Introducción a Kubernetes

Administración de Sistemas

Unai Lopez Novoa

unai.lopez@ehu.eus

eman ta zabal zazu

Universidad del País Vasco

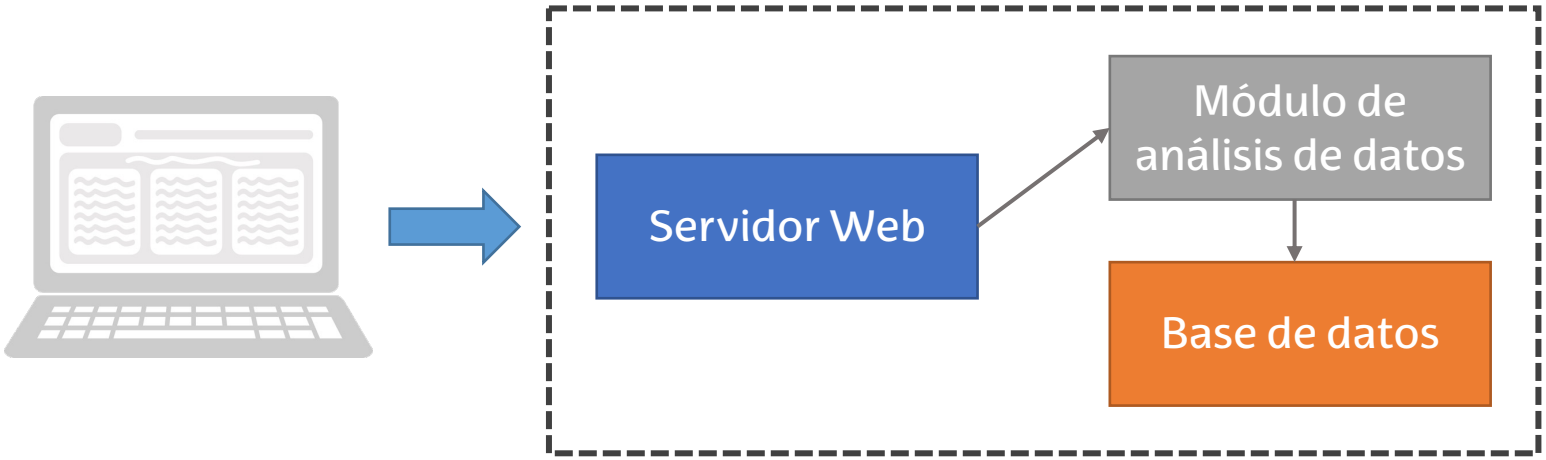
Euskal Herriko Unibertsitatea

Contenido

1. Introducción
    - Configuración en GCP
2. Objetos *Pod* y *Service*
3. Objetos *Deployment*

Introducción

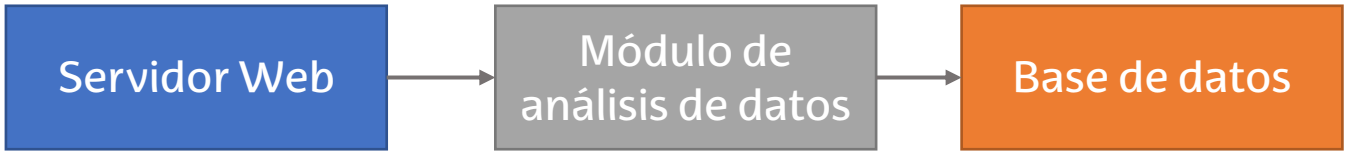
- Imaginad que tenemos una aplicación web compuesta por 3 contenedores:



  - Se utiliza para servir una web con los resultados de unos análisis calculados sobre la base de datos

Introducción

- Imaginad que cada uno de los contenedores se comporta de la siguiente forma:

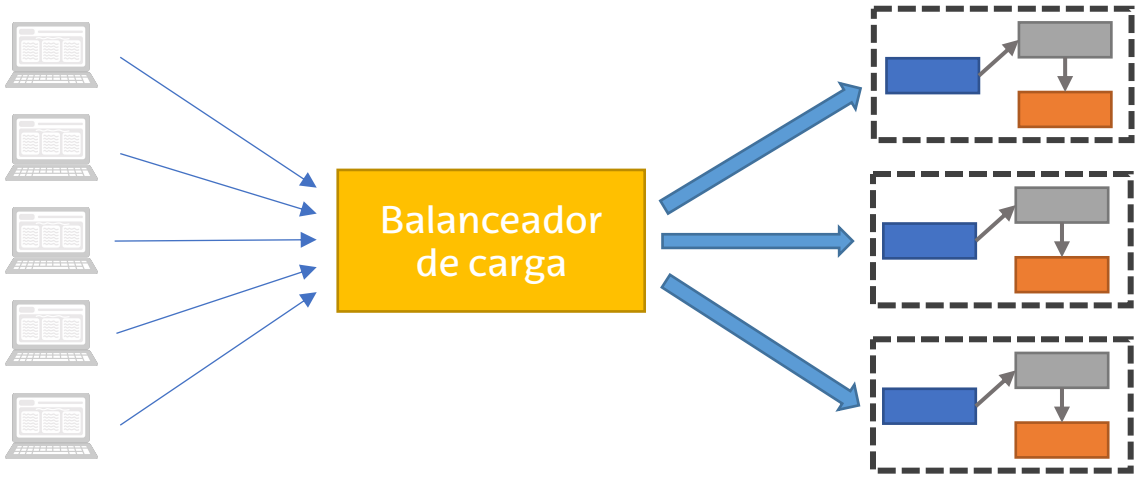


Tarea:	Servir página Web	Calcular análisis	Servir datos
Tiempo:	0,1 seg.	3 seg.	0,05 seg.

