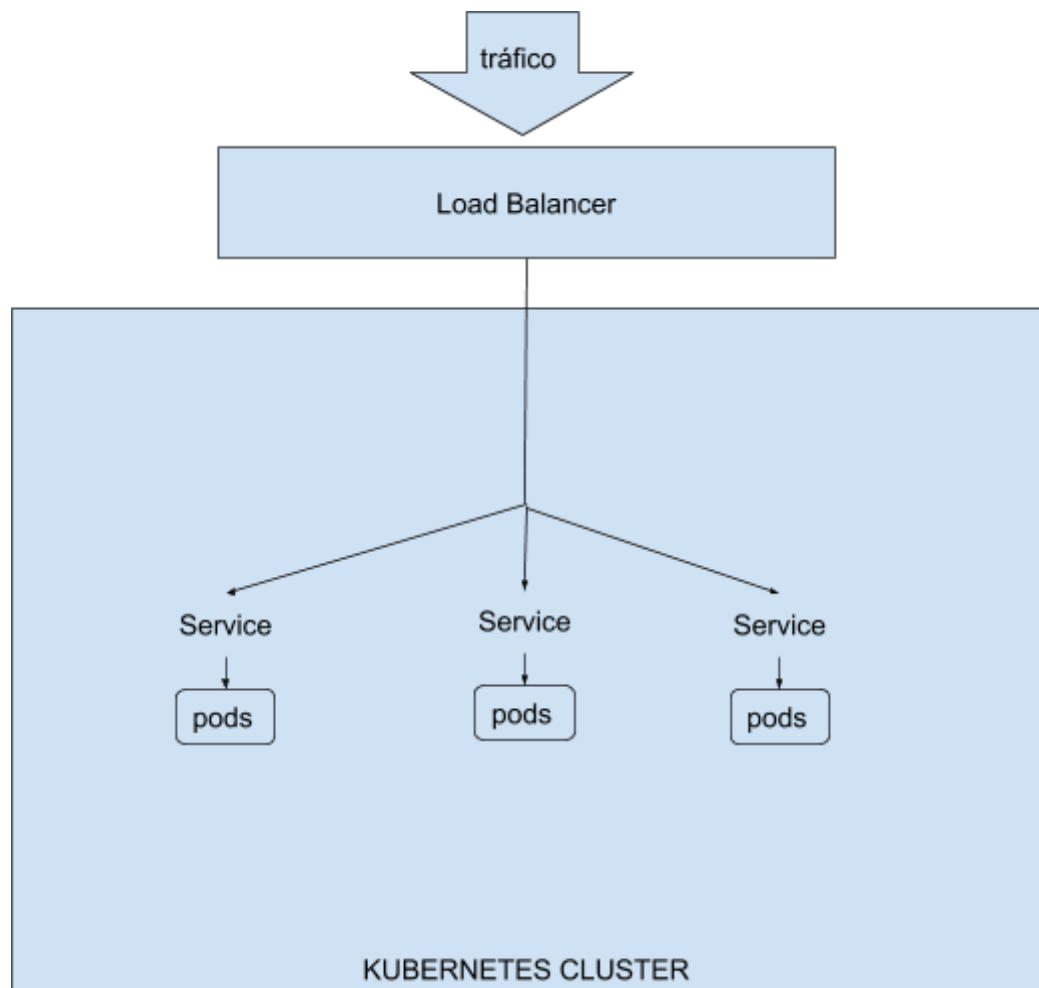
NodePort



NodePort establece un puerto del 3000-3275 que se abre de forma virtualizada en la capa exterior del cluster. Este puerto está directamente enlazado a un servicio, sin embargo la IP asociada está ligada a un nodo cualquiera, que no necesariamente contiene el servicio. Cuando la petición de servicio llega al nodo indicado en la IP este identifica la relación puerto-servicio y envía la petición de servicio al nodo que tiene la aplicación.

