



ADMINISTRACIÓN DE INFRAESTRUCTURAS Y SISTEMAS INFORMÁTICOS (AISI)

Grado en Ingeniería Informática

Grado en Ingeniería Informática

Roberto R. Expósito (roberto.rey.exposito@udc.es)

Jorge Veiga (jorge.veiga@udc.es)

PRÁCTICA 4

Kubernetes (K8s)



Contenidos

- Objetivo
- Arquitectura de un clúster K8s
- Objetos básicos de K8s
- Ejercicios propuestos



Contenidos

- **Objetivo**
- Arquitectura de un clúster K8s
- Objetos básicos de K8s
- Ejercicios propuestos



Objetivo

5

- El propósito de esta práctica es conocer la arquitectura y el funcionamiento básico de **Kubernetes (K8s)** desplegado en un clúster virtual basado en Ubuntu utilizando Vagrant y Ansible
- K8s es una plataforma portable y de código abierto para automatizar el despliegue, escalado y administración de **aplicaciones basadas en contenedores**, facilitando su **automatización** y configuración **declarativa**
 - Herramienta IaC para la orquestación de contenedores que soporta diferentes entornos de ejecución de contenedores como por ejemplo Docker
 - Desarrollado inicialmente por Google ([Borg](#)), liberó el proyecto en 2014 y fue donado posteriormente a la [Cloud Native Computing Foundation](#) (CNCF)
- Actualmente, K8s es el orquestador de contenedores más popular y utilizado



kubernetes

<https://kubernetes.io>



¿Por qué usar un orquestador?

6

- La tendencia actual implica desarrollar y desplegar múltiples servicios desacoplados (arquitectura basada en **microservicios**) y ejecutarlos en contenedores *software*
- Este incremento en el número de contenedores pone de manifiesto la necesidad de usar herramientas apropiadas para gestionarlos de forma centralizada, automatizada y declarativa
 - Además, es altamente deseable poder desplegar los contenedores en cualquier entorno: servidor físico/virtual, clúster, cloud...
- Principalmente, las herramientas para la orquestación de contenedores pretenden proporcionar, entre otras cosas:
 - Alta disponibilidad (no *downtime*)
 - Auto escalado horizontal y vertical (alto rendimiento y mejor uso de los recursos)
 - Recuperación ante desastres
 - Integración con herramientas CI/CD



Justificación de la práctica

7

- La realización de esta práctica se justificará de la siguiente forma:
 - Documento en formato PDF que incluya las capturas de pantalla indicadas para demostrar la realización de la **parte principal de cada ejercicio**

- **Debes incluir capturas similares a las mostradas en las transparencias:**



- **38, 40 (EJ1); 44, 47, 51 (EJ2); 53, 54, 59, 61 (EJ3)**
- **66, 68 (EJ4); 73, 74, 78 (EJ5)**

Busca este icono en la parte superior derecha



IMPORTANTE



- **ENTREGA** a través de Moodle: **13/04 (15:30)**
- **ES OBLIGATORIO** usar la nomenclatura que se propone para nombrar los recursos y debe apreciarse sin confusión en las capturas aportadas
 - **NO RECORTES** las capturas de pantalla, **debe verse toda la información** que sea relevante para comprobar el trabajo realizado
- **NO** seguir estas normas **IMPLICA UNA CALIFICACIÓN “C”** en esta práctica



Contenidos

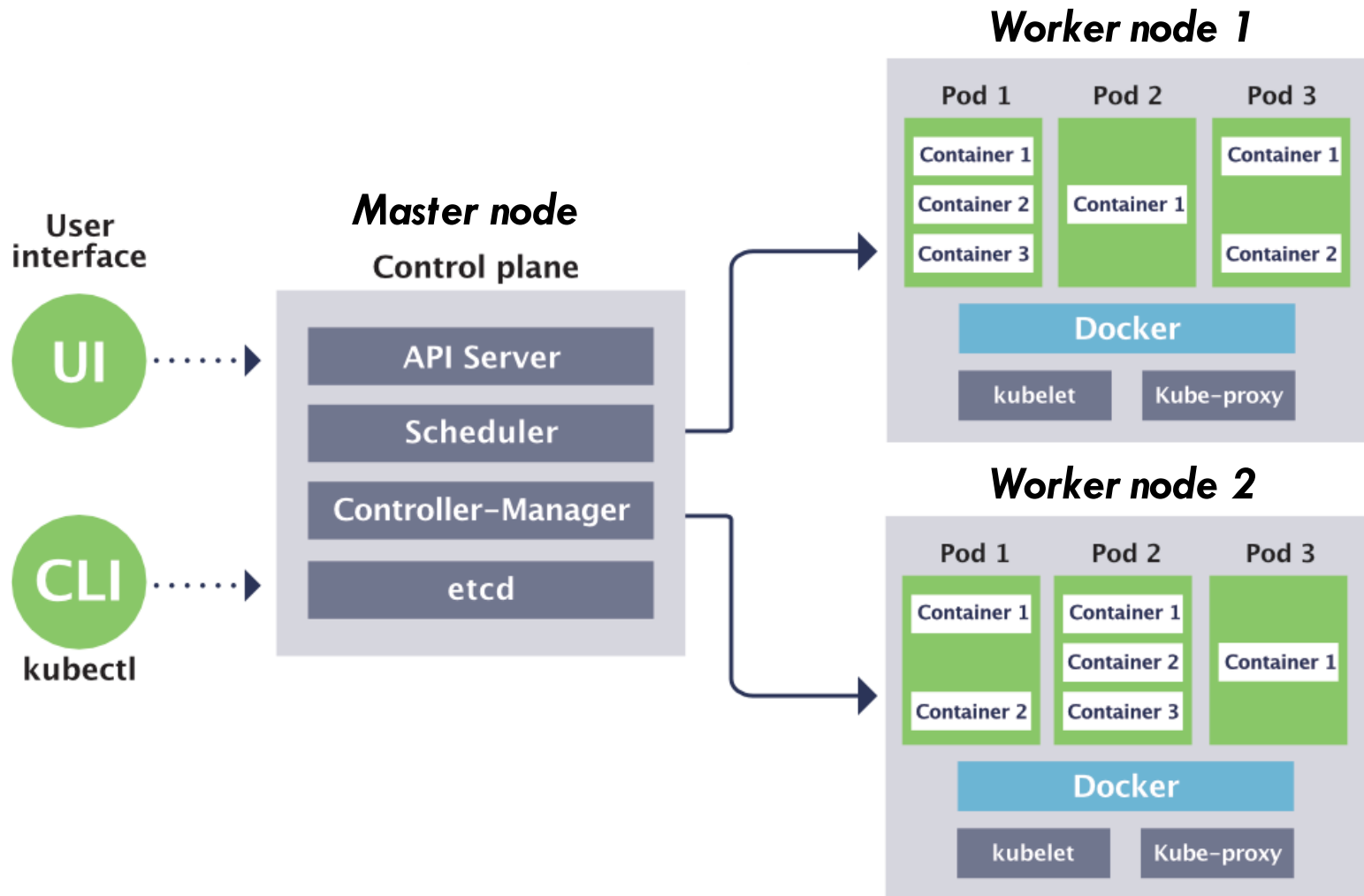
- Objetivo
- **Arquitectura de un clúster K8s**
- Objetos básicos de K8s
- Ejercicios propuestos



Arquitectura

9

- **Master/Worker**



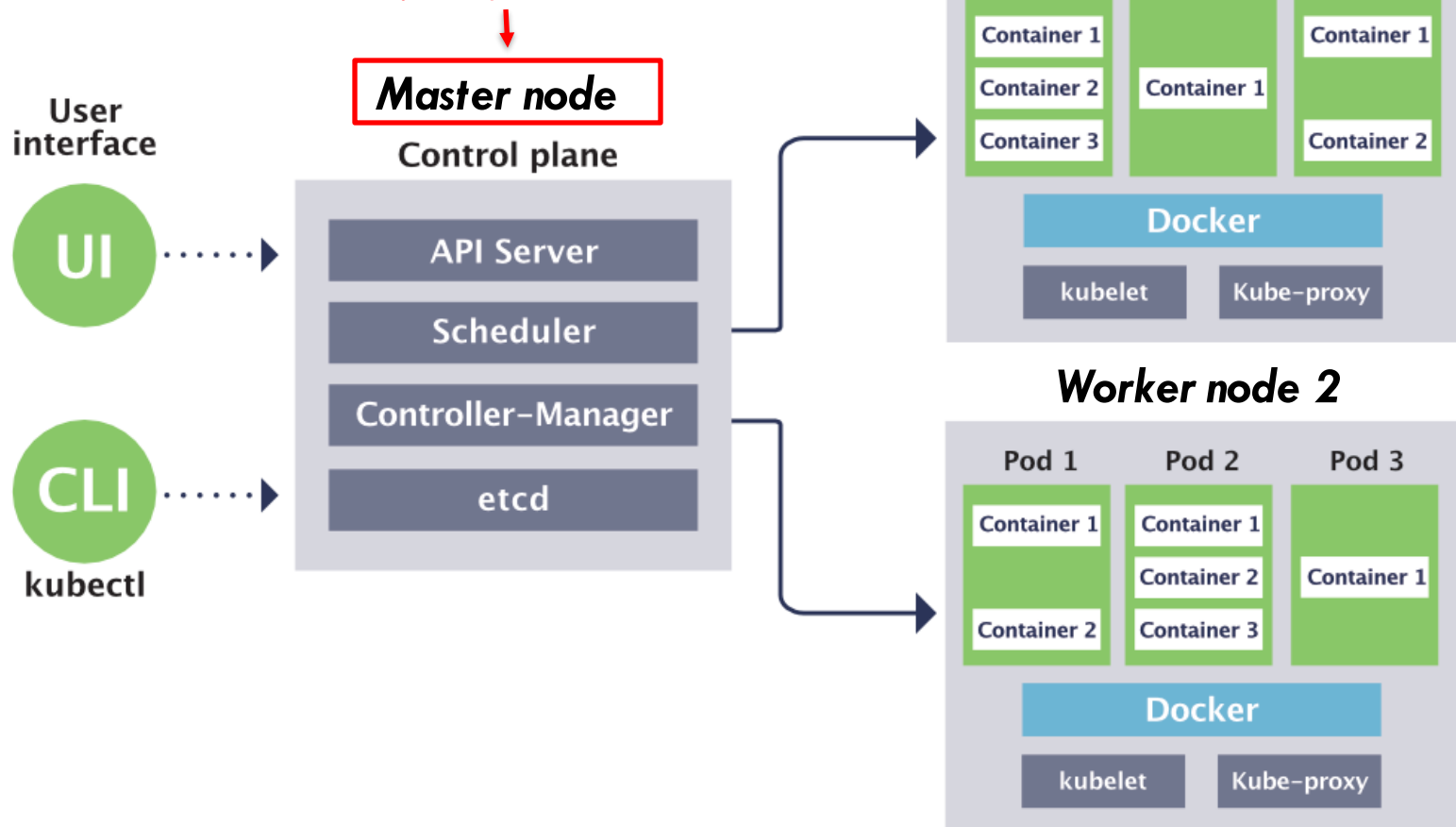


Arquitectura

10

● **Master/Worker**

El nodo *Master* ejecuta los procesos necesarios para gestionar y monitorizar el clúster K8s y sus aplicaciones