- Si nuestra aplicación empezase a recibir muchas peticiones, el módulo de análisis de datos sería un cuello de botella.

Introducción

- Para no crear grandes tiempos de espera, una solución posible es crear varias instancias de la aplicación:



- Sin embargo, en este enfoque:
  - Múltiples copias del servidor web y BBDD que no necesitamos
  - Sólo necesitaríamos más instancias del módulo de análisis

Introducción

- Una solución mejor sería crear múltiples instancias del módulo de análisis de datos exclusivamente.

- Esto es factible con Kubernetes

Kubernetes

- Plataforma para automatizar la implementación, escalado y administración de aplicaciones en contenedores.
  - Web oficial: https://kubernetes.io/
- Es código abierto
  - Desarrollado por Google
- Su nombre proviene del griego y significa *timonel* o *piloto*
  - En griego, se escribe κυβερνήτης
  - En ocasiones se abrevia como *k8s*

Alternativa: Docker Swarm

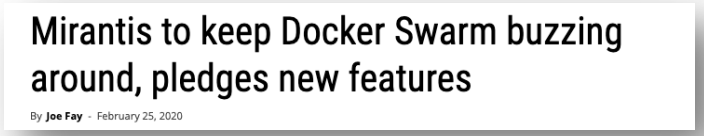
- Es una herramienta de orquestación integrada en Docker para gestionar clusters de contenedores.
  - Web: https://docs.docker.com/engine/swarm/
- Provee características gestionar la escalabilidad, la seguridad y la conectividad en red de un cluster.
- Alternativa completa a Kubernetes.
  - Más sencillo de utilizar y configurar que Kubernetes.

Alternativa: Docker Swarm

- Su futuro es incierto...

https://boxboat.com/2019/12/10/migrate-docker-swarm-to-kubernetes/

- Fecha: 10 de diciembre de 2019



https://devclass.com/2020/02/25/mirantis-to-keep-docker-swarm-buzzing-around-pledges-new-features/

- Fecha: 25 de febrero de 2020

https://searchitoperations.techtarget.com/news/252489220/Mirantis-rethinks-Docker-Swarm-vs-Kubernetes

- Fecha: 17 de septiembre de 2020

https://www.optimum-web.com/is-docker-swarm-dead-is-anyone-using-swarm-in-production-in-2021-future-docker-swarm/