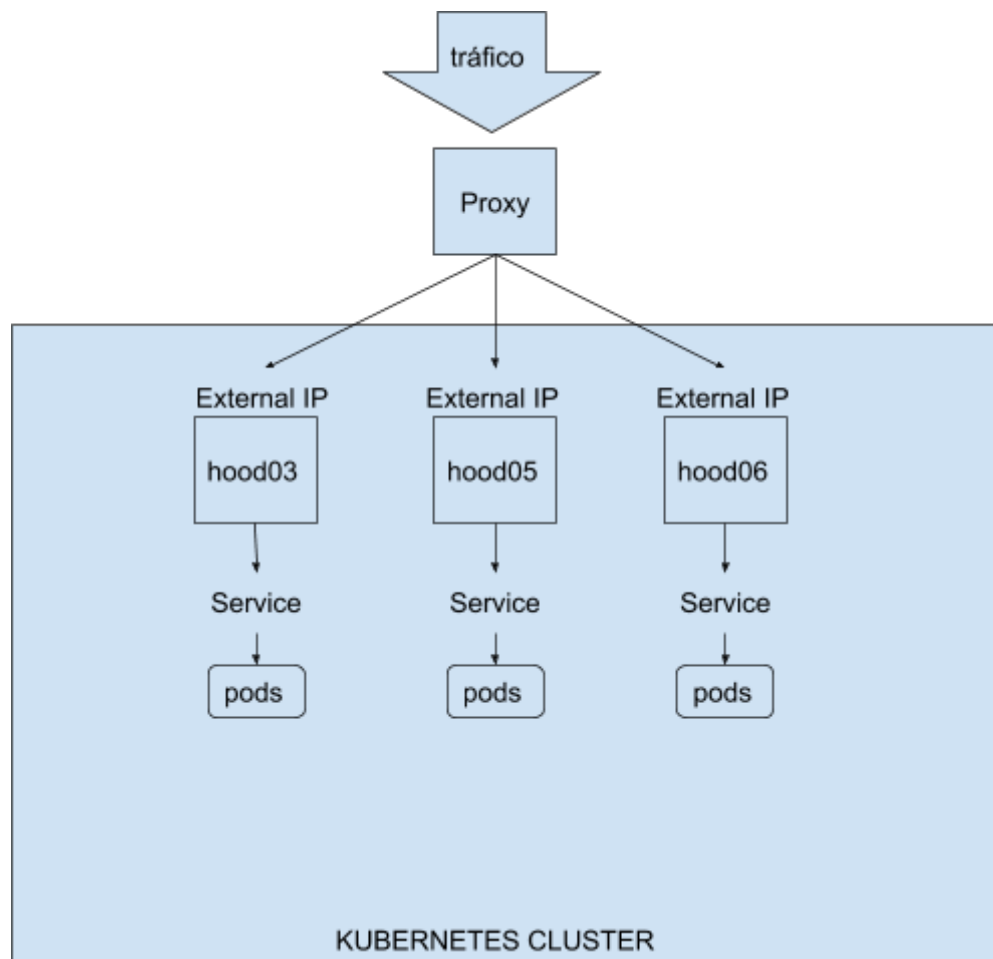
NOTA: la IP estática puede romperse si el nodo asociado a la IP está fuera de servicio, ya que las peticiones le llegaran pero no las redirigirá. Esto también implica que el tiempo de respuesta está limitado por el nodo de la IP estática.

LoadBalancer



LoadBalancer es el más fácil de utilizar, simplemente dejamos que nuestro cloud provider haga todo el trabajo. Se nos proveerá una IP externa que representa todo el cluster y sus servicios.

Proxy ClusterIP



Este tipo de acceso no permite su uso de forma externa, solo sirve para hacer tests dentro del propio cluster. Sin embargo podemos acceder los servicio utilizando kubernetes proxy desde la propia máquina donde tenemos instalado kubernetes.

Error de pantalla roja (bad operation code)

Durante la instalación del cluster había un nodo que se encontraba en un estado de “bad operation code”. Esto sucedía cuando se intentaba cargar la imagen de disco, y tampoco permitía cargar una nueva imagen de disco mediante DHCP.

Más tarde nos dimos cuenta que tras reiniciar las máquinas, por algún motivo se corrompía la imagen de rancher en el disco y los residuos que quedaban no permitían la instalación de otra imagen de disco.

Para solucionar esto se intentó borrar el disco mediante bios, pero al intentar reinstalar rancher no cogía la imagen de disco.

La solución final resultó en borrar el disco y crearlo mediante bios, pero luego instalar una imagen de CentOS que formateaba la estructura del disco que había corrompido rancher. Finalmente se instalaba rancher pisando la imagen de CentOS.