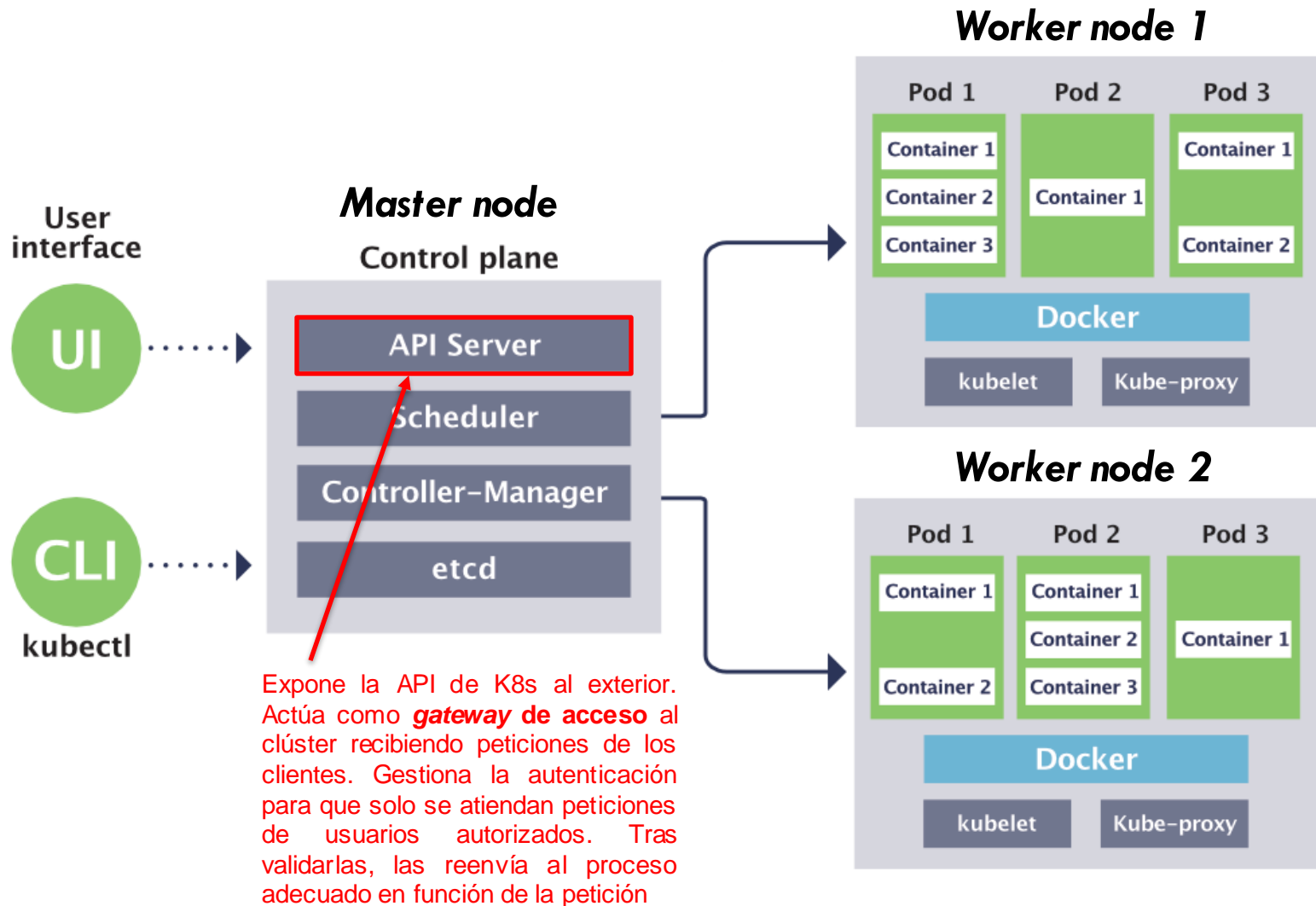




Arquitectura

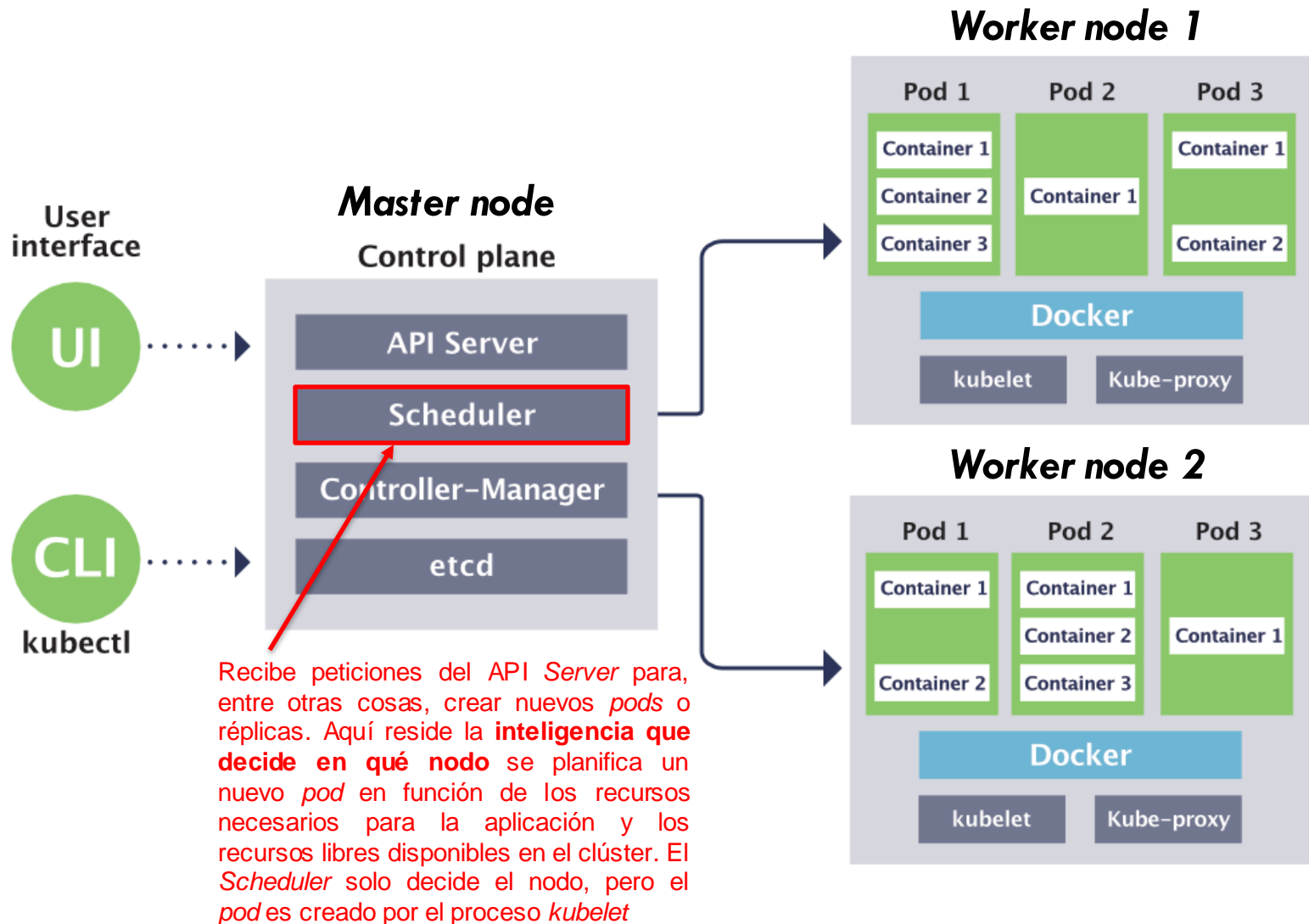
11

- **Master/Worker**



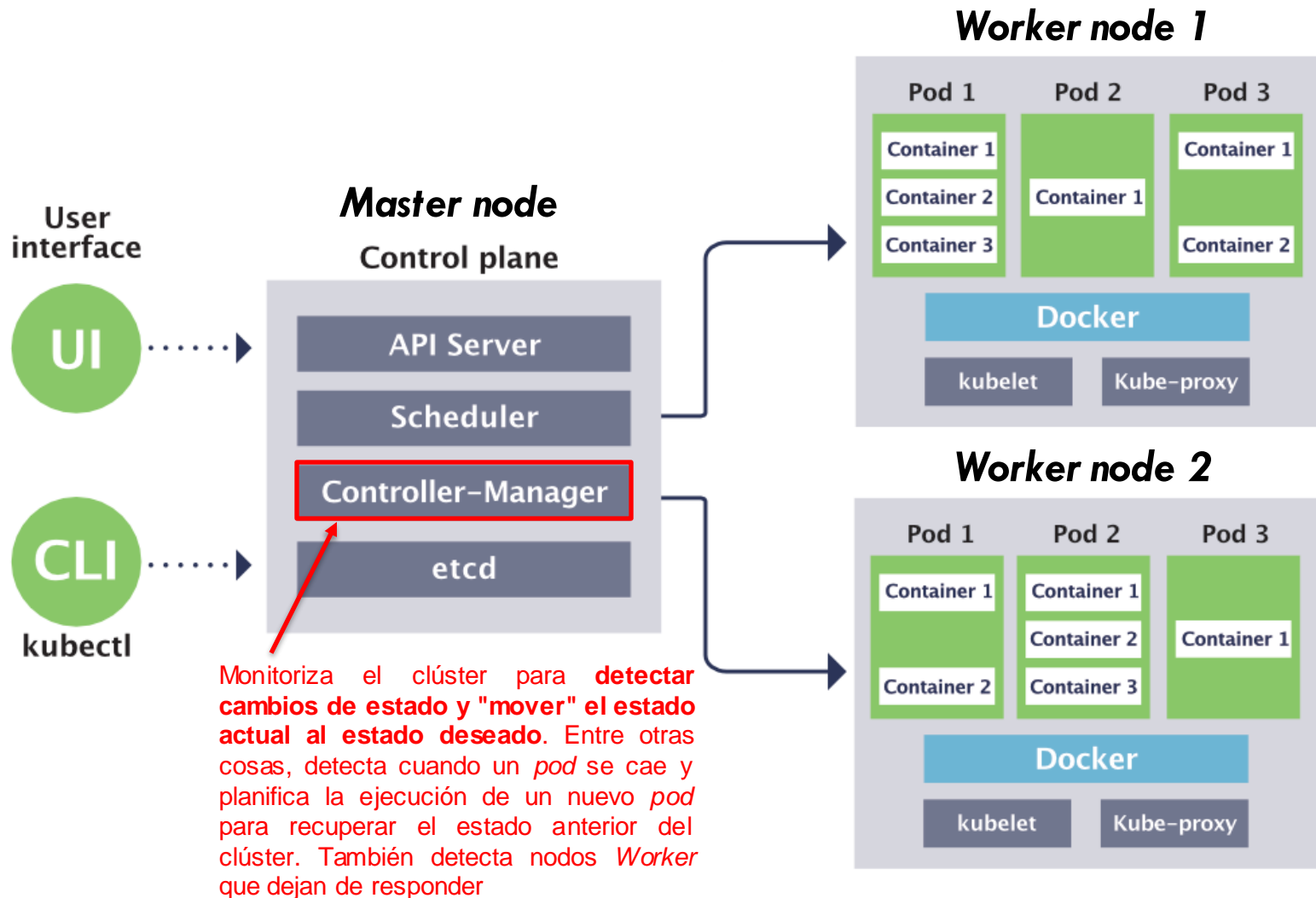


- **Master/Worker**



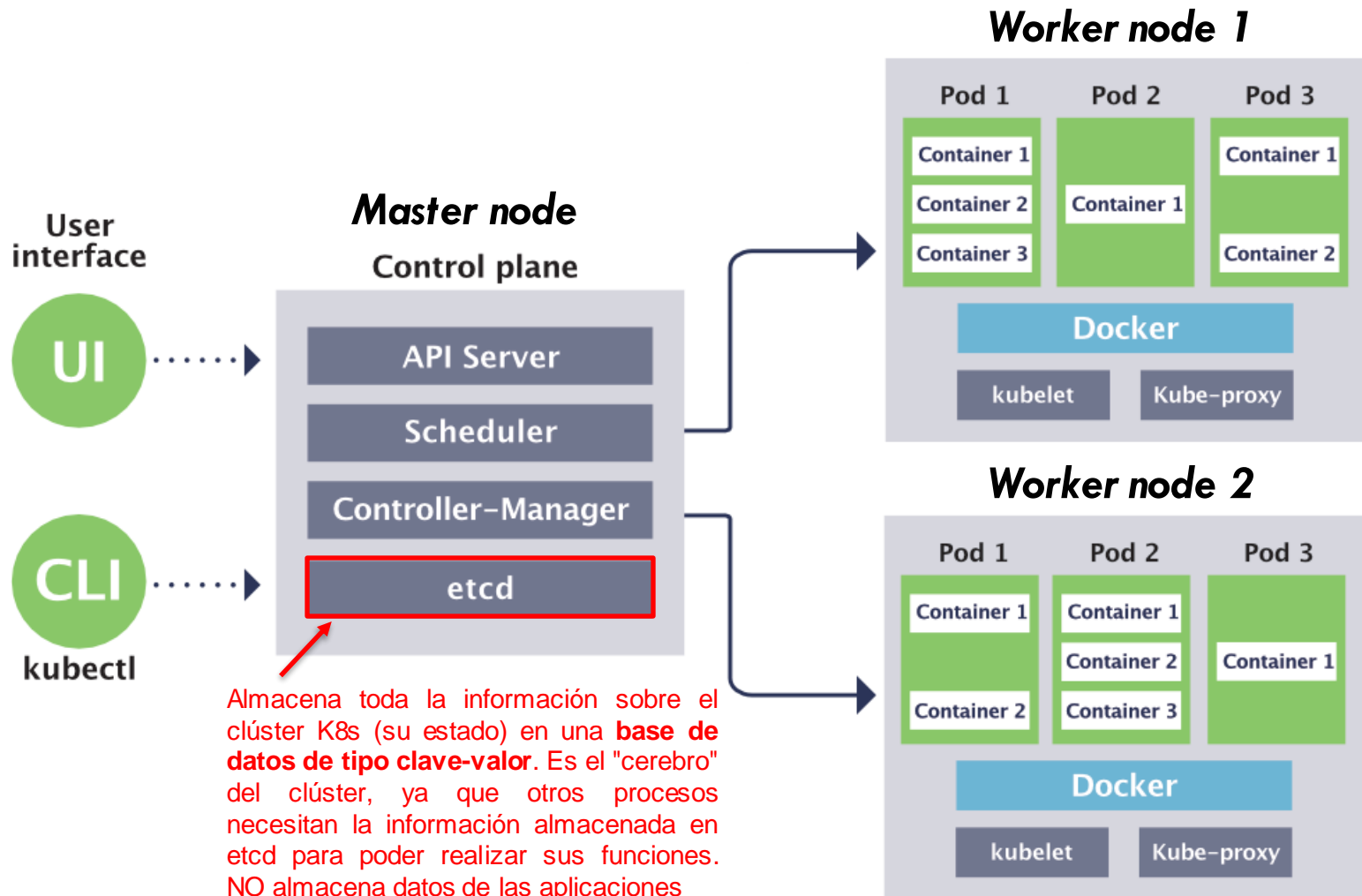


- **Master/Worker**



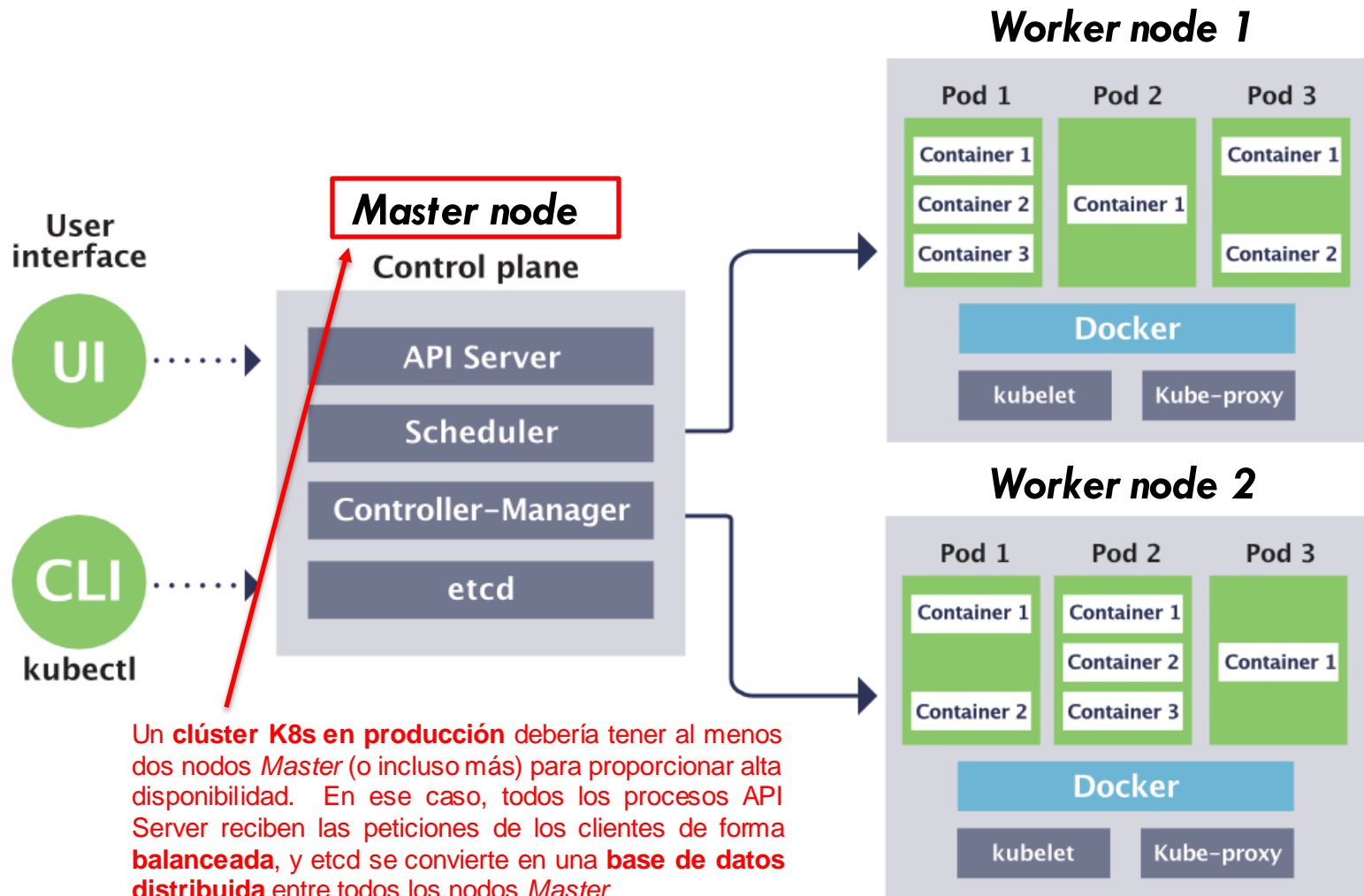


- **Master/Worker**





- **Master/Worker**



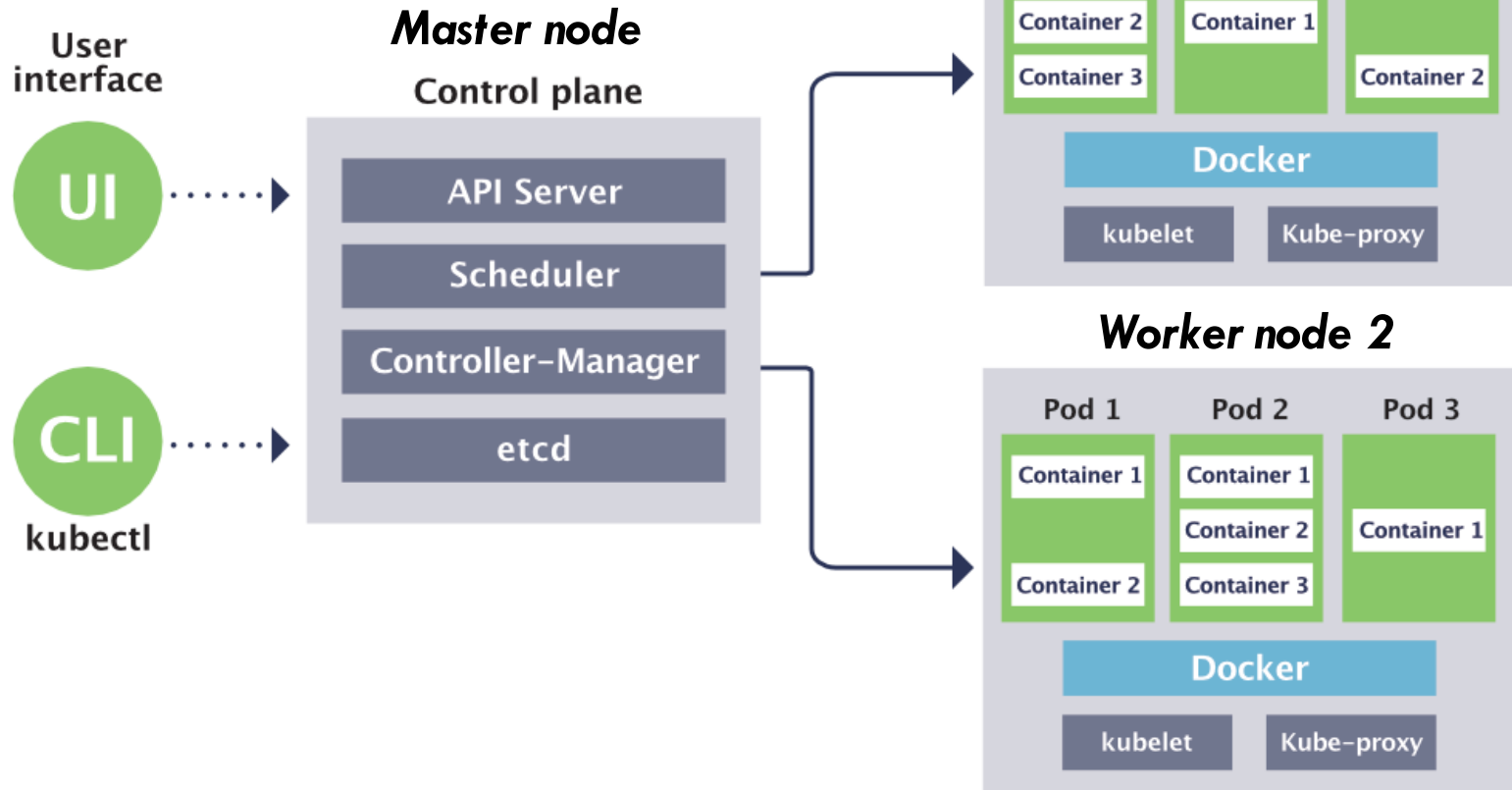


Arquitectura

16

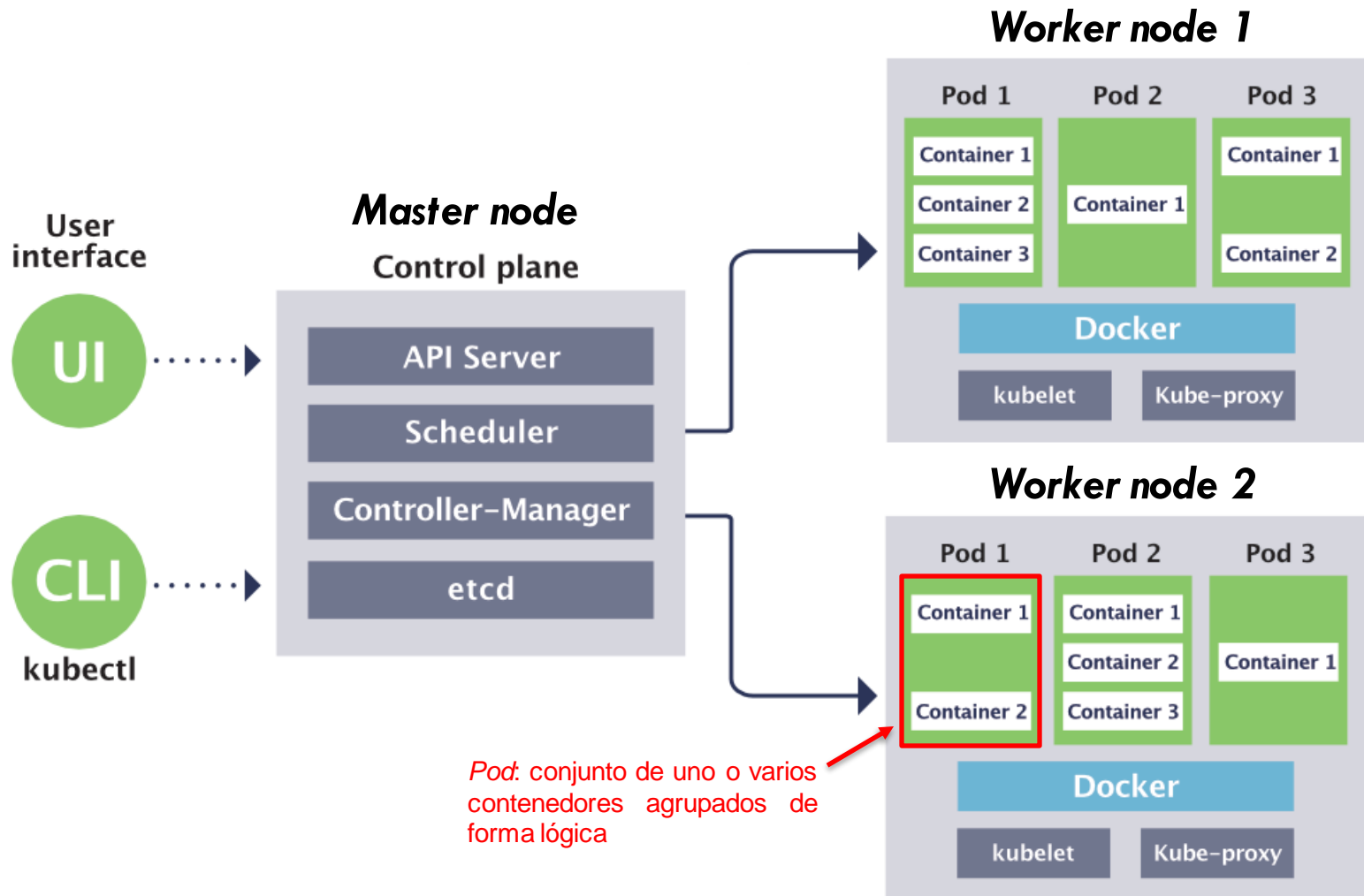
● **Master/Worker**

Los nodos *Worker* del clúster K8s proporcionan los recursos para ejecutar las aplicaciones en los *pods*