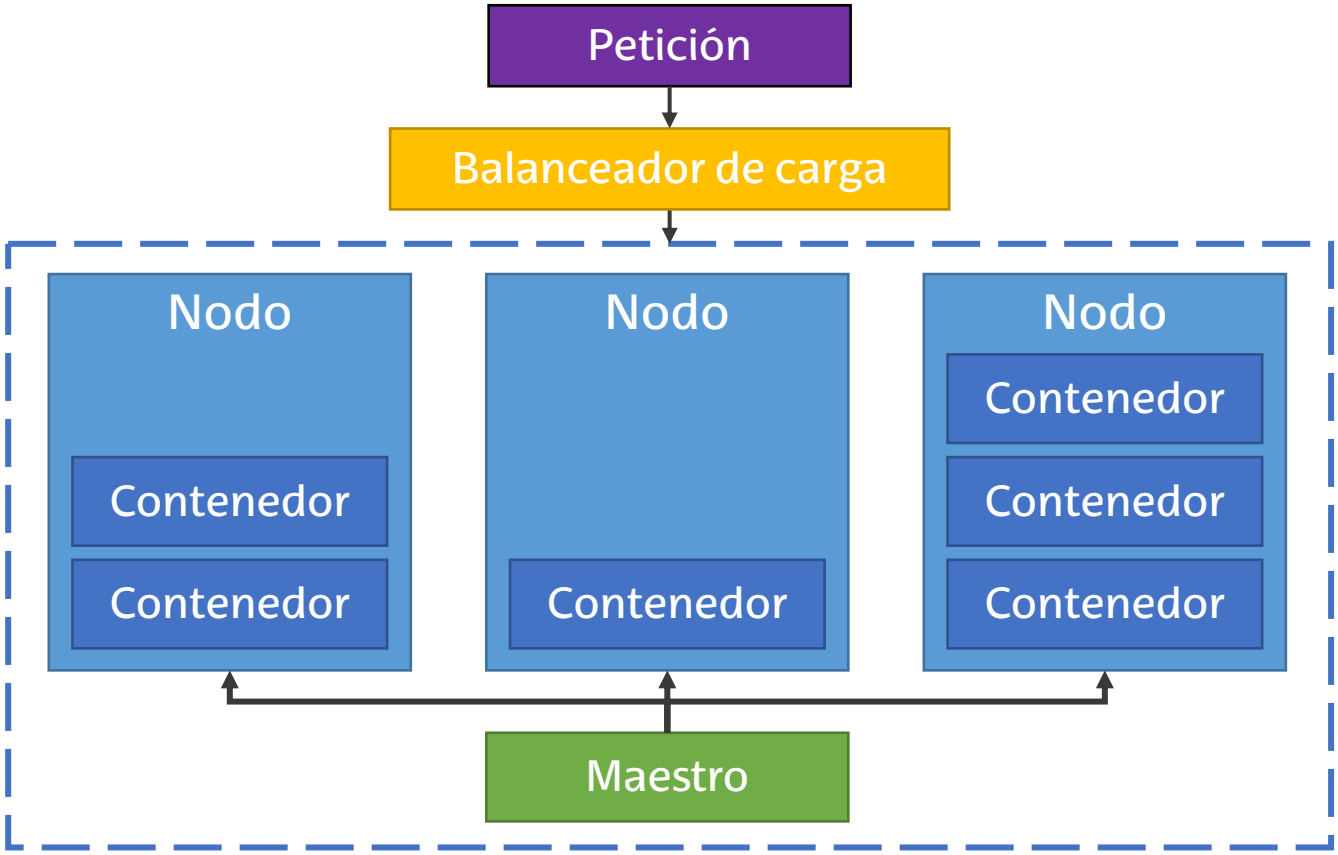
- Fecha: 10 de mayo de 2021

Alternativa: Docker Swarm

- Comparativa de popularidad con Google Trends
  - Fecha: 23 de octubre de 2023

Kubernetes

- Diagrama simple de un cluster k8s



Kubernetes

- Componentes de un cluster k8s
  - **Nodo**: Máquina física o virtual que ejecuta contenedores (o servicios).
    - Un nodo puede ejecutar diferentes contenedores simultáneamente.
  - **Maestro**: Servicio que controla los contenedores a ejecutar en cada nodo.
  - **Cluster** = Maestro + Nodos
- El balanceador de carga puede ser una pieza aparte de Kubernetes

Configuración

Trabajando con Kubernetes:

- En entornos locales:
  - Minikube, microk8s, kind, ...
- En entornos de producción:
  - Soluciones gestionadas, p.e.:
    - Amazon Elastic container service for Kubernetes (EKS)
    - Google Kubernetes Engine en GCP
  - Configuración *ad hoc*

Configuración

Trabajando con Kubernetes:

- En cualquier entorno, el comando principal es:

```
kubectl
```

  - Se utiliza para gestionar el cluster:
    - Realizar cambios en la configuración.
    - Monitorizar el estado de los objetos.
    - …
    - *Ejemplo:* para mostrar información del cluster:

```
kubectl cluster-info
```

Configuración en GCP

- Dos tipos de cluster Kubernetes en GKE:
  - Estándar:
    - Configuración manual de todos los parámetros (nodos, replicación, ...).
    - Se paga por cada MV del cluster.
    - Más flexible.
  - Autopilot:
    - GKE controla los parámetros relacionados con nodos y escalabilidad.
    - Se paga por cada objeto Pod.
    - *En este tema usaremos este tipo.*
  - *Más información: https://cloud.google.com/kubernetes-engine/docs/concepts/autopilot-overview*

Ejercicio 1

- Crear un cluster Kubernetes tipo “Autopilot” en GCP.
  - Modo: cluster público
- Abrir Cloud Shell y mostrar el estado del cluster.
- Instalar las herramientas *gcloud* y *kubectl* en vuestro equipo para gestionar el cluster desde línea de comando:
  1) Tutorial *gcloud*: https://cloud.google.com/sdk/docs/quickstart
  2) Instalar *kubectl*:

```
gcloud components install kubectl
```

Configuración

- En el Tema 3 utilizamos Docker Compose para gestionar entornos de múltiples contenedores.

- En un fichero docker-compose.yml, cada entrada en "services":
  - Representa un contenedor que queremos ejecutar.
  - Puede hacer que Compose construya una imagen.
  - Define los mapeos de puertos.
  - ...

```
services:
  servidor-redis:
    image: 'redis'
  servidor-web:
    build: .
    ports:
      - "6060:1080"
```

Ejemplo docker-compose.yml

Configuración

- Comparativa entre Compose y Kubernetes:

Compose	Kubernetes
Puede construir imágenes.	Kubernetes espera que las imágenes estén creadas.
Configuración de todos los contenedores en un único fichero.	Un fichero de configuración por cada *objeto** a crear.
La configuración de red se describe con los contenedores.	Configuración manual de la red.

* Definición de objeto más adelante.

Configuración

- La forma más sencilla para lanzar 1 contenedor en Kubernetes:

1) Kubernetes espera que las imágenes estén creadas.
   - Utilizar Docker Hub como alojamiento.

2) Un fichero de configuración por cada objeto.
   - Crear 1 fichero para definir la ejecución del contenedor.

3) Configuración manual de la red.
   - Crear 1 fichero de configuración para la red.

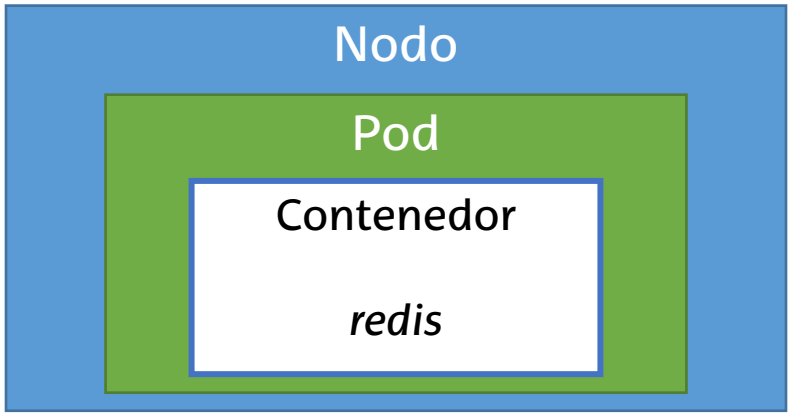
Configuración

- En Kubernetes, los objetos son entidades del cluster que sirven para diferentes propósitos.
  - Ejecutar/monitorizar un contenedor, configurar la red, ...
  - Un objeto no equivale a un contenedor.
- Un fichero de configuración YAML describe un objeto.
  - Se indica qué tipo de objeto con “kind:” (línea 2)
- Hay diferentes tipos de objetos.
  - P.e.: Pod, Service, ReplicaController, StatefulSet

Configuración: Pods

- Los objetos Pod se utilizan para ejecutar contenedores.
  - Se ejecutan dentro de un nodo.
  - Contienen un grupo de contenedores con un propósito concreto.