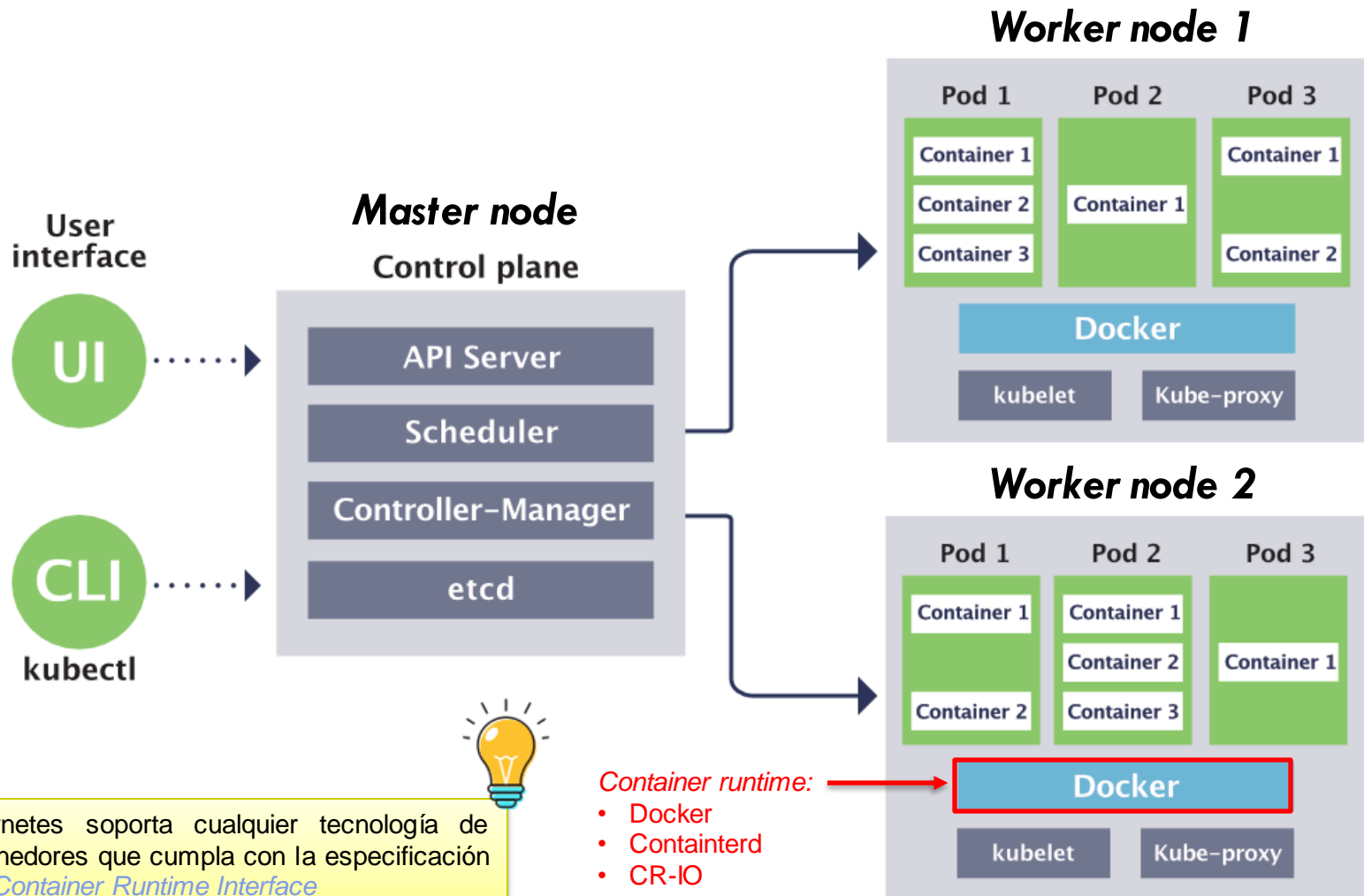


- **Master/Worker**





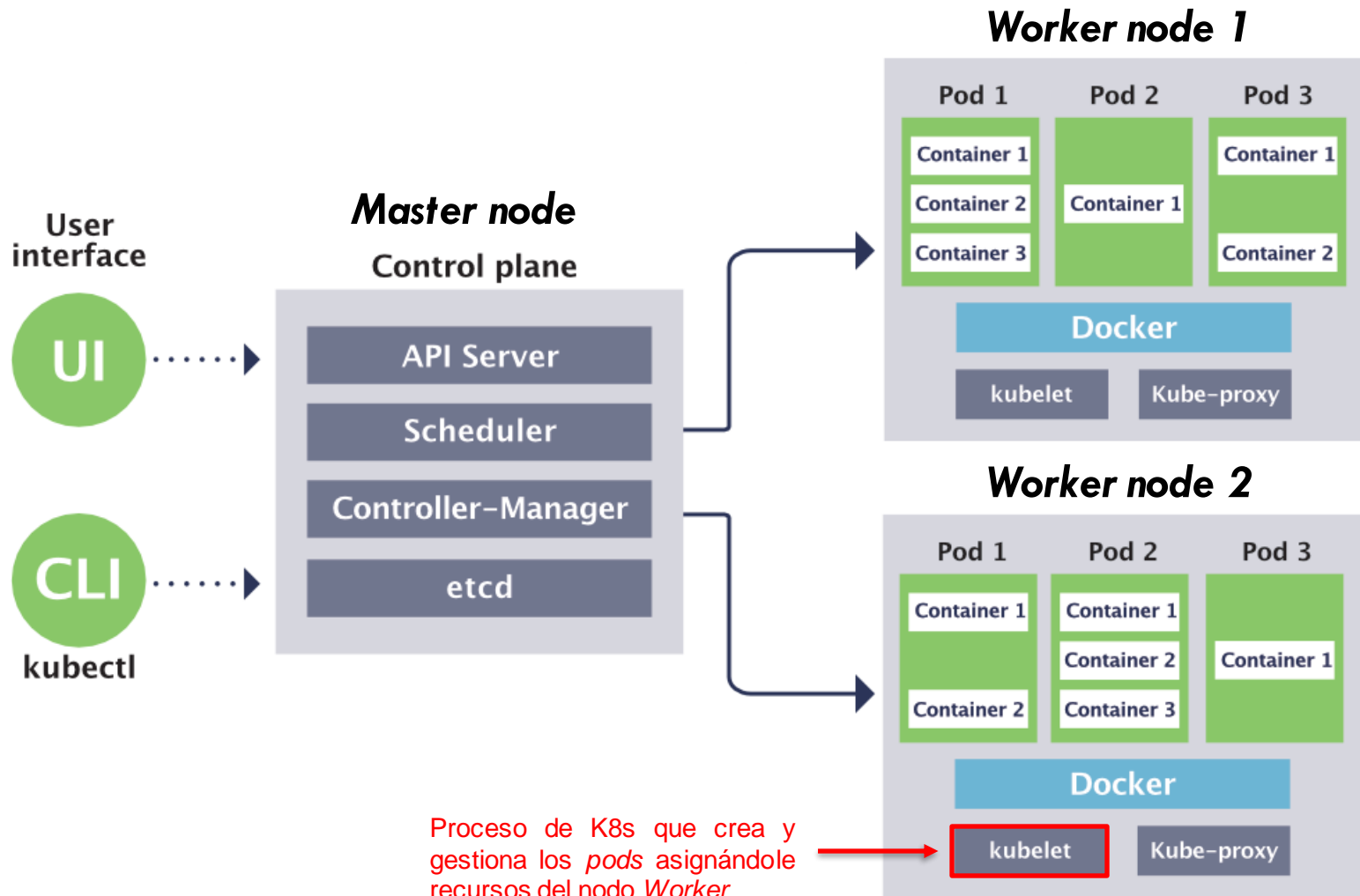
● *Master/Worker*



Kubernetes soporta cualquier tecnología de contenedores que cumpla con la especificación CRI: [Container Runtime Interface](#)

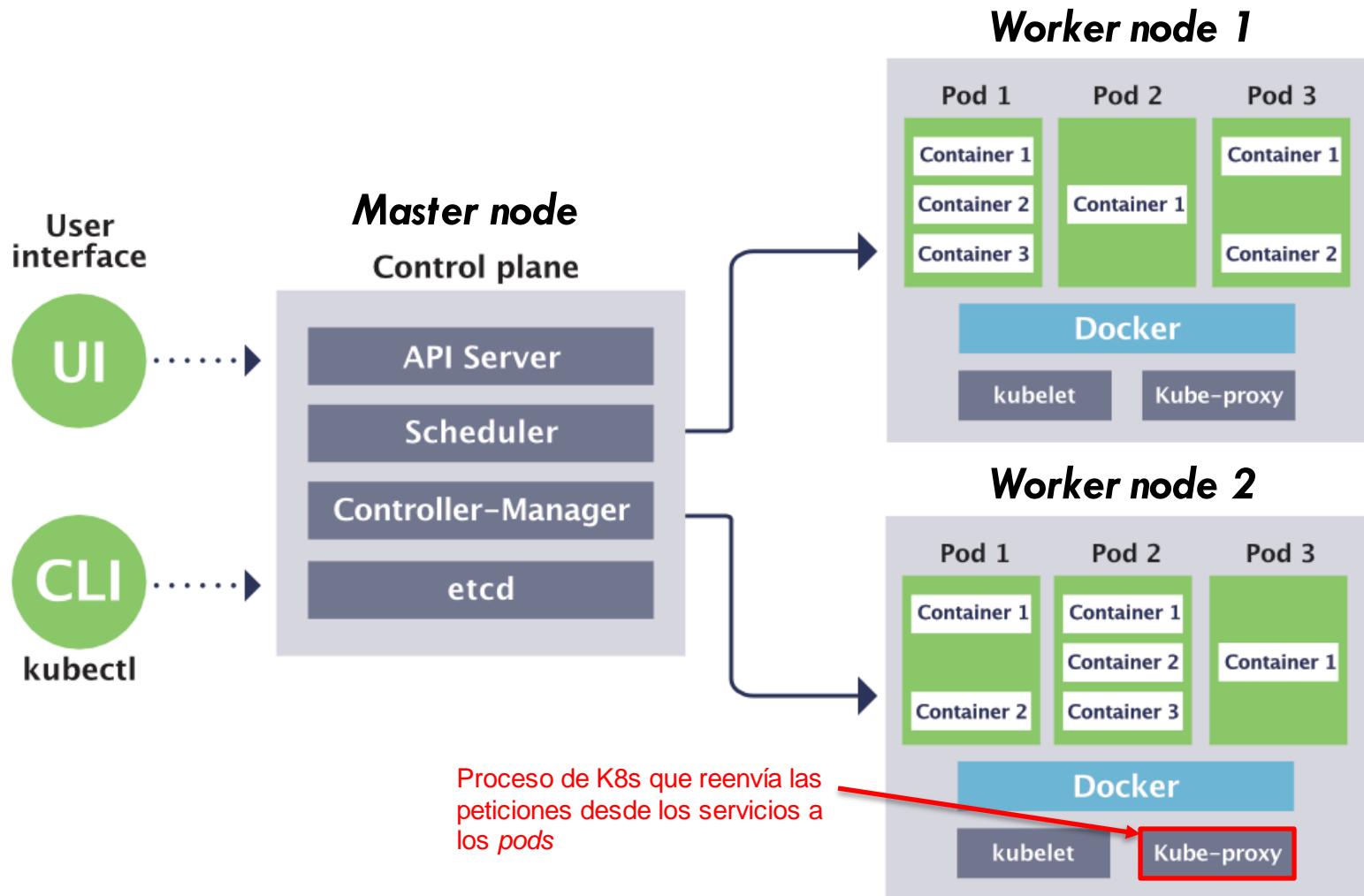


- **Master/Worker**



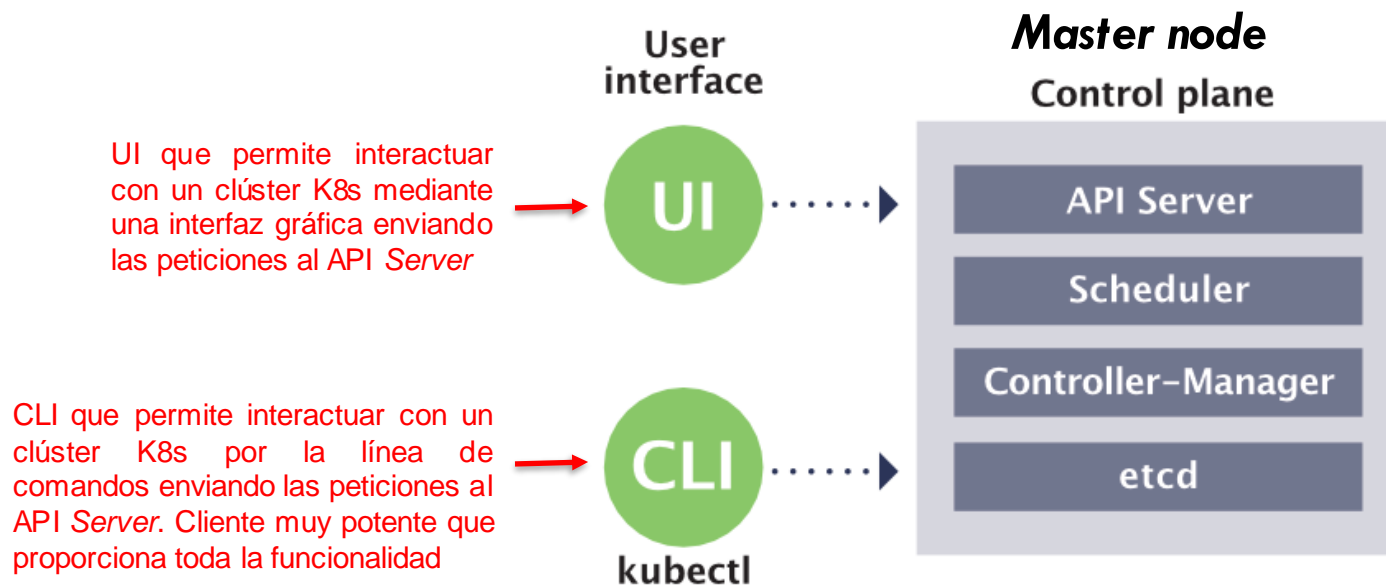


- **Master/Worker**





● **Master/Worker**



API Server expone una API RESTful por lo que también es posible interactuar con un clúster K8s de forma programática. Existen clientes disponibles para múltiples lenguajes de programación como Python, Go, Java, C, JavaScript.