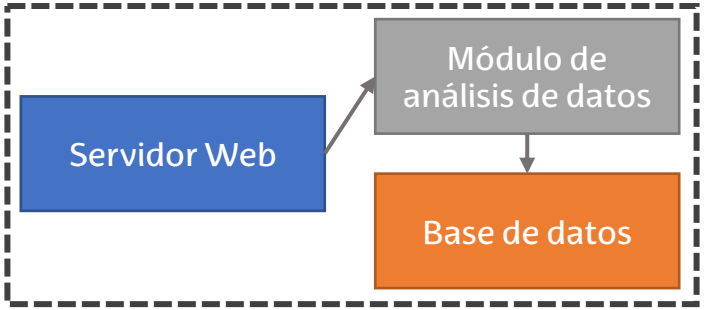
- Para lanzar un contenedor en Kubernetes es necesario utilizar al menos un Pod.



Configuración: Pods

- Los contenedores de un mismo Pod deben tener un mismo (o muy similar) propósito.
  - P.e. tienen que ejecutarse juntos para que la aplicación funcione.

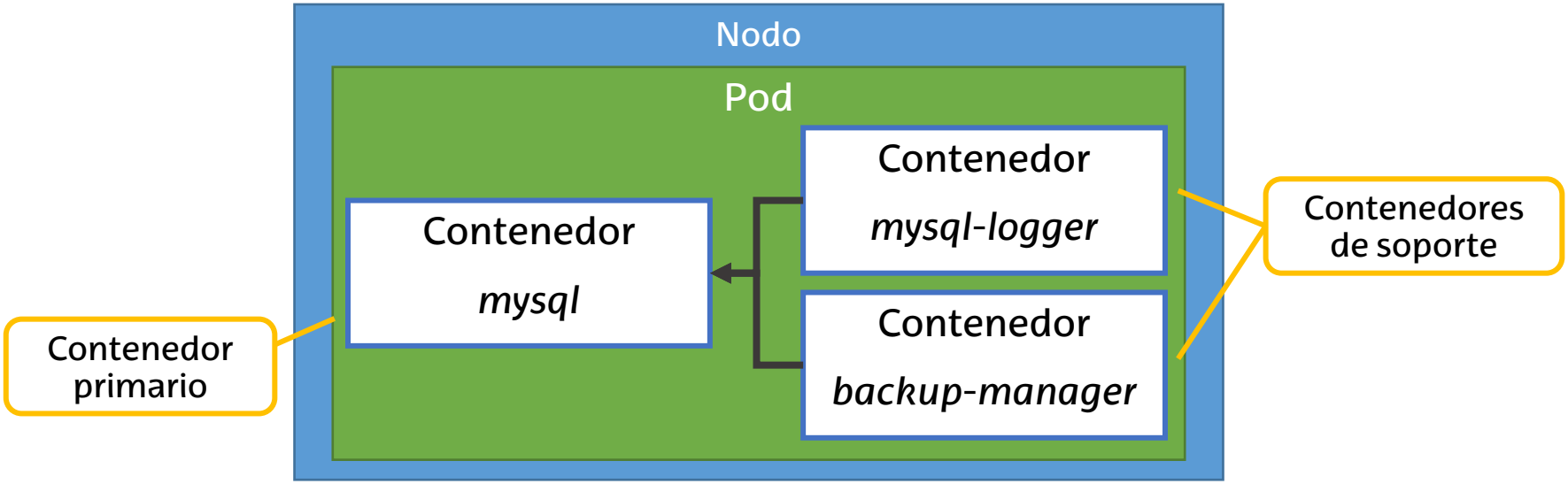
- Utilizando la aplicación inicial como ejemplo:



  - Aunque la BBDD se pare, el servidor Web sigue funcionando.
  - No sería conveniente tener estos 3 contenedores en el mismo Pod.

Configuración: Pods

- Los contenedores de un mismo Pod deben tener un mismo (o muy similar) propósito.
  - Ejemplo: Pod con 3 contenedores fuertemente relacionados:
    - Nota: *mysql-logger* y *backup-manager* son nombres inventados.



Configuración

- Ejemplo de fichero de configuración para 1 contenedor.
  - Fichero mi-pod.yml

```
apiVersion: v1
kind: Pod
metadata:
  name: mi-pod
  labels:
    component: web
spec:
  containers:
    - name: mi-contenedor
      image: ulopeznovoa/mi-linux
      ports:
        - containerPort: 3000
```