<https://kubernetes.io/docs/reference/using-api/client-libraries/>





Contenidos

- Objetivo
- Arquitectura de un clúster K8s
- **Objetos básicos de K8s**
- Ejercicios propuestos



Objetos de K8s

23

- **Node (*Worker*)**

- Servidor físico o virtual que forma parte del clúster K8s y ejecuta los pods

- **Pod**

- Conjunto de uno o más contenedores agrupados que **comparten red y almacenamiento**, además de unas especificaciones de cómo se ejecutan
 - Un pod representa la **unidad mínima que K8s puede desplegar y gestionar**
 - Abstracción sobre un contenedor (independencia del entorno de ejecución)
- Cada pod tiene una dirección IP asignada dinámicamente en su creación
- Entidad efímera: si falla, simplemente se recrea pudiendo cambiar su IP

- **Volume**

- Proporciona almacenamiento efímero o persistente a los pods
- Puede ser almacenamiento local al nodo o remoto (externo al clúster)
- K8s no gestiona la persistencia, es responsabilidad del administrador

- **ReplicaSet**

- Controlador cuyo objetivo es mantener un conjunto estable de **réplicas** de un determinado pod en estado de ejecución en todo momento



● **Service**

- Forma abstracta de exponer una aplicación que se ejecuta en un *pod* como un servicio de red con una **dirección IP estática**
 - El ciclo de vida de un servicio es independiente del ciclo de vida de un *pod*
 - Si un *pod* falla y se vuelve a crear, la IP del servicio no cambia
- Existen diferentes tipos de servicios (*NodePort*, *ClusterIP*...) con diferentes características y formas de exponer la aplicación al exterior
- Proporciona **balanceo de carga** entre las réplicas de un *pod*
 - Excepto los servicios de tipo *Headless*

● **Deployment**

- Proporciona una abstracción sobre los *pods* y los *ReplicaSets*
 - En vez de trabajar con *pods/ReplicaSets* directamente, K8s recomienda su gestión y configuración mediante *Deployments*
- Permite, entre otras cosas, desplegar nuevas versiones de un *pod* sin interrupción de servicio, especificar las réplicas que se quieren crear, etc
- Válido para replicar *pods* que ejecutan aplicaciones **sin estado**
 - Para replicar un *pod* que ejecuta una base de datos es necesario usar *StatefulSet*



● **StatefulSet**

- Permite replicar *Pods* que ejecutan aplicaciones **con estado** (bases de datos)
 - Los *Pods* mantienen identidad, *hostname* y almacenamiento
 - Para evitar problemas de inconsistencia, se sincronizan los accesos de escritura al estado (datos) desde las diferentes réplicas de un *pod*
 - Su configuración no es sencilla, con lo que en muchas ocasiones las bases de datos se ejecutan de forma externa al clúster K8s

● **ConfigMap**

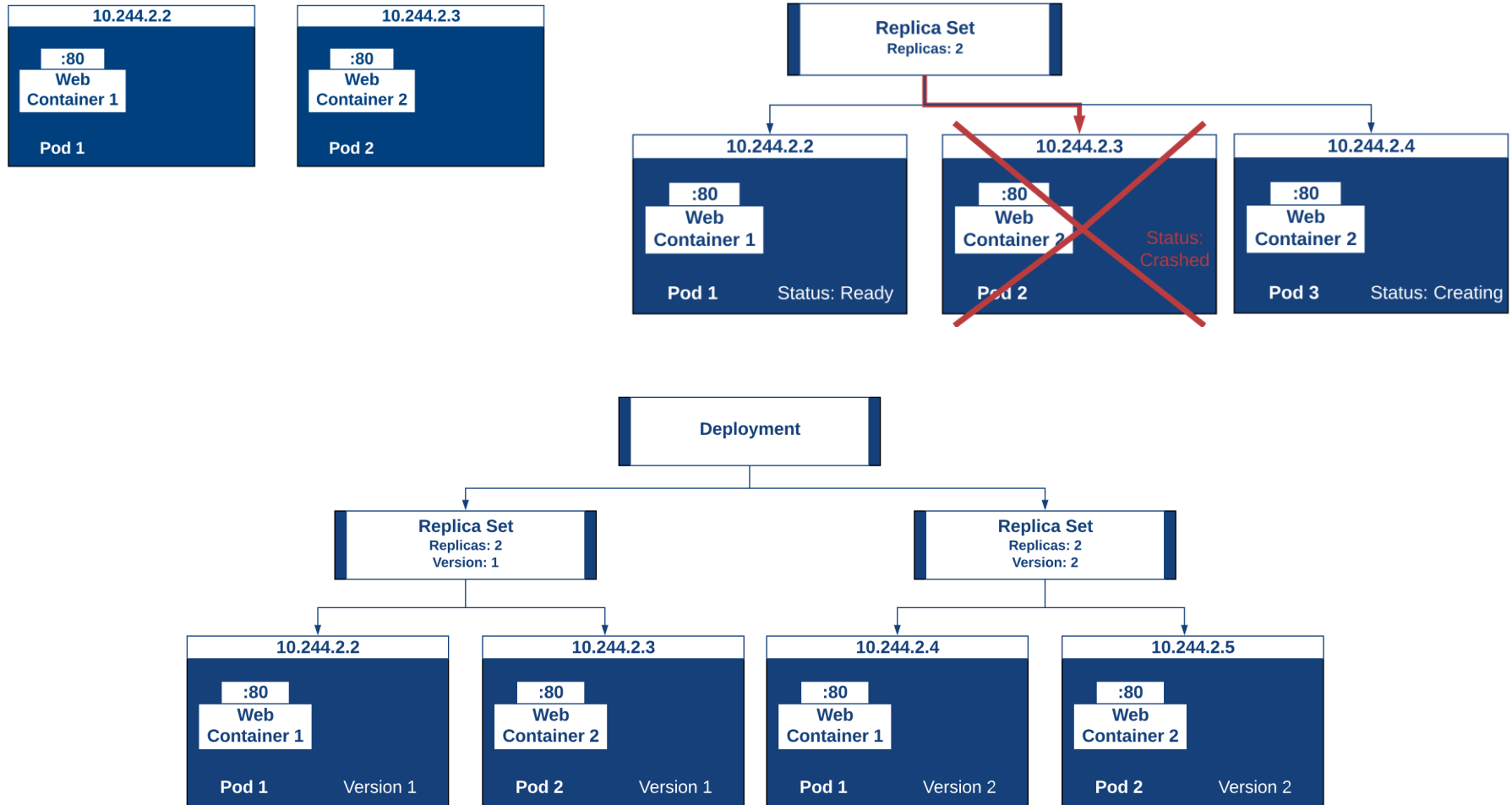
- Permite almacenar la configuración de las aplicaciones de forma externa
- Un *pod* puede usar un *ConfigMap* para obtener la configuración necesaria
 - Por ejemplo, la URL de conexión a una base de datos
- La información en un *ConfigMap* se almacena en texto plano
- No es válido para almacenar información confidencial

● **Secret**

- Similar a un *ConfigMap* pero la información se codifica usando base64
- Por tanto, es válido para almacenar información confidencial
 - Por ejemplo, las credenciales de acceso a una base de datos



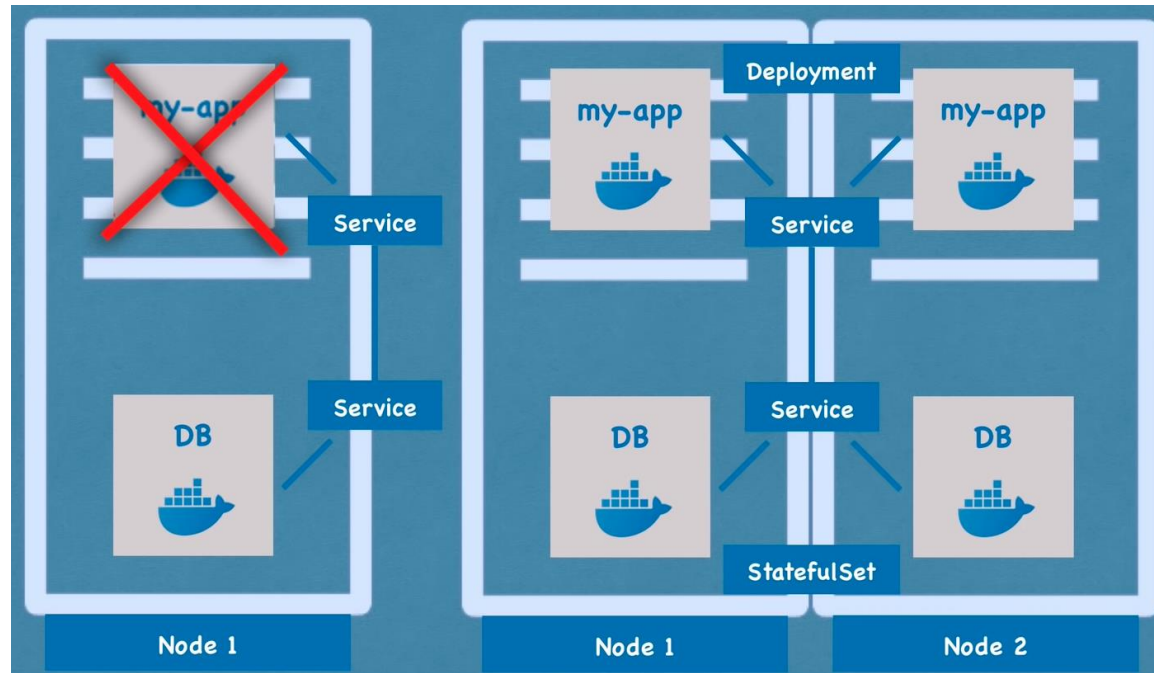
● Pod, ReplicaSet y Deployment





Objetos de K8s

- **Deployment y StatefulSet**





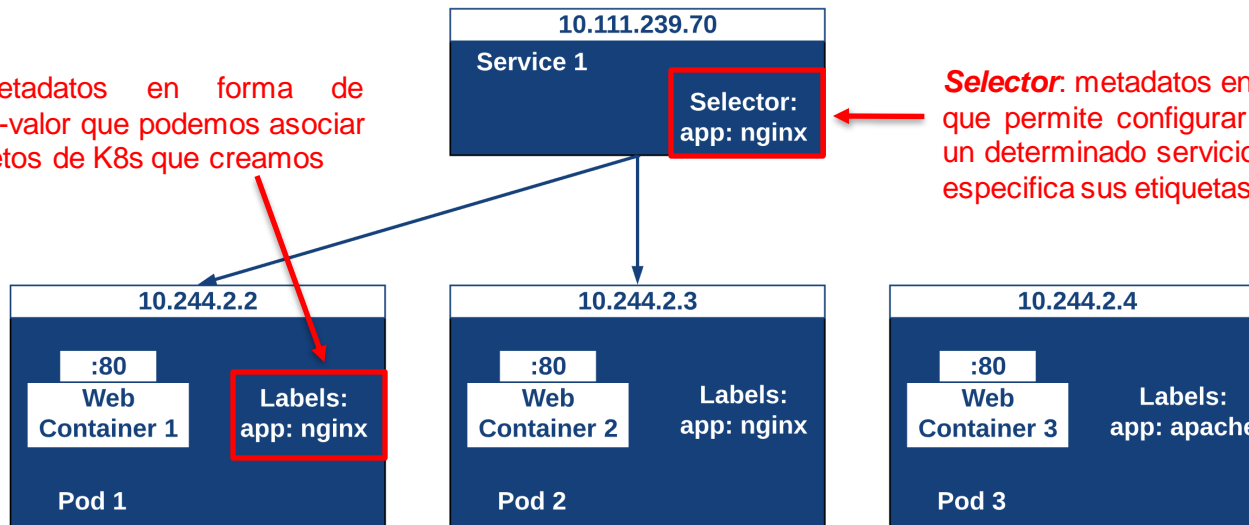
Objetos de K8s

28

Service

- **ClusterIP**: expone el servicio en una dirección IP interna del clúster (tipo **por defecto**)
- **NodePort**: expone el servicio externamente en cada nodo del clúster a través de su dirección IP (*NodeIP*) y un puerto estático (*NodePort*) en el rango 30000-32767
 - Servicio accesible desde fuera del clúster a través de `<NodeIP>:<NodePort>`
- **LoadBalancer**: expone el servicio externamente usando un balanceador de carga L4 externo proporcionado por un proveedor en la nube
 - El clúster K8s debe ejecutarse en un **entorno cloud compatible** y estar configurado con el paquete correcto del proveedor
- **Headless**: servicio sin IP interna ni balanceador (devuelve las IPs de los pods)

Label: metadatos en forma de pares clave-valor que podemos asociar con los objetos de K8s que creamos



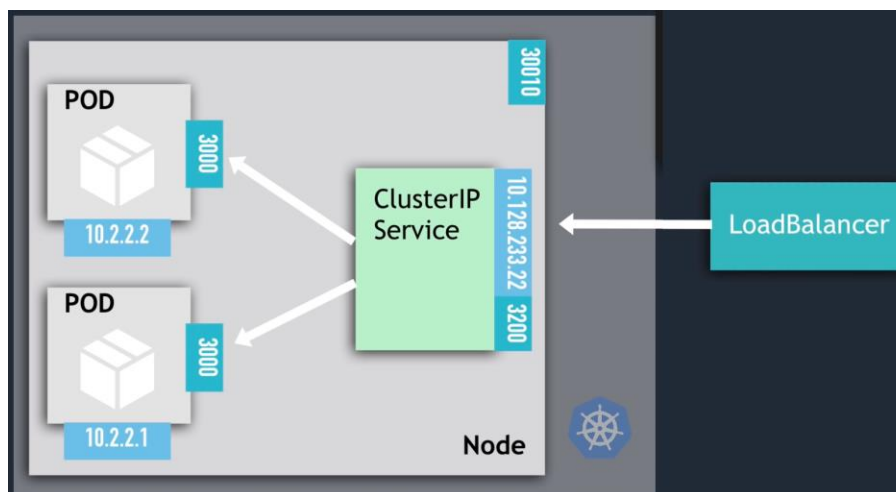
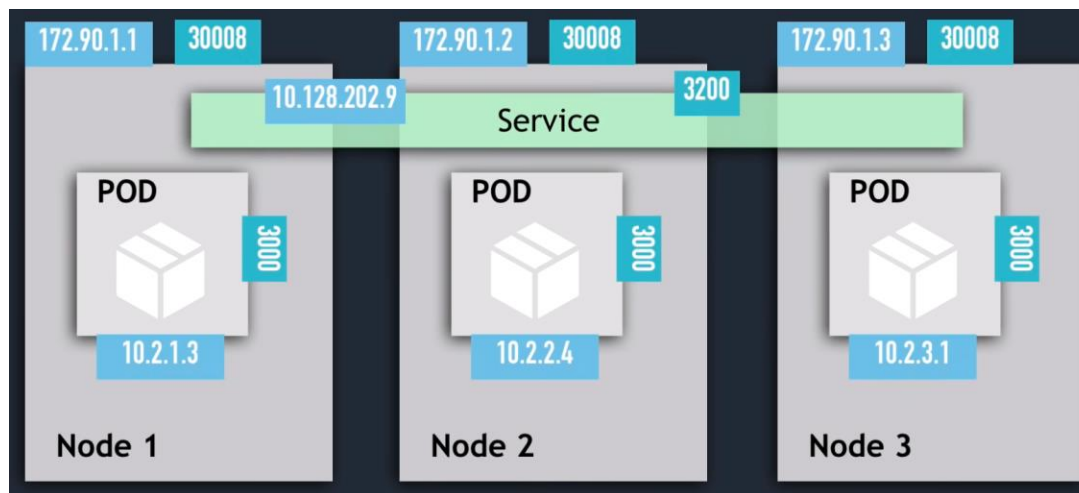
Selector: metadatos en forma de pares clave-valor que permite configurar los *pods* gestionados por un determinado servicio mediante un *selector* que especifica sus etiquetas o *labels*



Objetos de K8s

29

- **Servicios *NodePort* y *LoadBalancer***





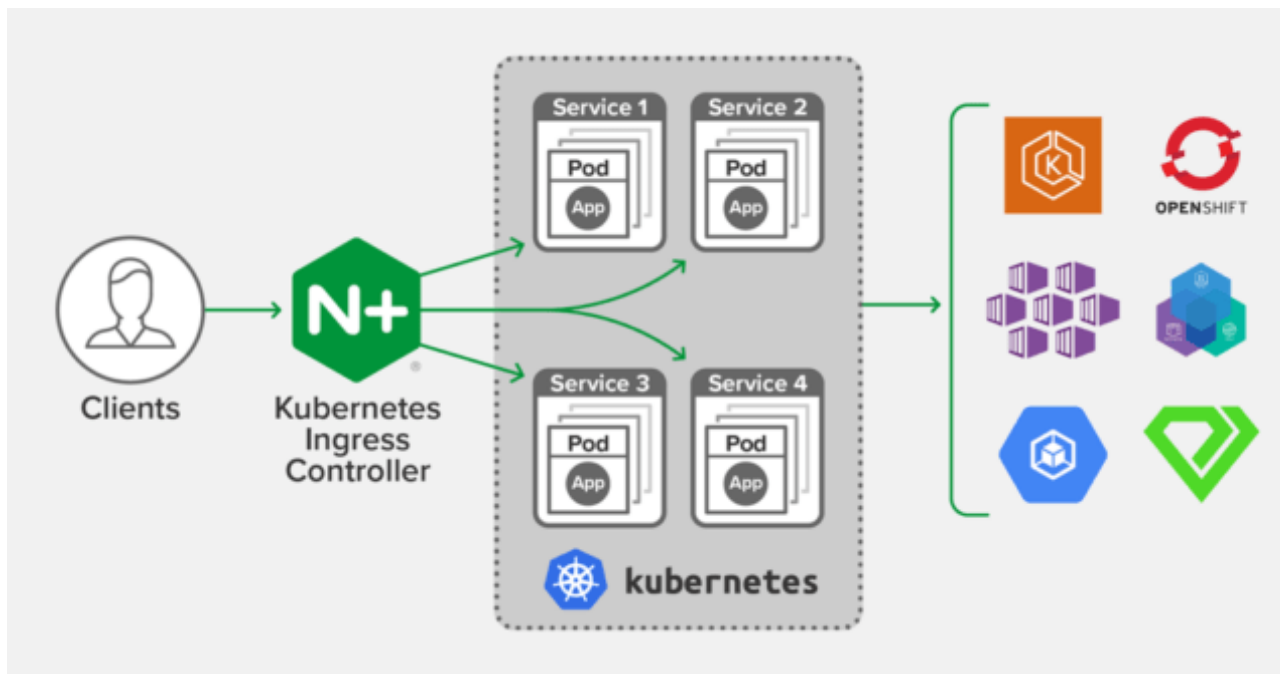
● *Ingress*

- Expone rutas HTTP/HTTPS desde el exterior del clúster K8s a los servicios en ejecución dentro del mismo
 - Proporciona un **punto único de entrada** al clúster y permite definir hacia dónde debe **enrutarse** el tráfico entrante
 - Permite consolidar las reglas de enrutamiento en un único recurso, ya que puede **exponer múltiples servicios bajo la misma dirección IP**
 - Aunque no es un tipo de *Service*, también se puede usar para exponer una aplicación ya que actúa como punto de entrada al clúster desde el exterior
- Consta de dos partes diferenciadas: **controlador y recurso**
 - El **controlador *Ingress*** es el responsable de realizar el enrutamiento del tráfico entrante a las rutas correctas en función de un conjunto de **reglas** definidas
 - El enrutamiento está controlado por las **reglas** definidas en un **recurso *Ingress***
- Por defecto, K8s no integra una implementación del controlador *Ingress*
 - K8s mantiene [Nginx Ingress Controller](#) (existen [implementaciones de terceros](#))
 - En entornos *cloud* compatibles, disponemos de implementaciones que se pueden combinar con un balanceador de carga externo proporcionado por el proveedor



Objetos de K8s

- **Ingress**






Contenidos

- Objetivo
- Arquitectura de un clúster K8s
- Objetos básicos de K8s
- **Ejercicios propuestos**




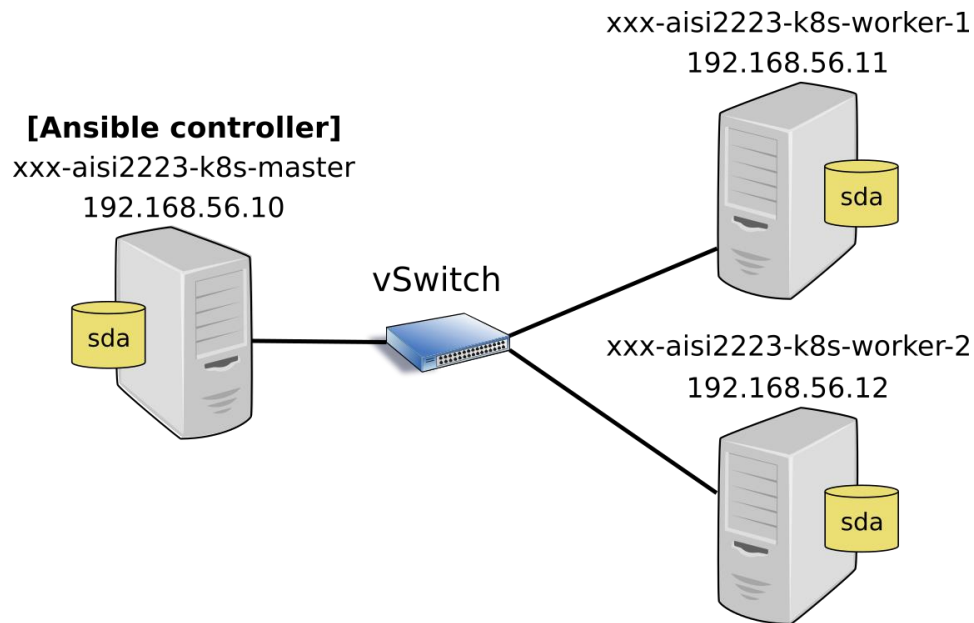
Ejercicio 1: Despliegue del clúster virtual

- Clona el [repositorio de la práctica 4](#) para obtener los ficheros necesarios para realizar los ejercicios propuestos
 - **Recuerda:** sin espacios, acentos, eñes o caracteres “raros” en la ruta 
- En cuanto a los recursos del clúster virtual, **no es necesario modificar** el *Vagrantfile* si tu *host* tiene al menos 4 GiB de memoria
 - Si dispones de menos recursos, ajusta la variable **NUM_WORKERS** en el *Vagrantfile* (línea 20) para desplegar **un único nodo worker**
 - **NO cambies la memoria** de las VMs ni ningún otro parámetro




Ejercicio 1: Despliegue del clúster virtual

- El despliegue del clúster K8s está formado por:
 - Una VM que hará las funciones de **nodo master y controlador Ansible**
 - Dos VMs (o una) que harán las funciones de **nodos worker**



Disponemos de conectividad entre las VMs a nivel de IP y *hostname*. La conectividad *ssh* desde el controlador Ansible hacia los nodos *worker* está configurado para hacerse sin introducir *password* (*ssh passwordless*)



Los *pods* que se ejecuten en los nodos *worker* del clúster tendrán direcciones IP asignadas dinámicamente por K8s en su creación dentro del rango de red **10.10.1.0/24**. Este rango es configurable y se establece cuando se inicializa K8s por primera vez durante el despliegue



Ejercicio 1: Despliegue del clúster virtual

- **Modifica el *Vagrantfile* para cambiar el *hostname* de las VMs**
 - **Modifica las variables *MASTER_HOSTNAME* y *WORKER_HOSTNAME***
 - **Debes sustituir "xxx" por tus iniciales correspondientes**
- **Edita también el fichero de inventario de Ansible (*ansible.inventory*) para sustituir "xxx" por tus iniciales correspondientes**
 - Solo si es necesario, ajusta el fichero en función de si tienes uno o dos nodos *worker*
- Fíjate en los grupos que se definen en el inventario:
 - El grupo ***masters*** incluye el nodo *master*
 - El grupo ***workers*** incluye los nodos *worker*
 - El grupo ***cluster*** está formado por los dos grupos anteriores



```
[masters]
rre-aisi2223-k8s-master
[workers]
rre-aisi2223-k8s-worker-1
rre-aisi2223-k8s-worker-2
# Group 'cluster' with all nodes
[cluster:children]
masters
workers
```



Ejercicio 1: Despliegue del clúster virtual

- El **primer despliegue** del clúster virtual con Vagrant se realizará en **dos fases**:
 1. *vagrant up --provision-with shell*
 - Despliega las VMs del clúster y las aprovisiona con un *script shell* que realiza la configuración *ssh passwordless* entre las VMs recomendada por Ansible
 2. *vagrant provision --provision-with ansible_local*
 - Aprovisiona las VMs con playbooks de Ansible para instalar Docker y K8s
- Ejecuta la **primera fase del despliegue** y al terminar conéctate al nodo *master* (*vagrant ssh*) para comprobar la conectividad *ssh passwordless* con los *workers*

```
vagrant@rre-aisi2223-k8s-master:~$ ssh rre-aisi2223-k8s-worker-1
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-137-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information disabled due to load higher than 1.0

14 updates can be applied immediately.
14 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable

New release '22.04.1 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Mon Jan 30 11:45:48 2023 from 192.168.56.10
vagrant@rre-aisi2223-k8s-worker-1:~$ exit
logout
Connection to rre-aisi2223-k8s-worker-1 closed.
vagrant@rre-aisi2223-k8s-master:~$
```



Comprueba también la conectividad *ssh* desde el nodo *master* hacia el segundo nodo *worker*



Ejercicio 1: Despliegue del clúster virtual

- Desde tu *host*, ejecuta la **segunda fase del despliegue** para aprovisionar las VMs con playbooks de Ansible

Resumen de las tareas
ejecutadas. Revisa que
no haya errores



```
PLAY RECAP *****
rre-aisi2223-k8s-master      : ok=30    changed=23    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
rre-aisi2223-k8s-worker-1   : ok=21    changed=14    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
rre-aisi2223-k8s-worker-2   : ok=21    changed=14    unreachable=0    failed=0    skipped=1    rescued=0    ignored=0
```

- Al terminar el aprovisionamiento, conéctate de nuevo por *ssh* al nodo *master* y ejecuta los siguientes comandos usando el CLI de K8s (*kubectl*):
 - kubectl get nodes -o wide*
 - kubectl cluster-info*



Ejercicio 1: Despliegue del clúster virtual



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
rre-aisi2223-k8s-master	Ready	control-plane	4m3s	v1.26.1	192.168.56.10	<none>	Ubuntu 20.04.5 LTS	5.4.0-137-generic	containerd://1.6.15
rre-aisi2223-k8s-worker-1	Ready	<none>	3m12s	v1.26.1	192.168.56.11	<none>	Ubuntu 20.04.5 LTS	5.4.0-137-generic	containerd://1.6.15
rre-aisi2223-k8s-worker-2	Ready	<none>	3m11s	v1.26.1	192.168.56.12	<none>	Ubuntu 20.04.5 LTS	5.4.0-137-generic	containerd://1.6.15

```
vagrant@rre-aisi2223-k8s-master:~$
```

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl cluster-info
```

Kubernetes control plane is running at <https://192.168.56.10:6443>

CoreDNS is running at <https://192.168.56.10:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy>

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

```
vagrant@rre-aisi2223-k8s-master:~$
```



Una vez desplegado el clúster K8s de forma exitosa por primera vez, deberás usar *vagrant halt/suspend* y *vagrant up* para detener e iniciar las VMs, respectivamente



Ejercicio 1: Despliegue del clúster virtual

- Para probar el funcionamiento del clúster, crea un **pod** que ejecute un servidor Nginx usando el siguiente comando:

- `kubectl apply -f /vagrant/ej1/nginx-pod.yml` ←

Fichero YAML (o manifiesto) que indica el tipo de objeto a crear, sus metadatos y la especificación de dicho objeto. Ver imagen inferior

Versión del API. Cada objeto puede usar un valor diferente

Tipo de objeto a crear

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-pod
spec:
```

Metadatos del objeto en forma de pares clave-valor. En este ejemplo, le asignamos un nombre al *pod*



Para listar todos los recursos del API disponibles y la versión del API para cada tipo de objeto, puedes ejecutar el comando: `kubectl api-resources`

```
  containers:
  - name: nginx-container
    image: nginx
    ports:
    - containerPort: 80
```

Especificación del objeto (el estado deseado). Los atributos a configurar dependerán del tipo de objeto. En este ejemplo, para un *pod* indicamos el nombre del contenedor, la imagen a usar del *Docker Hub* y el puerto en el que escucha el contenedor que se ejecuta dentro del *pod*



Es posible crear objetos de K8s de forma imperativa sin usar ficheros de manifiesto. Para crear el *pod* del ejemplo previo ejecutaríamos: `kubectl run nginx-pod --image=nginx --port=80`

Para modificar el objeto creado, podríamos ejecutar: `kubectl edit pod nginx-pod`. También es posible obtener un fichero YAML del objeto creado que podremos modificar posteriormente, tal y como se explica [aquí](#).



Ejercicio 1: Despliegue del clúster virtual



- Para ver los *pods* en ejecución, ejecuta:
 - `kubectl get pods -o wide`
- Obtén información detallada del *pod*:
 - `kubectl describe pod nginx-pod`

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl apply -f /vagrant/ej1/nginx-pod.yml
pod/nginx-pod created
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
NAME          READY   STATUS    RESTARTS   AGE   IP            NODE                   NOMINATED NODE
nginx-pod     1/1     Running   0           4s    10.10.1.130   rre-aisi2223-k8s-worker-1   <none>
vagrant@rre-aisi2223-k8s-master:~$ kubectl describe pod nginx-pod
Name:          nginx-pod
Namespace:     default
Priority:       0
Service Account: default
Node:          rre-aisi2223-k8s-worker-1/192.168.56.11
Start Time:    Mon, 30 Jan 2023 12:05:27 +0000
Labels:        <none>
Annotations:   cni.projectcalico.org/containerID: c1ab98a86a219a0a47eedecdc53b88729fd8157cb694fd6c0d!
               cni.projectcalico.org/podIP: 10.10.1.130/32
               cni.projectcalico.org/podIPs: 10.10.1.130/32
Status:        Running
IP:            10.10.1.130
IPs:           IP: 10.10.1.130
Containers:
  nginx-container:
    Container ID:  containerd://a3939c37e40b556ab83bd4163e9a9a757458
    Image:         nginx
    Image ID:      docker.io/library/nginx@sha256:b8f2383a95879e1ae0
    Port:         80/TCP
    Host Port:     0/TCP
    State:         Running
      Started:     Mon, 30 Jan 2023 12:05:28 +0000
    Ready:         True
    Restart Count: 0
    Environment:   <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-64xbx (ro)
```

← Dirección IP del pod dentro del rango 10.10.1.0/24



Es normal que la primera vez tarde unos 15-20 segundos en ejecutar el *pod* (que veas el estado *Running*) ya que K8s tiene que descargar la imagen del *Docker Hub*



Ejercicio 1: Despliegue del clúster virtual

- Realiza las siguientes pruebas de conectividad:

- `ping -c 2 10.10.1.130`
- `ping -c 2 nginx-pod`
- `curl 10.10.1.130:80`
- `curl xxx-aisi2223-k8s-worker-x`
- `curl 192.168.56.11:80`



De acuerdo a la transparencia previa, deberás usar la dirección IP de tu *pod*, así como la dirección IP y el *hostname* del nodo *worker* que ejecute dicho *pod*

Accede desde el navegador web de tu *host* a la IP del *pod* y a la IP del nodo *worker* que lo ejecuta. ¿Obtienes respuesta?



```
vagrant@rre-aisi2223-k8s-master:~$ ping -c 2 10.10.1.130
PING 10.10.1.130 (10.10.1.130) 56(84) bytes of data.
64 bytes from 10.10.1.130: icmp_seq=1 ttl=63 time=0.450 ms
64 bytes from 10.10.1.130: icmp_seq=2 ttl=63 time=0.389 ms

--- 10.10.1.130 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1007ms
rtt min/avg/max/mdev = 0.389/0.419/0.450/0.030 ms
vagrant@rre-aisi2223-k8s-master:~$ ping -c 2 nginx-pod
ping: nginx-pod: Temporary failure in name resolution
vagrant@rre-aisi2223-k8s-master:~$ curl 10.10.1.130:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
vagrant@rre-aisi2223-k8s-master:~$ curl rre-aisi2223-k8s-worker-1
curl: (7) Failed to connect to rre-aisi2223-k8s-worker-1 port 80: Connection refused
vagrant@rre-aisi2223-k8s-master:~$ curl 192.168.56.11:80
curl: (7) Failed to connect to 192.168.56.11 port 80: Connection refused
vagrant@rre-aisi2223-k8s-master:~$
```



Ejercicio 1: Despliegue del clúster virtual

- Para ver los *logs* del *pod* ejecuta:

- `kubectl logs nginx-pod`
- Alternativamente, podríamos haber ejecutado: `kubectl logs pod/nginx-pod`

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl logs nginx-pod
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2023/01/30 12:05:28 [notice] 1#1: using the "epoll" event method
2023/01/30 12:05:28 [notice] 1#1: nginx/1.23.3
2023/01/30 12:05:28 [notice] 1#1: built by gcc 10.2.1 20210110 (Debian 10.2.1-6)
2023/01/30 12:05:28 [notice] 1#1: OS: Linux 5.4.0-137-generic
2023/01/30 12:05:28 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2023/01/30 12:05:28 [notice] 1#1: start worker processes
2023/01/30 12:05:28 [notice] 1#1: start worker process 28
10.10.1.192 - - [31/Jan/2023:09:16:58 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.68.0" "-"
vagrant@rre-aisi2223-k8s-master:~$
```



Los recursos de K8s pueden referenciarse como: **tipo/nombre**. El tipo de recurso acepta **nombres cortos**. Aquí podríamos haber usado "po" como abreviatura de *pod*. [Aquí](#) puedes consultar los nombres cortos para otros recursos

- Finalmente, para eliminar el *pod* ejecuta:

- `kubectl delete -f /vagrant/ej1/nginx-pod.yml`
- Alternativamente, podríamos haber ejecutado cualquiera de estos comandos:
 - `kubectl delete pod nginx-pod`
 - `kubectl delete pod/nginx-pod`



Ejercicio 2: *ReplicaSet* y *Deployment*

- Crea un ***ReplicaSet*** que gestione un *pod* con **dos réplicas** que ejecuten un contenedor Nginx usando el siguiente comando:

- `kubectl apply -f /vagrant/ej2/nginx-replicaset.yml`

Fíjate que la API a usar es diferente para un *pod* y un *ReplicaSet*

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: nginx-replicaset
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx
        ports:
        - containerPort: 80
```

El número de réplicas es uno de los atributos que podemos especificar para un *ReplicaSet*. Las réplicas se crean a partir de una plantilla (*template*) que contiene metadatos y su especificación (*spec*)

El atributo *selector* permite indicar qué *Pods* gestiona un objeto *ReplicaSet* en función de sus etiquetas. En este ejemplo, cualquier *pod* etiquetado como "app: nginx" será gestionado por este *ReplicaSet*

El atributo *template* permite especificar una plantilla con la que serán desplegados los *Pods*. Estos *Pods* se etiquetan como "app: nginx" de forma que serán gestionados por el *ReplicaSet*. La especificación de cada réplica es la misma que usamos para crear el *pod* en el ejercicio anterior

- Obtén información detallada del *ReplicaSet* y los *Pods* en ejecución:

- `kubectl describe replicaset nginx-replicaset`
- `kubectl get pods -o wide`



Ejercicio 2: ReplicaSet y Deployment



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl apply -f /vagrant/ej2/nginx-replicaset.yml
replicaset.apps/nginx-replicaset created
vagrant@rre-aisi2223-k8s-master:~$ kubectl describe replicaset nginx-replicaset
Name:          nginx-replicaset
Namespace:     default
Selector:      app=nginx
Labels:        <none>
Annotations:   <none>
Replicas:      2 current / 2 desired
Pods Status:   2 Running / 0 Waiting / 0 Succeeded / 0 Failed
Pod Template:
  Labels:  app=nginx
  Containers:
    nginx-container:
      Image:      nginx
      Port:       80/TCP
      Host Port:  0/TCP
      Environment: <none>
      Mounts:      <none>
      Volumes:     <none>
Events:
  Type     Reason              Age   From                      Message
  ----     -
  Normal   SuccessfulCreate    5s    replicaset-controller     Created pod: nginx-replicaset-m4phl
  Normal   SuccessfulCreate    5s    replicaset-controller     Created pod: nginx-replicaset-m6l2s
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE
nginx-replicaset-m4phl              1/1     Running   0          10s   10.10.1.132    rre-aisi2223-k8s-worker-1
nginx-replicaset-m6l2s              1/1     Running   0          10s   10.10.1.3      rre-aisi2223-k8s-worker-2
vagrant@rre-aisi2223-k8s-master:~$
```

Comprueba con *ping* y *curl* si ambas réplicas del *pod* son accesibles por nombre y/o por IP



- Elimina ambos *pods* con *kubectl delete* y vuelve a listar los *pods* en ejecución
 - Ver siguiente transparencia
 - ¿Qué comportamiento observas?
 - ¿Qué deduces del nombrado de *pods* y sus IPs?