Configuración: Pods

- Configuración de un Pod con 1 contenedor en un fichero YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: mi-pod
  labels:
    component: web
spec:
  containers:
    - name: mi-contenedor
      image: ulopeznovoa/mi-linux
      ports:
        - containerPort: 3000
```

Configuración del Pod:
- name: Nombre arbitrario
- labels: Etiquetas identificativas

Configuración del contenedor a ejecutar dentro del Pod:
- name: Nombre arbitrario
- image: Repositorio de la imagen
- ports: Configuración de red

Indica que se debe exponer el puerto 3000

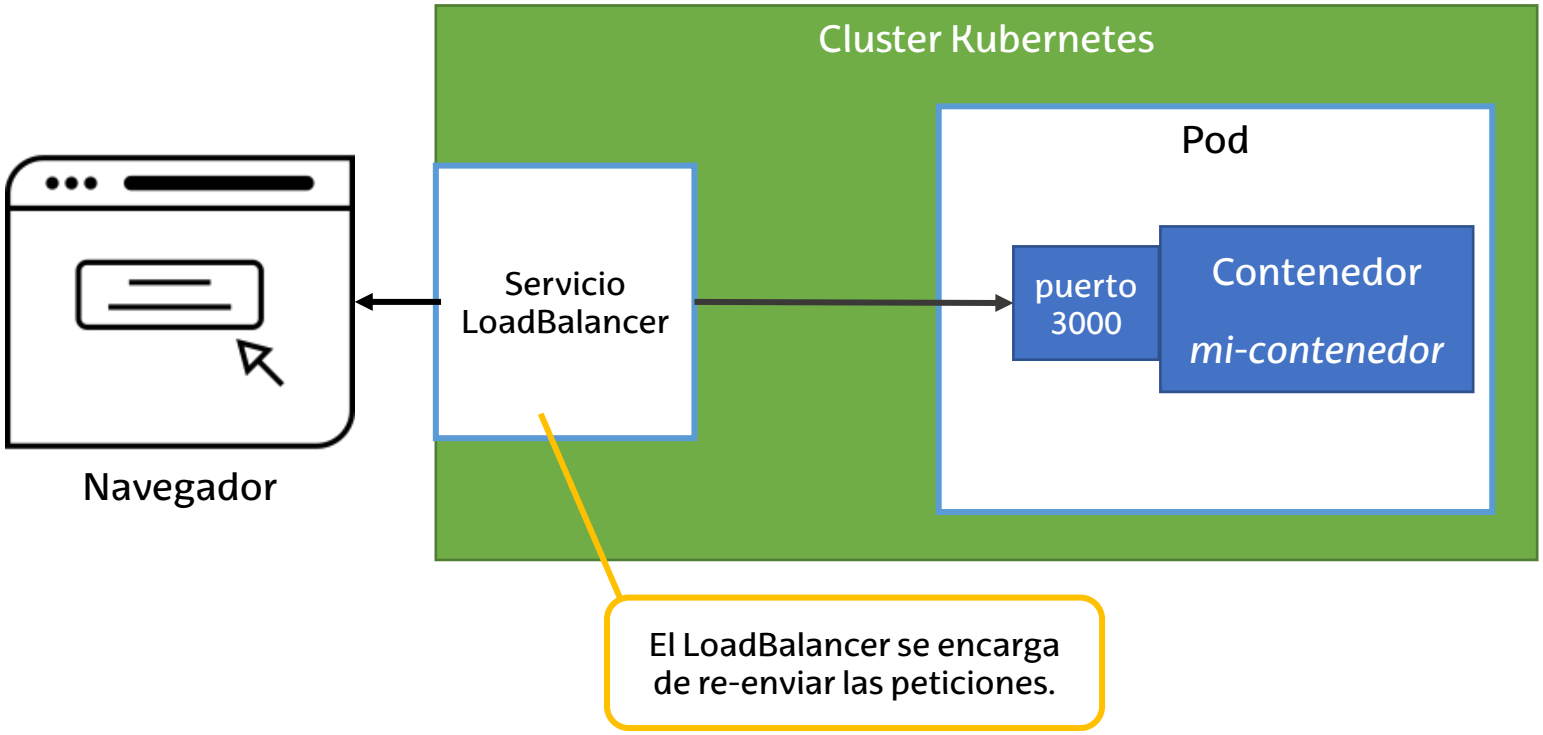
Configuración: Services

- Los objetos Service se utilizan para configurar elementos de red en el cluster Kubernetes.

- Hay 4 tipos de objetos Service:
  - ClusterIP
  - NodePort
  - LoadBalancer
    - Expone un puerto de un Pod al exterior del cluster.
    - Sólo se debe utilizar en entornos de desarrollo, no en producción.
  - Ingress

Configuración: Services

- El objeto Service se encarga de re-enviar las peticiones.
  - Representación usando el Pod del fichero YAML anterior:



Configuración: Services

- Ejemplo de configuración para un LoadBalancer:
  - Fichero mi-load-balancer.yml

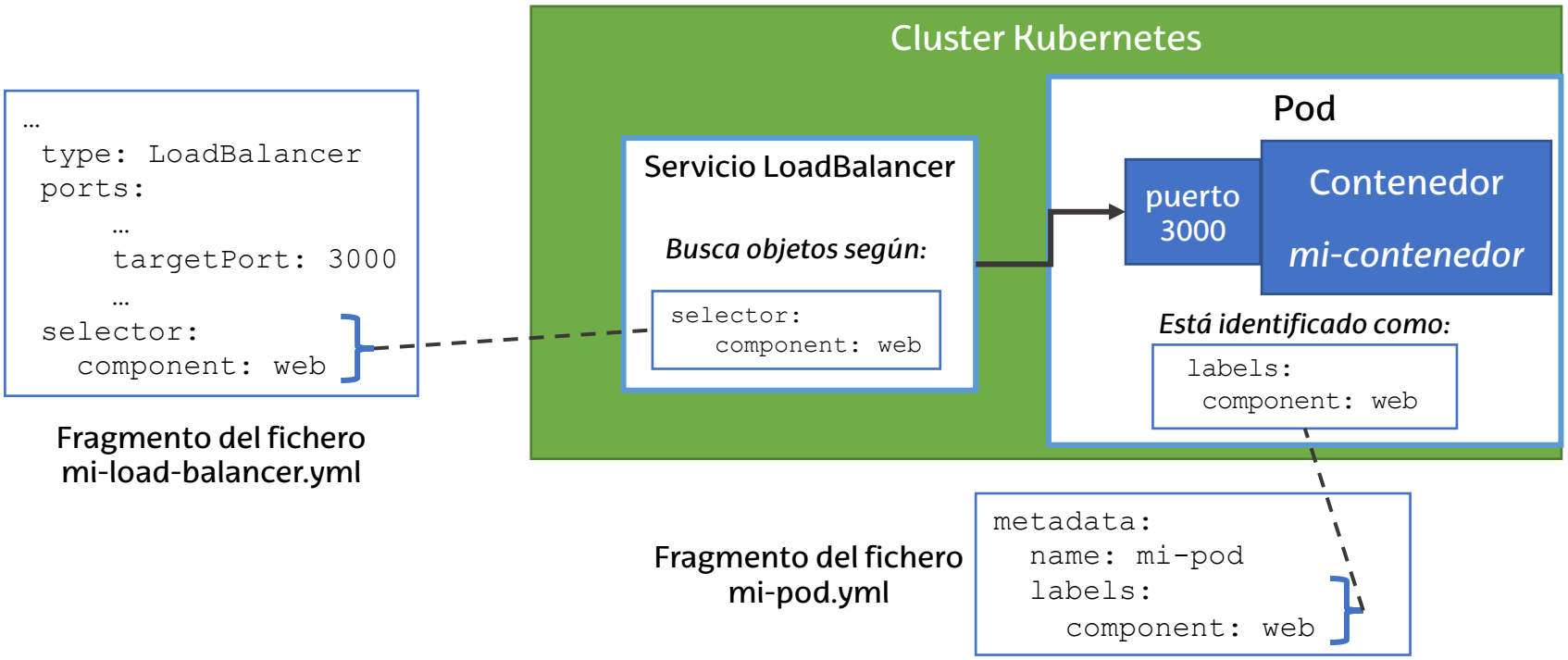