Ejercicio 2: ReplicaSet y Deployment

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl delete pod nginx-replicaset-m4phl nginx-replicaset-m6l2s
pod "nginx-replicaset-m4phl" deleted
pod "nginx-replicaset-m6l2s" deleted
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
nginx-replicaset-4s9rm              1/1     Running   0           5s    10.10.1.4       rre-aisi2223-k8s-worker-2
nginx-replicaset-nz4cd              1/1     Running   0           5s    10.10.1.133     rre-aisi2223-k8s-worker-1
vagrant@rre-aisi2223-k8s-master:~$
```

- Escala ahora el número de réplicas a 3 y vuelve a listar los *pods* en ejecución

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl scale --replicas=3 rs/nginx-replicaset
replicaset.apps/nginx-replicaset scaled
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE
nginx-replicaset-4s9rm              1/1     Running   0           2m29s  10.10.1.4       rre-aisi2223-k8s-worker-2
nginx-replicaset-cn7w               1/1     Running   0           5s     10.10.1.134     rre-aisi2223-k8s-worker-1
nginx-replicaset-nz4cd              1/1     Running   0           2m29s  10.10.1.133     rre-aisi2223-k8s-worker-1
vagrant@rre-aisi2223-k8s-master:~$
```

- Obtén información sobre el estado actual de los *ReplicaSet*
 - DESIRED: número deseado de réplicas (*pods*) de la aplicación
 - CURRENT: número de *pods* en ejecución
 - READY: número de *pods* disponibles a los usuarios

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get replicaset -o wide
NAME            DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES   SELECTOR
nginx-replicaset 3          3         3       87m   nginx-container  nginx    app=nginx
vagrant@rre-aisi2223-k8s-master:~$
```

- Finalmente, para eliminar el *ReplicaSet* ejecuta:
 - `kubectl delete -f /vagrant/ej2/nginx-replicaset.yml`



Ejercicio 2: *ReplicaSet* y *Deployment*

- Crea un **Deployment** que gestione un *ReplicaSet* de un *pod* con **dos réplicas** que ejecuten un contenedor Nginx usando el siguiente comando:

- `kubectl apply -f /vagrant/ej2/nginx-deployment.yml`

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx-container
        image: nginx:1.20.2
        ports:
        - containerPort: 80
```

Fíjate que, respecto al manifiesto del *ReplicaSet*, simplemente hemos cambiado el tipo de objeto a crear y su nombre. Adicionalmente, en este caso también especificamos una versión concreta de la imagen nginx que usarán los contenedores a ejecutar con el objetivo de ejemplificar una de las funcionalidades interesantes que permite un *Deployment* como son las actualizaciones sin interrupción de servicio

- Obtén información detallada del *Deployment* y de los *pods* en ejecución:
 - `kubectl describe deployment nginx-deployment`
 - `kubectl get pods -o wide`



Ejercicio 2: ReplicaSet y Deployment



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl describe deployment nginx-deployment
```

```
Name: nginx-deployment
Namespace: default
CreationTimestamp: Tue, 31 Jan 2023 10:58:38 +0000
Labels: <none>
Annotations: deployment.kubernetes.io/revision: 1
Selector: app=nginx
Replicas: 2 desired | 2 updated | 2 total | 2 available | 0 unavailable
```

```
StrategyType: RollingUpdate
MinReadySeconds: 0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
```

Estrategia de actualización del Deployment. Ver transparencia siguiente

```
Pod Template:
```

```
Labels: app=nginx
```

```
Containers:
```

```
nginx-container:
```

```
Image: nginx:1.20.2
```

```
Port: 80/TCP
```

```
Host Port: 0/TCP
```

```
Environment: <none>
```

```
Mounts: <none>
```

```
Volumes: <none>
```

```
Conditions:
```

```
Type Status Reason
```

```
----
```

```
Available True MinimumReplicasAvailable
```

```
Progressing True NewReplicaSetAvailable
```

```
OldReplicaSets: <none>
```

```
NewReplicaSet: nginx-deployment-584bcf88fc (2/2 replicas created)
```

ReplicaSet gestionado por el Deployment

```
Events:
```

```
Type Reason Age From Message
```

```
----
```

```
Normal ScalingReplicaSet 2m28s deployment-controller Scaled up replica set nginx-deployment-584bcf88fc to 2
```

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
```

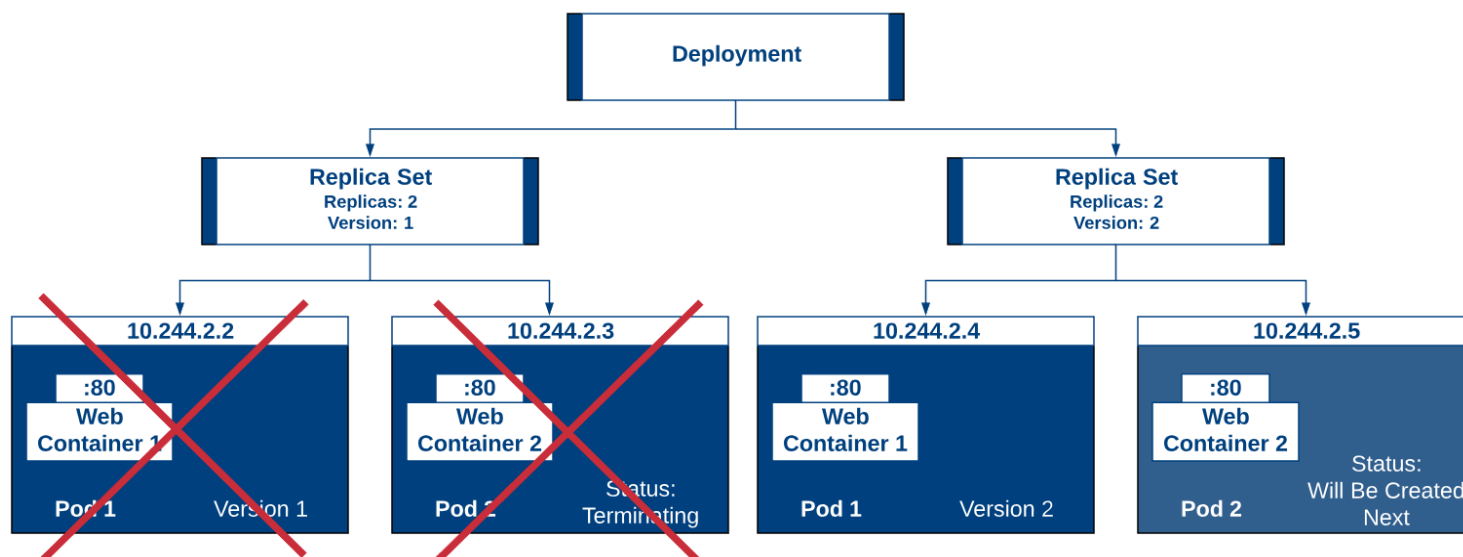
NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-deployment-584bcf88fc-ntnvc	1/1	Running	0	2m29s	10.10.1.5	rre-aisi2223-k8s-worker-2
nginx-deployment-584bcf88fc-slfq5	1/1	Running	0	2m29s	10.10.1.135	rre-aisi2223-k8s-worker-1

```
vagrant@rre-aisi2223-k8s-master:~$
```



Ejercicio 2: *ReplicaSet* y *Deployment*

- Un *Deployment* proporciona **actualizaciones declarativas** de *Pods* y *ReplicaSets*
- La **estrategia** por defecto es ***RollingUpdate***, la cual nos permite reemplazar la versión existente de los *Pods* con una nueva versión **sin interrupción de servicio**
 - Garantiza que solo una cierta cantidad de *Pods* estén inactivos mientras se actualiza la aplicación. Por defecto, se ejecutarán al menos el 75% del número deseado de *Pods*, es decir, como máximo el 25% de los *Pods* estarán inactivos (***maxUnavailable* = 25%**)
 - Garantiza que solo se desplieguen una cierta cantidad de *Pods* por encima del número deseado. Por defecto, estarán activos un máximo del 125% del número deseado de *Pods*, es decir, se permite un aumento máximo del 25% (***maxSurge* = 25%**)





Ejercicio 2: *ReplicaSet* y *Deployment*

- Obtén información sobre el estado actual de los *Deployment*
 - READY: número de réplicas (*pods*) disponibles en función del número deseado
 - UP-TO-DATE: número de *pods* actualizados para alcanzar el estado deseado
 - AVAILABLE: número de *pods* disponibles a los usuarios

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get deployments -o wide
NAME                READY    UP-TO-DATE    AVAILABLE    AGE    CONTAINERS    IMAGES    SELECTOR
nginx-deployment    2/2      2             2            5m23s  nginx-container  nginx:1.20.2  app=nginx
vagrant@rre-aisi2223-k8s-master:~$
```

- Modifica el manifiesto del *Deployment* de la siguiente manera:
 - Incrementa el número de réplicas de 2 a 4
 - Establece explícitamente la estrategia de actualización a *RollingUpdate*
 - Configura los parámetros de la estrategia *RollingUpdate* de la siguiente forma:
 - Establece a 1 el número de *pods* que pueden estar inactivos en un momento dado (*maxUnavailable*) y el número de *pods* que se despliegan al mismo tiempo (*maxSurge*)
 - Estos dos parámetros admiten valores numéricos y porcentajes
 - Actualiza la imagen de nginx a la versión 1.21.6



Ejercicio 2: *ReplicaSet* y *Deployment*

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 4
  strategy:
    type: RollingUpdate
    rollingUpdate: #Update Pods a certain number at a time
      #Total number of pods that can be unavailable at once
      maxUnavailable: 1
      #Maximum number of pods that can be deployed above desired state
      maxSurge: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.21.6
          ports:
            - containerPort: 80
```



Al igual que un *ReplicaSet*, podemos **escalar** un *Deployment* usando *kubectl* (ver transparencia 45). Ejemplo (no lo ejecutes):

```
kubectl scale --replicas=4 deploy/nginx-deployment
```

- Aplica los cambios para que se actualice el *Deployment* y obtén información sobre los *Deployment*, los *ReplicaSet* y los pods:
 - `kubectl apply -f /vagrant/ej2/nginx-deployment.yml`
 - `kubectl get deployments -o wide`
 - `kubectl get replicaset -o wide`
 - `kubectl get pods -o wide`



Ejercicio 2: ReplicaSet y Deployment



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl apply -f /vagrant/ej2/nginx-deployment.yml
deployment.apps/nginx-deployment configured
vagrant@rre-aisi2223-k8s-master:~$ kubectl get deployments -o wide
NAME                READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS        IMAGES              SELECTOR
nginx-deployment    3/4     2             3           9m15s  nginx-container    nginx:1.21.6        app=nginx
vagrant@rre-aisi2223-k8s-master:~$ kubectl get deployments -o wide
NAME                READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS        IMAGES              SELECTOR
nginx-deployment    3/4     3             3           9m27s  nginx-container    nginx:1.21.6        app=nginx
vagrant@rre-aisi2223-k8s-master:~$ kubectl get deployments -o wide
NAME                READY   UP-TO-DATE   AVAILABLE   AGE    CONTAINERS        IMAGES              SELECTOR
nginx-deployment    4/4     4             4           9m46s  nginx-container    nginx:1.21.6        app=nginx
vagrant@rre-aisi2223-k8s-master:~$ kubectl get replicaset -o wide
NAME                DESIRED   CURRENT   READY   AGE    CONTAINERS        IMAGES              SELECTOR
nginx-deployment-584bcf88fc  0         0         0       9m55s  nginx-container    nginx:1.20.2        app=nginx,pc
nginx-deployment-5c8958485d  4         4         4       48s    nginx-container    nginx:1.21.6        app=nginx,pc
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE    IP             NODE
nginx-deployment-5c8958485d-99sg4  1/1    Running   0          35s    10.10.1.8      rre-aisi2223-k8s-worker-2
nginx-deployment-5c8958485d-9tk4t  1/1    Running   0          55s    10.10.1.7      rre-aisi2223-k8s-worker-2
nginx-deployment-5c8958485d-cgkk4  1/1    Running   0          42s    10.10.1.138    rre-aisi2223-k8s-worker-1
nginx-deployment-5c8958485d-p7k5t  1/1    Running   0          55s    10.10.1.136    rre-aisi2223-k8s-worker-1
vagrant@rre-aisi2223-k8s-master:~$
```

Comprueba si puedes acceder al servidor Nginx desde el navegador web de tu *host*. Prueba usando las IPs de los *pods* y las IPs de los nodos *worker*



- Finalmente, para eliminar el *Deployment* ejecuta:
 - `kubectl delete -f /vagrant/ej2/nginx-deployment.yml`



Ejercicio 3: Service

52

- Crea un **Service** de tipo *ClusterIP* que permita exponer el acceso al servidor web Nginx mediante el *Deployment* del ejercicio anterior:
 - `kubectl apply -f /vagrant/ej2/nginx-deployment.yml`
 - `kubectl apply -f /vagrant/ej3/nginx-service.yml`

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: ClusterIP
  ports:
  - name: http
    port: 8080
    targetPort: 80
    protocol: TCP
  selector:
    app: nginx
```

← Tipo de servicio (ver transparencia 28)

← El servicio mapea peticiones del puerto 8080 (*port*) al puerto 80 de cualquier *pod* (*targetPort*) etiquetado como "app: nginx" (*selector*). Ver las *labels* en la definición del *Deployment*

- Obtén información detallada del *Service* y los *pods* en ejecución:
 - `kubectl describe service nginx-service`
 - `kubectl get pods -o wide`



Ejercicio 3: Service



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl describe service nginx-service
```

```
Name:          nginx-service
Namespace:     default
Labels:        <none>
Annotations:   <none>
Selector:      app=nginx
Type:          ClusterIP
IP Family Policy: SingleStack
IP Families:   IPv4
IP:            10.107.24.187
IPs:           10.107.24.187
Port:          http 8080/TCP
TargetPort:    80/TCP
Endpoints:     10.10.1.10:80,10.10.1.139:80,10.10.1.140:80 + 1 more...
Session Affinity: None
Events:        <none>
```

Lista de direcciones o **endpoints** a las que un **Service** enviará tráfico. Cuando el **selector** de un **Service** coincide con la etiqueta de un **pod**, esa dirección IP se agrega a sus **endpoints**

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
nginx-deployment-5c8958485d-djdv	1/1	Running	0	37s	10.10.1.139	rre-aisi2223-k8s-worker-1
nginx-deployment-5c8958485d-ql5s	1/1	Running	0	37s	10.10.1.10	rre-aisi2223-k8s-worker-2
nginx-deployment-5c8958485d-vp7wt	1/1	Running	0	37s	10.10.1.140	rre-aisi2223-k8s-worker-1
nginx-deployment-5c8958485d-z6sns	1/1	Running	0	37s	10.10.1.9	rre-aisi2223-k8s-worker-2

```
vagrant@rre-aisi2223-k8s-master:~$
```

● Obtén información sobre el estado actual de los Service

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get services -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	23h	<none>
nginx-service	ClusterIP	10.107.24.187	<none>	8080/TCP	117s	app=nginx

Este primer servicio de tipo **ClusterIP** se crea por defecto y es necesario para el correcto funcionamiento de un clúster K8s

IP asignada al servicio que hemos creado para poder acceder a la aplicación Nginx independientemente de las IPs asignadas a los **pods**. Es una dirección IP interna al clúster. Por lo tanto, la aplicación no se puede acceder desde el exterior del mismo



Ejercicio 3: Service

54



- Accede al servidor Nginx mediante *curl* usando la *ClusterIP* del Service

```
vagrant@rre-aisi2223-k8s-master:~$ curl 10.107.24.187:8080
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
vagrant@rre-aisi2223-k8s-master:~$
```



Accede desde el navegador web de tu *host* a la IP del servicio y puerto 8080. ¿Obtienes respuesta?



Ejercicio 3: Service

55

- Crea un *pod* de prueba para comprobar el acceso al servicio mediante *curl*

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl run shell-pod --rm -it --image arunvelsriram/utils -- bash
```

```
If you don't see a command prompt, try pressing enter.
```

```
utils@shell-pod:~$ curl 10.107.24.187:8080
```

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<title>Welcome to nginx!</title>
```

```
<style>
```

```
html { color-scheme: light dark; }
```

```
body { width: 35em; margin: 0 auto;
```

```
font-family: Tahoma, Verdana, Arial, sans-serif; }
```

```
</style>
```

```
</head>
```

```
<body>
```

```
<h1>Welcome to nginx!</h1>
```

```
<p>If you see this page, the nginx web server is successfully installed and working. Further configuration is required.</p>
```

```
<p>For online documentation and support please refer to
```

```
<a href="http://nginx.org/">nginx.org</a>.<br>
```

```
Commercial support is available at
```

```
<a href="http://nginx.com/">nginx.com</a>.</p>
```

```
<p><em>Thank you for using nginx.</em></p>
```

```
</body>
```

```
</html>
```

```
utils@shell-pod:~$ nslookup 10.107.24.187
```

```
187.24.107.10.in-addr.arpa      name = nginx-service.default.svc.cluster.local.
```

```
utils@shell-pod:~$ nslookup nginx-service.default.svc.cluster.local
```

```
Server:      10.96.0.10
```

```
Address:     10.96.0.10#53
```

```
Name:   nginx-service.default.svc.cluster.local
```

```
Address: 10.107.24.187
```

```
utils@shell-pod:~$ exit
```

```
exit
```

```
Session ended, resume using 'kubectl attach shell-pod -c shell-pod -i -t' command when the pod is running
```

```
pod "shell-pod" deleted
```

```
vagrant@rre-aisi2223-k8s-master:~$
```

Creamos un *pod* de forma interactiva ejecutando un *shell bash*. Fíjate en el cambio del *prompt*. Puede tardar unos segundos mientras se descarga la imagen del *Docker Hub*

Accedemos al servicio desde el *pod* usando *curl*

Resolvemos la *ClusterIP* del servicio usando *nslookup*. Fíjate en el *FQDN* del servicio

Hacemos la resolución inversa usando el *FQDN*

Salimos del *pod* terminando el *shell*. Esto elimina el *pod* debido al uso del parámetro *--rm* al crearlo

Puedes comprobar que la resolución funciona igualmente si usas su nombre (*nginx-service*), ya que tanto el *pod* como el servicio se ejecutan en el *namespace* por defecto (*default*), puesto que no hemos especificado ninguno a la hora de crear ambos recursos, Más adelante trabajaremos con *namespaces*





Ejercicio 3: Service

56

- Elimina el *Deployment* y el *Service* ejecutando:
 - `kubectl delete -f /vagrant/ej2/nginx-deployment.yml`
 - `kubectl delete -f /vagrant/ej3/nginx-service.yml`
- Modifica la especificación de la *template* del manifiesto del *Deployment* de la siguiente manera:

template:

```
metadata:
  labels:
    app: nginx
spec:
  containers:
    - name: nginx-container
      image: nginx:1.21.6
      ports:
        - containerPort: 80
```

```
    volumeMounts:
      - name: nginx-index-file
        mountPath: /usr/share/nginx/html/
```

```
volumes:
  - name: nginx-index-file
    configMap:
      name: index-configmap
```

Montamos un *Volume* de tipo *configMap* en la ruta donde Nginx sirve la página web por defecto

Añadimos un volumen de tipo *configMap*



Un *ConfigMap* provee una manera de inyectar datos de configuración a los *Pods* (ver transparencia 25). Los datos almacenados en un *ConfigMap* se pueden referenciar en un [volumen de tipo configMap](#) y luego ser consumidos por aplicaciones containerizadas corriendo en un *pod*



Ejercicio 3: Service

57

- Modifica el manifiesto del Service de la siguiente manera:
 - Cambiar el tipo de servicio de *ClusterIP* a *NodePort*
 - En la sección *ports* de la especificación del servicio, añade el parámetro *nodePort* con valor 30001

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  type: NodePort
  ports:
  - name: http
    port: 8080
    targetPort: 80
    nodePort: 30001
    protocol: TCP
  selector:
    app: nginx
```

Las peticiones al puerto 8080 (*port*) se mapean al puerto 30001 de cada nodo *worker* (*nodePort*), donde se mapean a su vez al puerto 80 de cualquier *pod* (*targetPort*) etiquetado como "app: nginx" (*selector*). Ver transparencias 28-29


Si no se establece *nodePort*, K8s asigna un puerto de forma aleatoria en el rango especificado en la transparencia 28





Ejercicio 3: Service

58

- Modifica el fichero *index.html* proporcionado en el repositorio de la práctica
 - **Debes sustituir "[alumno/a]" por tu nombre y apellido** 
- Crea el ConfigMap a partir del contenido del fichero *index.html* modificado:
 - `kubectl create configmap index-configmap --from-file=/vagrant/ej3/html/index.html`
- Crea de nuevo el Deployment y el Service:
 - `kubectl apply -f /vagrant/ej2/nginx-deployment.yml`
 - `kubectl apply -f /vagrant/ej3/nginx-service.yml`
- Obtén información detallada del Service y de todos los servicios en ejecución:
 - `kubectl describe service nginx-service`
 - `kubectl get services -o wide`



Ejercicio 3: Service



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl describe service nginx-service
```

```
Name: nginx-service
Namespace: default
Labels: <none>
Annotations: <none>
Selector: app=nginx
Type: NodePort
IP Family Policy: SingleStack
IP Families: IPv4
IP: 10.97.189.58
IPs: 10.97.189.58
Port: http 8080/TCP
TargetPort: 80/TCP
NodePort: http 30001/TCP
Endpoints: 10.10.1.11:80,10.10.1.12:80,10.10.1.142:80 + 1 more...
Session Affinity: None
External Traffic Policy: Cluster
Events: <none>
```

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get services -o wide
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	23h	<none>
nginx-service	NodePort	10.97.189.58	<none>	8080:30001/TCP	16s	app=nginx

```
vagrant@rre-aisi2223-k8s-master:~$
```

Un servicio de tipo *NodePort* tiene también asignado una dirección IP (una *ClusterIP*)

Puerto 8080 mapeado al puerto 30001 de cada nodo *worker*



Ejercicio 3: Service

- Accede al servidor Nginx mediante *curl* usando la *ClusterIP* del Service

```
vagrant@rre-aisi2223-k8s-master:~$ curl 10.97.189.58:8080
<html>
<head>
  <meta charset= "utf-8">
  <title>GEI AISI: Test Page</title>
  <script type="text/javascript">
    function getURL() {
      document.write("URL: " + window.location.href);
    }
    function getTime() {
      document.getElementById("current_date").innerHTML = Date();
    }
  </script>
</head>
<body>
  <div style="width:600px;height:200px;border:2px solid #000;text-align: center;">
    <strong><br>
    <u>GEI AISI: 2022/2023</u>
    <p><u>Nginx Web Server (K8s cluster)</u></p>
    <p>Página web de Roberto Rey Expósito</p>
    <p><script>getURL();</script></p>
    <p><div id="current_date"><script>getTime();</script></p>
  </strong>
</div>
</body>
</html>
vagrant@rre-aisi2223-k8s-master:~$
```



Accede al servicio con *curl* usando su *ClusterIP* y el puerto 30001. ¿Obtienes respuesta?



Ejercicio 3: Service

61



- Desde el navegador de tu *host*, accede a las siguientes URLs:

- <http://192.168.56.11:30001>
- <http://192.168.56.12:30001>



Fíjate que estas son las direcciones IP de los nodos *worker* (ver transparencia 34)

GEIAISI: Test Page x +

← → ↻ No es seguro 192.168.56.11:30001

GEIAISI: 2022/2023

Nginx Web Server (K8s cluster)

Página web de Roberto Rey Expósito

URL: <http://192.168.56.11:30001/>

Tue Sep 13 2022 17:50:24 GMT+0200 (hora de verano de Europa central)

GEIAISI: Test Page x +

← → ↻ No es seguro 192.168.56.12:30001

GEIAISI: 2022/2023

Nginx Web Server (K8s cluster)

Página web de Roberto Rey Expósito

URL: <http://192.168.56.12:30001/>

Tue Sep 13 2022 17:52:34 GMT+0200 (hora de verano de Europa central)



Prueba usando la IP del nodo *master* y el puerto 30001. ¿Obtienes respuesta?



Ejercicio 3: Service

62

- Pruebas adicionales que podrías hacer **desde el navegador de tu host**

- <http://192.168.56.10:30001>
- <http://192.168.56.10:8080>
- <http://192.168.56.11:8080>
- <http://xx.xx.xx.xx:30001>
- <http://xx.xx.xx.xx:8080>

Trata de razonar si el acceso debería funcionar en cada caso de prueba



← Sustituye xx por la dirección *ClusterIP* correspondiente a tu servicio de tipo *NodePort*

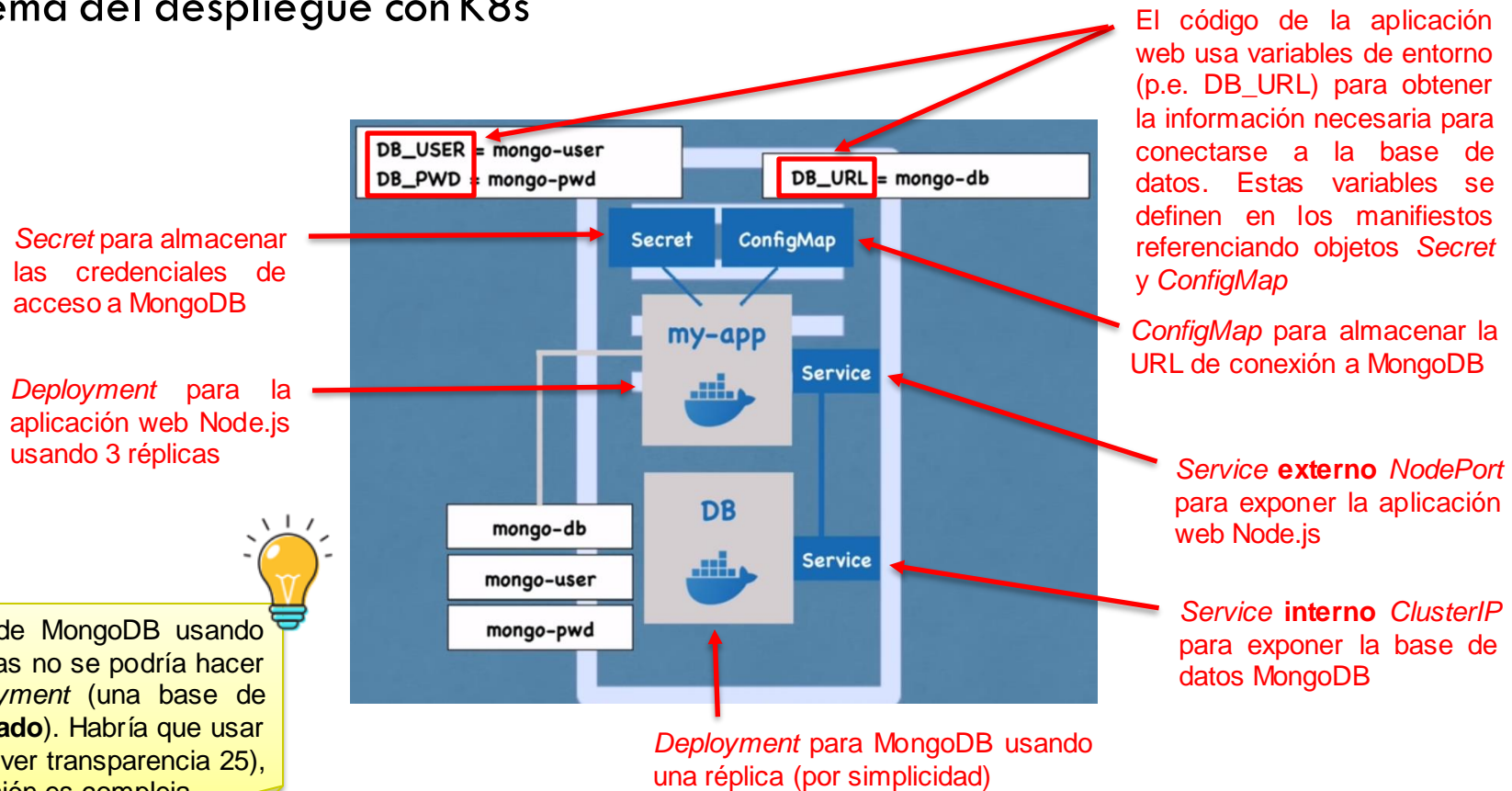
- Elimina todos los recursos: *ConfiMap*, *Deployment* y *Service*

- `kubectl delete configmap index-configmap`
- `kubectl delete -f /vagrant/ej2/nginx-deployment.yml`
- `kubectl delete -f /vagrant/ej3/nginx-service.yml`



Ejercicio 4: Despliegue de una aplicación web

- Despliegue de una aplicación web Node.js+Express+MongoDB
 - [MongoDB](#) es una base de datos distribuida de tipo NoSQL orientada a documentos que proporciona alta disponibilidad y escalabilidad horizontal
- Esquema del despliegue con K8s



El despliegue de MongoDB usando múltiples réplicas no se podría hacer con un *Deployment* (una base de datos tiene **estado**). Habría que usar un *StatefulSet* (ver transparencia 25), cuya configuración es compleja



Ejercicio 4: Despliegue de una aplicación web

- Desplegaremos la aplicación en un nuevo *namespace* llamado **aisi2223**
 - Los *namespaces* son una forma segura de dividir los recursos de un clúster K8s entre múltiples usuarios, aplicaciones, entornos, proyectos...
 - Hasta ahora hemos estado usando el *namespace* por defecto (*default*) ya que no especificábamos ninguno en concreto en los manifiestos
 - Al crear un objeto (p.e. *Service*) se puede especificar el *namespace* en la sección *metadata* del manifiesto (curiosear cualquiera de los ficheros de este ejercicio)
 - El *namespace* forma parte del FQDN de un objeto (ver transparencia 55), aunque objetos dentro del mismo *namespace* pueden referenciarse usando su nombre simple
 - Crear un nuevo *namespace* es muy sencillo:
 - De forma imperativa: `kubectl create aisi2223`
 - De forma declarativa:

```
apiVersion: v1
kind: Namespace
metadata:
  name: aisi2223
```
- Crea el nuevo *namespace* ejecutando el comando:
 - `kubectl apply -f /vagrant/ej4/namespace.yml`



Ejercicio 4: Despliegue de una aplicación web

- Curiosear los manifiestos del ejercicio para entender el despliegue
 - Especialmente la parte donde se definen las variables de entorno (USER_NAME, DB_URL) que referencian los objetos *ConfigMap* y *Secret*
- Despliega todos los objetos K8s de la aplicación usando los manifiestos
 - `kubectl apply -f /vagrant/ej4/mongodb-config.yml`
 - `kubectl apply -f /vagrant/ej4/mongodb-secret.yml`
 - `kubectl apply -f /vagrant/ej4/mongodb.yml`
 - `kubectl apply -f /vagrant/ej4/webapp.yml`
- Tras el despliegue, ejecuta los siguientes comandos:
 - `kubectl get namespaces`
 - `kubectl get services -o wide -n aisi2223`
 - `kubectl get deployments -o wide -n aisi2223`
 - `kubectl get pods -o wide -n aisi2223`
 - `kubectl get configmaps -n aisi2223`
 - `kubectl get secrets -n aisi2223`

Ahora tenemos que especificar nuestro *namespace* para listar sus objetos usando la opción *-n*. De lo contrario solo veríamos aquellos objetos que pertenecen al *namespace default*



Ejercicio 4: Despliegue de una aplicación web



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get namespaces
```

NAME	STATUS	AGE
aisi2223	Active	77s
default	Active	24h
kube-node-lease	Active	24h
kube-public	Active	24h
kube-system	Active	24h

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get services -o wide -n aisi2223
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE	SELECTOR
mongodb-service	ClusterIP	10.101.190.69	<none>	27017/TCP	72s	app=mongodb
webapp-service	NodePort	10.111.180.169	<none>	3000:30100/TCP	70s	app=webapp

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get deployments -o wide -n aisi2223
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE	CONTAINERS	IMAGES	SELECTOR
mongodb-deployment	1/1	1	1	72s	mongodb	mongo:5.0	app=mongodb
webapp-deployment	3/3	3	3	70s	webapp	nanajanashia/k8s-demo-app:v1.0	app=webapp

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -o wide -n aisi2223
```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE	NOMINATED
mongodb-deployment-5ccdfcd8f6-dgcrx	1/1	Running	0	72s	10.10.1.144	rre-aisi2223-k8s-worker-1	<none>
webapp-deployment-79847d649b-4kn5w	1/1	Running	0	70s	10.10.1.145	rre-aisi2223-k8s-worker-1	<none>
webapp-deployment-79847d649b-f2ztl	1/1	Running	0	70s	10.10.1.13	rre-aisi2223-k8s-worker-2	<none>
webapp-deployment-79847d649b-qjndq	1/1	Running	0	70s	10.10.1.146	rre-aisi2223-k8s-worker-1	<none>

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get configmaps -n aisi2223
```

NAME	DATA	AGE
kube-root-ca.crt	1	77s
mongodb-config	1	72s

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get secrets -n aisi2223
```

NAME	TYPE	DATA	AGE
mongodb-secret	Opaque	2	74s

```
vagrant@rre-aisi2223-k8s-master:~$
```



Ejercicio 4: Despliegue de una aplicación web

- Desde el navegador de tu *host*, accede a las siguientes URLs:

- <http://192.168.56.10:3000>
- <http://192.168.56.10:30100>
- <http://192.168.56.10:27017>
- <http://192.168.56.11:30100>
- <http://xx.xx.xx.xx:3000>
- <http://xx.xx.xx.xx:30100>



Sustituye xx por la dirección *ClusterIP* correspondiente al servicio de la aplicación web



¿En qué casos obtienes acceso a la aplicación web y por qué?



User profile



Name: **Anna Smith**

Email: **anna.smith@example.com**

Interests: **coding**

Edit Profile



Ejercicio 4: Despliegue de una aplicación web



- Modifica el perfil de usuario:
 - *Name:* **xxx-aisi2223** (debes sustituir "xxx" por tus iniciales correspondientes)
 - *Email:* tu dirección de la UDC
 - *Interests:* añade alguna afición/interés



User profile



Name: **rre-aisi2223**

Email: **rreye@udc.es**

Interests: **F1, fútbol y simracing**

Edit Profile

- Para terminar el ejercicio, elimina todos los recursos



Ejercicio 5: Despliegue multi-aplicación

- Desplegaremos dos aplicaciones en el clúster de K8s: el mismo servidor web Nginx del ejercicio 3 y otro servidor web diferente usando Apache
 - Para exponer ambas aplicaciones podríamos usar servicios de tipo *NodePort*, pero es una aproximación con múltiples limitaciones
 - *NodePort* permite exponer varios servicios bajo la misma IP pero solo un servicio por puerto
 - Cada servicio debe ser accedido mediante un número de puerto distinto
 - Solo se puede usar un rango determinado de puertos (por defecto: 30000-32767)
 - Si la dirección IP de un nodo cambia, hay que reconfigurar
 - La mejor forma de exponer múltiples servicios es mediante un *Ingress Controller*
 - Permite exponer múltiples servicios bajo la misma IP y múltiples servicios por puerto
 - Permite realizar enrutamientos basados en rutas y subdominios para los servicios
 - Por ejemplo, es posible enviar el tráfico entrante hacia `www.example.com/foo` al servicio `foo`, y el tráfico entrante hacia `www.example.com/bar` al servicio `bar`
 - Debemos instalar un controlador y crear un recurso que defina las reglas de enrutado
 - Instalaremos el controlador [*Nginx Ingress Controller*](#) (ver transparencia 30)
 - El controlador se ejecuta en el clúster como un *pod* y se expone asimismo al exterior como un servicio de tipo *NodePort* (en clústers K8s *bare metal*) o *LoadBalancer* (K8s en la nube)



Ejercicio 5: Despliegue multi-aplicación

- Instala el *Nginx Ingress Controller* para K8s *bare metal*:

- `kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.5.1/deploy/static/provider/baremetal/deploy.yaml`



Nuestro despliegue de un clúster virtual K8s se considera **bare metal**, aunque usemos VMs. En este contexto, K8s *bare metal* se refiere al despliegue e instalación de K8s desde cero, tal y como se ha hecho en esta práctica utilizando Ansible. En un clúster de servidores físicos (sin VMs) el despliegue de K8s se podría hacer utilizando el mismo aprovisionamiento con Ansible. La alternativa a un clúster *bare metal* es usar un **servicio gestionado de K8s en la nube** proporcionado por un proveedor *cloud*, siendo la opción más habitual actualmente, como por ejemplo: GCE ([GKE](#)), AWS ([EKS](#)), Azure ([AKS](#)). En ese caso, la instalación de *Nginx Ingress Controller* se haría usando un [manifiesto](#) distinto, pudiendo además configurarlo de forma específica en [función del proveedor cloud](#).

- Comprueba la instalación del controlador ejecutando el comando:

- `kubectl get pods -n=ingress-nginx` ← El controlador *Ingress* se configura por defecto en un nuevo *namespace* que se crea como parte del proceso de instalación

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get pods -n=ingress-nginx
NAME                                READY   STATUS    RESTARTS   AGE
ingress-nginx-admission-create-kr4s8 0/1     Completed 0           10m
ingress-nginx-admission-patch-rclnw   0/1     Completed 2           10m
ingress-nginx-controller-64f79ddbcc-xv6s 1/1     Running   0           10m
vagrant@rre-aisi2223-k8s-master:~$
```

Pod que ejecuta el controlador *Ingress* en el clúster K8s (si su estado no es *Running* espera unos segundos y vuelve a ejecutar el comando). Los otros dos *pods* se ejecutaron como parte de la instalación del controlador



Ejercicio 5: Despliegue multi-aplicación


- Comprueba ahora los servicios relacionados con el controlador *Ingress*:
 - `kubectl get services -n=ingress-nginx`