```
apiVersion: v1
kind: Service
metadata:
  name: mi-load-balancer
  labels:
    component: web
spec:
  type: LoadBalancer
  ports:
  - port: 80
    targetPort: 3000
  selector:
    component: web
```

Puerto a exponer al exterior del cluster

Puerto al que redirigir tráfico

Configuración: Services

- El objeto Service se encarga de re-enviar las peticiones.
  - Representación de los 2 ficheros YAML anteriores:



Configuración: Etiquetas

- En ambos ficheros, se puede utilizar cualquier par clave:valor para identificar los objetos.
  - No es necesario que la etiqueta siempre sea “component:”
  - Deben coincidir en ambos objetos.
    - En los ficheros de ejemplo, *label* (para el Pod) y *selector* (LoadBalancer)

- Algunas alternativas a “component: web”:
  - tier: frontend
  - componente: web
  - modulo: servidor
  - ...

Configuración: Resumen

- Un nodo ejecuta Pods.
  - En GKE, un nodo es una máquina virtual.
- Un Pod ejecuta uno o varios contendores.
  - Si son varios, deberían estar fuertemente relacionados entre sí.
- Los objetos Service se utilizan para configurar la red.
  - El tipo LoadBalancer permite exponer puertos de un Pod al exterior del cluster.

Comandos

- Cargar un fichero de configuración en Kubernetes:

```
kubectl apply -f <fichero>
```

  - donde:
    - apply Cambiar la configuración actual del cluster
    - -f Especifica que se proporciona un fichero

  - Se puede cargar cada fichero individualmente.
    - Para crear un Pod con la configuración de la diapositiva 25:

```
kubectl apply -f mi-pod.yaml
```

Comandos

- Obtener el estado de los objetos en el cluster

```
kubectl get <objeto>
```

  - donde:
    - get Obtener información de un objeto
    - ‹objeto› Tipo de objeto

  - Ejemplo:
    - Obtener información de los Pods

```
kubectl get pods
```

    - Obtener información de los Services:

```
kubectl get services
```

Comandos

- Para eliminar un objeto del cluster:

```
kubectl delete -f <fichero-config>
```

  - donde:
    - ‹fichero-config› Es el fichero usado para crear/modificar el objeto
  - kubectl lee el fichero y elimina los objetos del tipo especificado en “kind:” y con el nombre definido en “name:” dentro de “metadata:”.
  - Ejemplo: para eliminar un Pod creado anteriormente:
    - Si el fichero utilizado para crearlo ha sido mi-pod.yaml.

```
kubectl delete -f mi-pod.yaml
```

Ejercicio 2

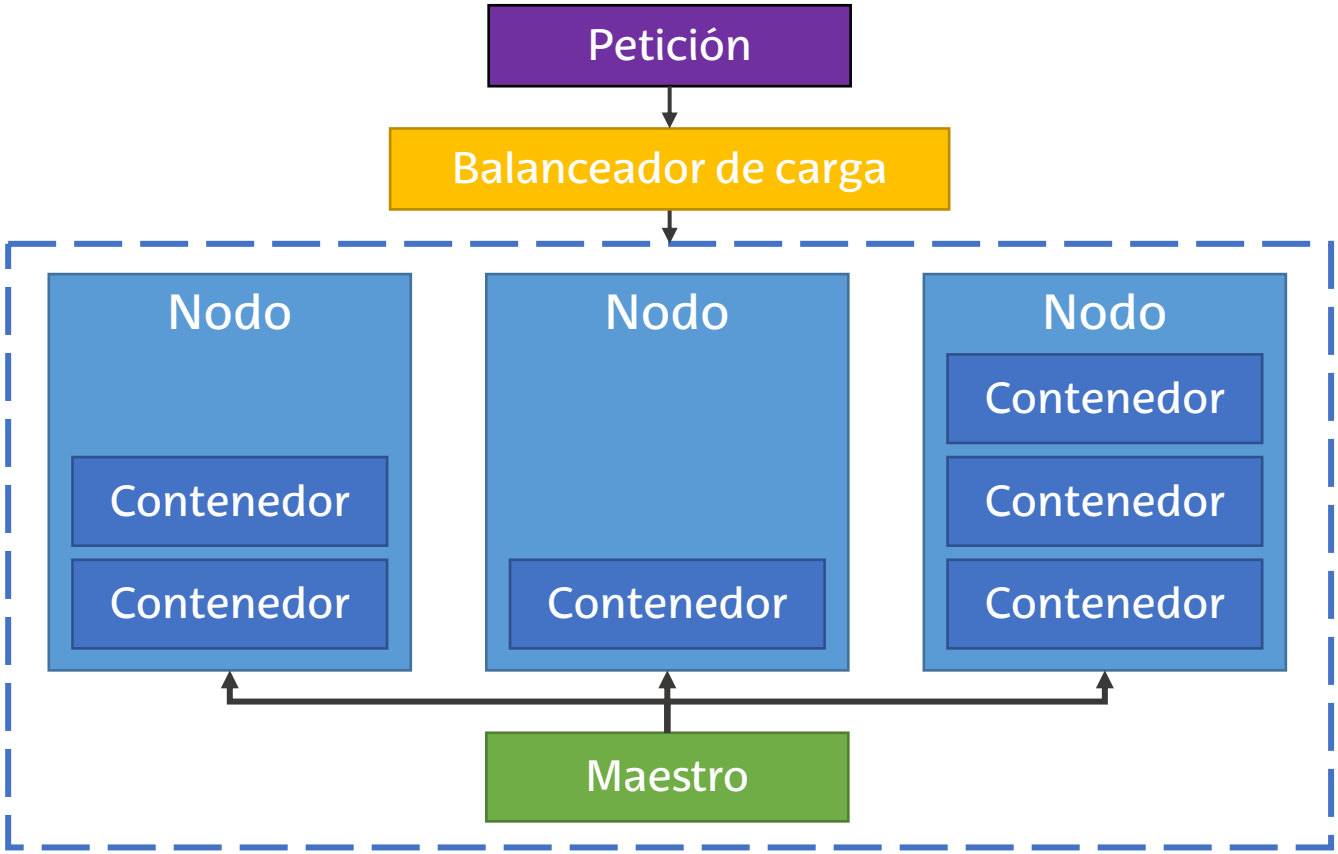