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get services -n=ingress-nginx
```

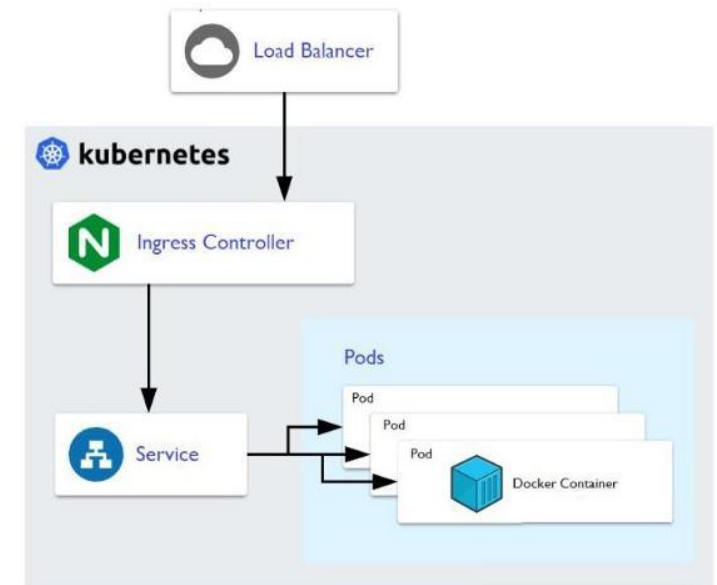
NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)
ingress-nginx-controller	NodePort	10.102.118.172	<none>	80:30544/TCP, 443:31511/TCP
ingress-nginx-controller-admission	ClusterIP	10.102.131.152	<none>	443/TCP

```
vagrant@rre-aisi2223-k8s-master:~$
```

El primer servicio, de tipo *NodePort*, expone el controlador *Ingress* en los puertos 30544 y 31511 (en este ejemplo) para tráfico HTTP y HTTPS, respectivamente. El segundo servicio se encarga de validar los recursos *Ingress* que crearemos posteriormente




En el caso de ejecutar K8s en la nube, el servicio del controlador *Ingress* sería de tipo *LoadBalancer* en vez de *NodePort*, dado que los proveedores *cloud* proporcionan bajo demanda balanceadores de carga externos (ver figura derecha). K8s no proporciona ninguna implementación de *LoadBalancer* para clusters *bare metal*, aunque existen implementaciones de terceros (p.e. [MetalLB](#)). La opción más simple y que se configura por defecto en clusters *bare metal* es exponer el controlador *Ingress* como un servicio *NodePort*, tal y como se ha visto en la figura superior, aunque existen otras formas de configurarlo (entre ellas usar *MetalLB*). Si tienes interés, [aquí](#) tienes más información sobre este tópico





Ejercicio 5: Despliegue multi-aplicación

- Con el controlador *Ingress* instalado, desplegaremos ahora ambas aplicaciones
- Modifica en tu equipo el fichero *index.html* correspondiente a este ejercicio
 - **Debes sustituir "[alumno/a]" por tu nombre y apellido** 
- Crea el *ConfigMap* a partir del fichero *index.html* modificado
 - `kubectl create configmap apache-cfgmap --from-file=/vagrant/ej5/html/index.html`
- Despliega el *Deployment* y el *Service* correspondientes al servidor Apache:
 - `kubectl apply -f /vagrant/ej5/apache.yml`
- Repite los pasos previos para desplegar el servidor web Nginx
 - Comprueba que tienes modificado el *index.html* del ejercicio 3 (el que usa Nginx)
 - `kubectl create configmap nginx-cfgmap --from-file=/vagrant/ej3/html/index.html`
 - `kubectl apply -f /vagrant/ej5/nginx.yml`
- Obtén información de todos los recursos que hemos creado
 - `kubectl get all`



Ejercicio 5: Despliegue multi-aplicación



```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get all
```

NAME	READY	STATUS	RESTARTS	AGE
pod/apache-deployment-7c5b649b96-5vqd2	1/1	Running	0	30s
pod/apache-deployment-7c5b649b96-fc4ds	1/1	Running	0	30s
pod/apache-deployment-7c5b649b96-gskc7	1/1	Running	0	30s
pod/nginx-deployment-54c9699c4b-48n68	1/1	Running	0	30s
pod/nginx-deployment-54c9699c4b-8v6dq	1/1	Running	0	30s

Espera unos segundos y repite el comando hasta que veas el estado *Running* en todos los *pods*

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/apache-service	ClusterIP	10.98.148.113	<none>	9090/TCP	30s
service/kubernetes	ClusterIP	10.96.0.1	<none>	443/TCP	45h
service/nginx-service	ClusterIP	10.109.36.85	<none>	8080/TCP	30s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/apache-deployment	3/3	3	3	30s
deployment.apps/nginx-deployment	2/2	2	2	30s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/apache-deployment-7c5b649b96	3	3	3	30s
replicaset.apps/nginx-deployment-54c9699c4b	2	2	2	30s

```
vagrant@rre-aisi2223-k8s-master:~$
```

● Comprueba el acceso a ambas aplicaciones mediante *curl*

- *curl http://X.X.X.X:8080* ← Utiliza las *ClusterIP* correspondientes en tu caso en función de la salida obtenida en el comando previo (ver figura superior). Los puertos de acceso están definidos en los manifiestos correspondientes (curiosear los ficheros *nginx.yml* y *apache.yml* del ejercicio)
- *curl http://X.X.X.X:9090*



Ejercicio 5: Despliegue multi-aplicación



```
vagrant@rre-aisi2223-k8s-master:~$ curl 10.98.148.113:9090
```

```
<html>
<head>
  <meta charset= "utf-8">
  <title>GEI AISI: Test Page</title>
  <script type="text/javascript">
    function getURL() {
      document.write("URL: " + window.location.href);
    }
    function getTime() {
      document.getElementById("current_date").innerHTML = Date();
    }
  </script>
</head>
<body>
  <div style="width:600px;height:200px;border:2px solid #000;text-align:
  <strong><br>
  <u>GEI AISI: 2022/2023</u>
  <p><u>Apache Web Server (K8s cluster)</u></p>
  <p>Página web de Roberto Rey Expósito</p>

  <p><script>getURL();</script></p>
  <p><div id="current_date"><script>getTime();</script></p>
  </strong>
</div>
</body>
</html>
vagrant@rre-aisi2223-k8s-master:~$
```

```
vagrant@rre-aisi2223-k8s-master:~$ curl 10.109.36.85:8080
```

```
<html>
<head>
  <meta charset= "utf-8">
  <title>GEI AISI: Test Page</title>
  <script type="text/javascript">
    function getURL() {
      document.write("URL: " + window.location.href);
    }
    function getTime() {
      document.getElementById("current_date").innerHTML = Date();
    }
  </script>
</head>
<body>
  <div style="width:600px;height:200px;border:2px solid #000;text-align:
  <strong><br>
  <u>GEI AISI: 2022/2023</u>
  <p><u>Nginx Web Server (K8s cluster)</u></p>
  <p>Página web de Roberto Rey Expósito</p>


  <p><script>getURL();</script></p>
  <p><div id="current_date"><script>getTime();</script></p>
  </strong>
</div>
</body>
</html>
vagrant@rre-aisi2223-k8s-master:~$
```



Ejercicio 5: Despliegue multi-aplicación

- Para exponer el acceso al exterior de ambas aplicaciones a través del controlador *Ingress*, debemos crear un **recurso de tipo *Ingress*** que defina la reglas de enrutado que queremos aplicar:

- `kubectl apply -f /vagrant/ej5/ingress.yml`



Recuerda que el controlador y el recurso *Ingress* son cosas diferentes. Repasa la transparencia 30

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: aisi2223.k8s.es
    http:
      paths:
      - path: /apache
        pathType: Exact
        backend:
          service:
            name: apache-service
            port:
              number: 9090
      - path: /nginx
        pathType: Exact
        backend:
          service:
            name: nginx-service
            port:
              number: 8080
```

Lista de reglas con las que se "comparan" las peticiones entrantes. En este ejemplo, tenemos una única regla

Cada regla contiene: un *host* (p.e. *foo.bar.com*), y una lista de rutas (*paths*), cada una de las cuales tiene un *backend* asociado. Un *backend* es una combinación servicio:puerto al que el controlador *Ingress* enruta el tráfico entrante cuando el *host* y la ruta coinciden con el contenido de una petición. En este ejemplo tenemos dos rutas, una para cada servicio que queremos exponer

Una petición entrante a `http://aisi2223.k8s.es/apache` será dirigida al servicio Apache (puerto 9090)

Una petición entrante a `http://aisi2223.k8s.es/nginx` será dirigida al servicio Nginx (puerto 8080)



Ejercicio 5: Despliegue multi-aplicación

- Obtén información sobre el recurso *Ingress* creado previamente
 - *kubectl get ingress*
 - *kubectl describe ingress*

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get ingress
NAME      CLASS  HOSTS          ADDRESS  PORTS  AGE
ingress   nginx  aisi2223.k8s.es  80      8s
vagrant@rre-aisi2223-k8s-master:~$ kubectl describe ingress
Name:
Labels:      <none>
Namespace:   default
Address:
Ingress Class:  nginx
Default backend: <default>
Rules:
  Host            Path  Backends
  ----            -
  aisi2223.k8s.es  /apache  apache-service:9090 (10.10.1.15:80,10.10.1.150:80,10.10.1.17:80)
                  /nginx   nginx-service:8080 (10.10.1.149:80,10.10.1.16:80)
Annotations:  nginx.ingress.kubernetes.io/rewrite-target: /
Events:
  Type    Reason    Age   From                      Message
  ----    -
  Normal  Sync      7s    nginx-ingress-controller  Scheduled for sync
vagrant@rre-aisi2223-k8s-master:~$
```


¿Con qué se corresponde la lista de direcciones IP que se muestra para cada servicio? ¿Por qué un servicio tiene más IPs más que el otro? La ejecución de estos comandos te ayudará:



```
kubectl get pods -o wide
kubectl describe service nginx-service
kubectl describe service apache-service
```



Ejercicio 5: Despliegue multi-aplicación

- Para comprobar el acceso a los servicios desde el navegador de tu equipo, debemos habilitar la resolución del nombre *aisi2223.k8s.es* a la dirección IP del nodo *master* del clúster K8s
 - En sistemas Linux, modifica el fichero */etc/hosts* para añadir la siguiente línea:
 - 192.168.56.10 aisi2223.k8s.es
 - En sistemas Windows, mira este [enlace](#)
 - En sistemas macOS, mira este [enlace](#)
 - Antes de continuar, comprueba con el comando *ping* que la resolución de dicho nombre funciona desde tu equipo
- **Desde el navegador de tu equipo, accede a las siguientes URLs:**
 - <http://aisi2223.k8s.es:XX/nginx>  Dado que el servicio del controlador *Ingress* es de tipo *NodePort*, debemos acceder al puerto donde se expone dicho servicio. Debes usar el puerto correspondiente en tu caso (ver transparencia 71). Como se explica en esa misma transparencia, esta es la opción más simple y que se configura por defecto en clusters *bare metal*, pero en caso de utilizar un servicio gestionado de K8s en la nube se usaría un servicio de tipo *LoadBalancer* que evitaría hacer una petición a un puerto concreto
 - <http://aisi2223.k8s.es:XX/apache>



Ejercicio 5: Despliegue multi-aplicación



GEI AISI: Test Page x +

← → ↻ ⚠ No es seguro aisi2223.k8s.es:30544/apache

GEI AISI: 2022/2023

Apache Web Server (K8s cluster)

Página web de Roberto Rey Expósito

URL: <http://aisi2223.k8s.es:30544/apache>

Wed Feb 01 2023 10:55:06 GMT+0100 (hora estándar de Europa central)

GEI AISI: Test Page x +

← → ↻ ⚠ No es seguro aisi2223.k8s.es:30544/nginx

GEI AISI: 2022/2023

Nginx Web Server (K8s cluster)

Página web de Roberto Rey Expósito

URL: <http://aisi2223.k8s.es:30544/nginx>

Wed Feb 01 2023 10:54:44 GMT+0100 (hora estándar de Europa central)



Ejercicio 5: Despliegue multi-aplicación

- Para terminar el ejercicio, elimina todos los recursos:
 - `kubectl delete -f /vagrant/ej5/nginx.yml`
 - `kubectl delete -f /vagrant/ej5/apache.yml`
 - `kubectl delete configmap nginx-cfgmap`
 - `kubectl delete configmap apache-cfgmap`
 - `kubectl delete -f /vagrant/ej5/ingress.yml`
 - `kubectl delete -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.5.1/deploy/static/provider/baremetal/deploy.yaml`

Puede tardar un par de minutos. **Paciencia**



Para saber (mucho) más sobre K8s

- Documentación oficial de K8s: <https://kubernetes.io/es/docs/home>
- **Autoescalado horizontal y vertical de pods en función de la carga**
 - *Horizontal Pod Autoscaler* (HPA)
 - <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale>
 - *Vertical Pod Autoscaler* (VPA)
 - <https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler>
- **Despliegue de aplicaciones usando Helm Charts: <https://helm.sh>**
 - Helm se autodefine como un "gestor de paquetes" para K8s
 - Los Charts de Helm ayudan a definir, instalar y actualizar aplicaciones de K8s, siendo fáciles de crear, versionar, compartir y publicar
 - <https://refactorizando.com/como-crear-helm-chart-kubernetes>
 - **Helm se instaló en el nodo *master* del clúster K8s durante el despliegue con Vagrant y Ansible (ejecuta *helm version*)**
 - Las siguientes dos transparencias muestran un ejemplo muy simple de uso de Helm que puedes replicar si quieres



Helm Charts

81

```
vagrant@rre-aisi2223-k8s-master:~$ helm repo add bitnami https://charts.bitnami.com/bitnami
"bitnami" has been added to your repositories
vagrant@rre-aisi2223-k8s-master:~$ helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "bitnami" chart repository
Update Complete. *Happy Helming!*
vagrant@rre-aisi2223-k8s-master:~$ helm pull bitnami/nginx --untar
vagrant@rre-aisi2223-k8s-master:~$ ls nginx/
Chart.lock  Chart.yaml  README.md  charts  templates  values.schema.json  values.yaml
vagrant@rre-aisi2223-k8s-master:~$ helm install mynginx ./nginx --set service.type=NodePort
```

```
NAME: mynginx
LAST DEPLOYED: Wed Feb  1 10:02:39 2023
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
CHART NAME: nginx
CHART VERSION: 13.2.23
APP VERSION: 1.23.3
```

```
** Please be patient while the chart is being deployed **
NGINX can be accessed through the following DNS name from within your cluster:
```

```
mynginx.default.svc.cluster.local (port 80)
```

To access NGINX from outside the cluster, follow the steps below:

1. Get the NGINX URL by running these commands:

```
export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services mynginx)
export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
echo "http://${NODE_IP}:${NODE_PORT}"
```

```
vagrant@rre-aisi2223-k8s-master:~$ export NODE_PORT=$(kubectl get --namespace default -o jsonpath="{.spec.ports[0].nodePort}" services mynginx)
vagrant@rre-aisi2223-k8s-master:~$ export NODE_IP=$(kubectl get nodes --namespace default -o jsonpath="{.items[0].status.addresses[0].address}")
vagrant@rre-aisi2223-k8s-master:~$ echo "http://${NODE_IP}:${NODE_PORT}"
```

```
http://192.168.56.10:30759
```

```
vagrant@rre-aisi2223-k8s-master:~$ helm list
```

NAME	NAMESPACE	REVISION	UPDATED	STATUS	CHART	APP VERSION
mynginx	default	1	2023-02-01 10:02:39.121755411 +0000 UTC	deployed	nginx-13.2.23	1.23.3

← Añadimos un repositorio de Charts y lo actualizamos. Luego se descarga (pull) un Chart para desplegar Nginx. El Chart se compone de una serie de ficheros con una configuración por defecto que podemos modificar si se desea personalizar el despliegue

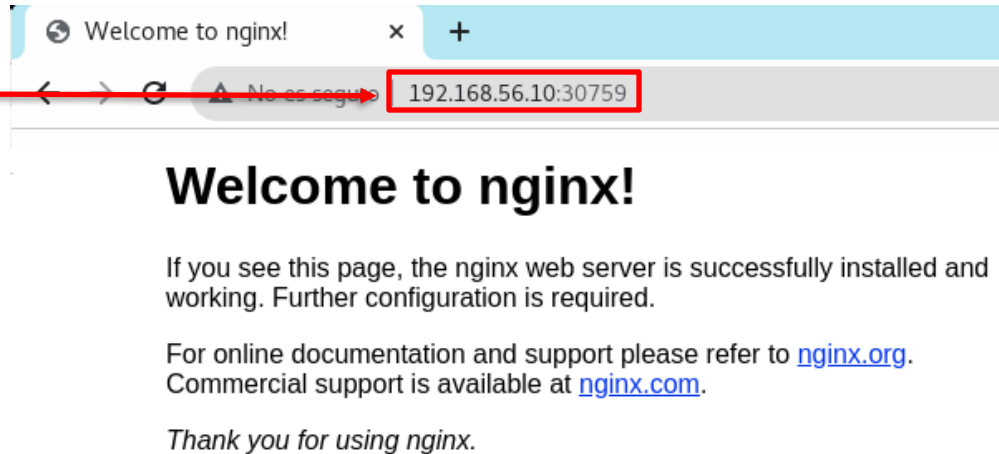
← Se instala (despliega) el Chart modificando el tipo de servicio a *NodePort* (por defecto, usa *LoadBalancer*). Alternativamente podríamos haber cambiado el tipo de servicio editando directamente el fichero *values.yaml* del Chart. La salida de este comando de instalación nos indica como obtener la IP y puerto de acceso



Helm Charts

82

Acceso al servidor Nginx desde el navegador del *host*. Vemos la página por defecto



Comprobamos los recursos que ha creado Helm. Terminamos desinstalando el Chart

```
vagrant@rre-aisi2223-k8s-master:~$ kubectl get all
NAME                                READY    STATUS    RESTARTS   AGE
pod/mynginx-5995d9d66c-4lm4n        1/1      Running   0           10m

NAME                                TYPE          CLUSTER-IP    EXTERNAL-IP    PORT(S)
service/kubernetes                  ClusterIP      10.96.0.1      <none>          443/TCP
service/mynginx                     NodePort       10.98.46.248   <none>          80:30759/TCP

NAME                                READY    UP-TO-DATE    AVAILABLE    AGE
deployment.apps/mynginx             1/1      1              1            10m

NAME                                DESIRED    CURRENT    READY    AGE
replicaset.apps/mynginx-5995d9d66c  1          1          1        10m
vagrant@rre-aisi2223-k8s-master:~$ helm uninstall mynginx
release "mynginx" uninstalled
vagrant@rre-aisi2223-k8s-master:~$ helm list
NAME      NAMESPACE    REVISION    UPDATED STATUS   CHART          APP VERSION
vagrant@rre-aisi2223-k8s-master:~$
```