- Configurar el cluster Kubernetes con:
  - Un Pod que:
    - Ejecute la imagen “ulopeznovoa/simple-web-80”.
      - Contiene un servidor Web que muestra un HTML estático.
      - Escucha peticiones en el puerto 80.
    - Su etiqueta sea “modulo: servidor-web”
  - Un LoadBalancer que exponga el puerto 80 del Pod al exterior.

- Verificar que el en contenedor está en ejecución y es accesible:
  - Abrir http://‹IP-externa-cluster› en un navegador.

- Eliminar los objetos Pod y LoadBalancer recién creados.

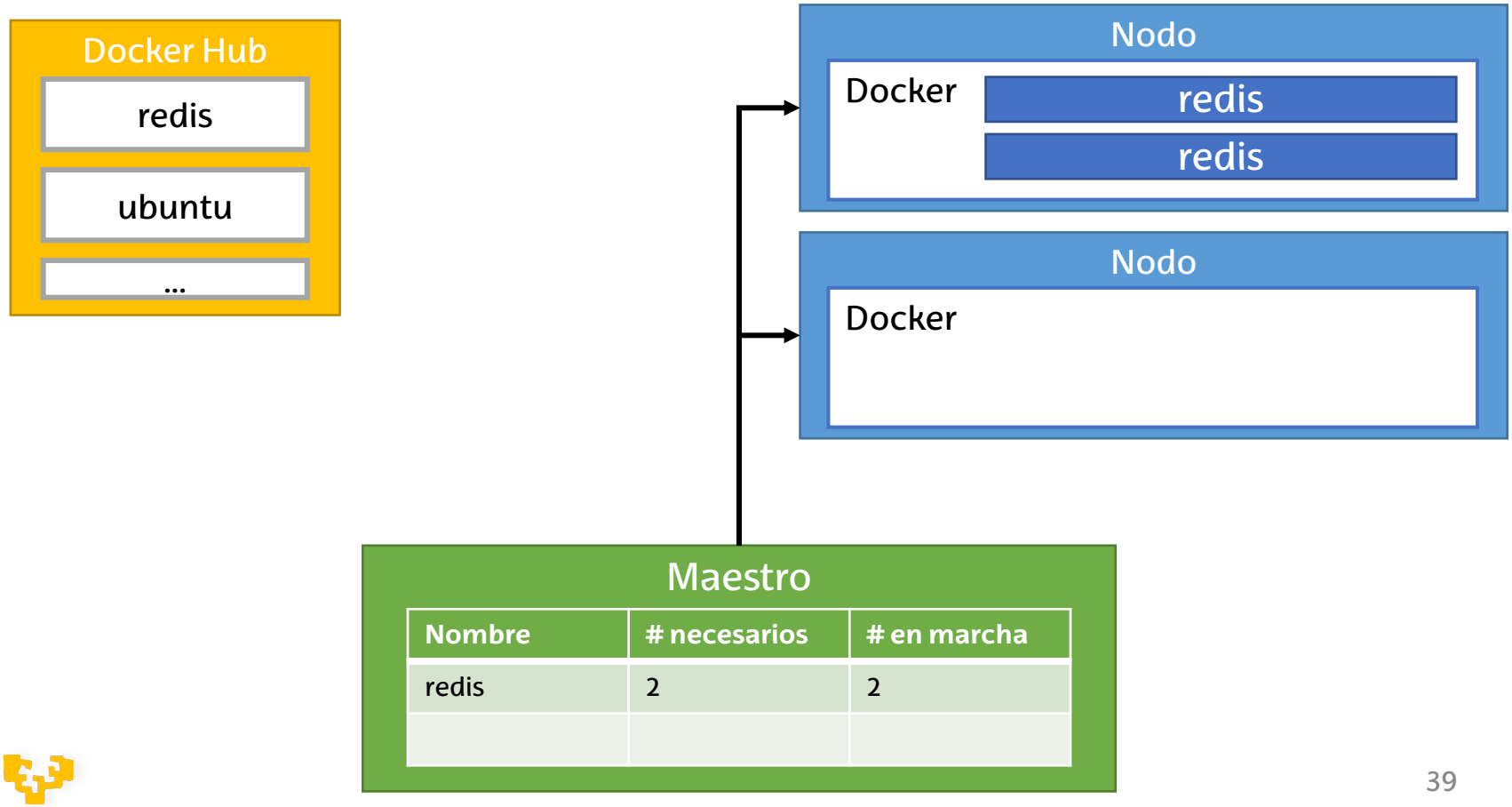
Kubernetes

- Diagrama simple de un cluster k8s



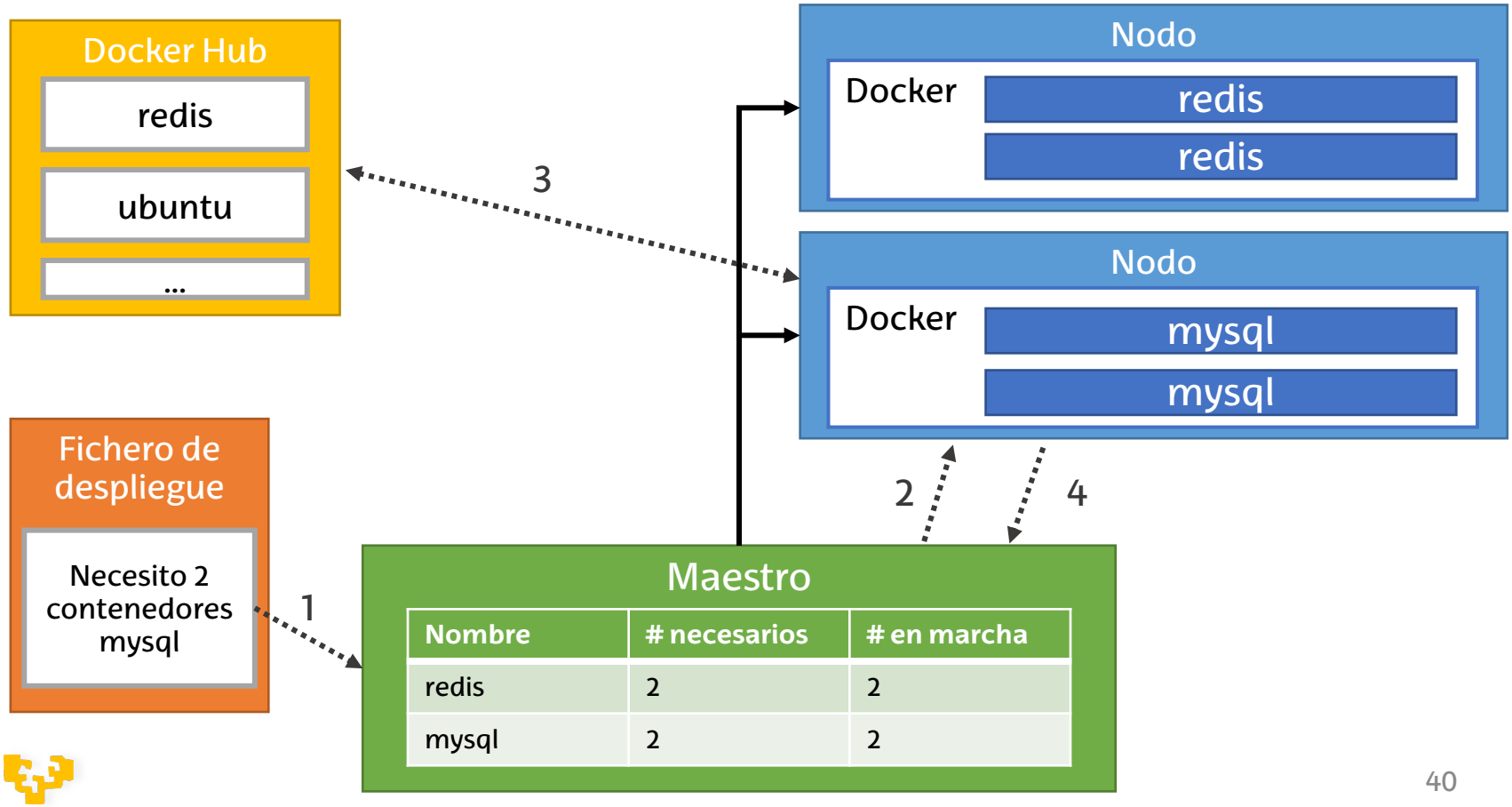
Kubernetes

- Componentes en un cluster k8s:



Kubernetes

- Diagrama de un cambio de configuración en el cluster:



Kubernetes

- Fases de un cambio de configuración.
  - Siguiendo el diagrama anterior.

1) Se recibe un fichero con el nuevo estado deseado
   - El maestro actualiza el estado del cluster: faltan contenedores

2) El maestro despliega los Pods necesarios en los nodos

3) Los nodos descargan la imagen de DockerHub

4) Los nodos envían su estado al maestro
   - El maestro actualiza el estado del cluster

Despliegues

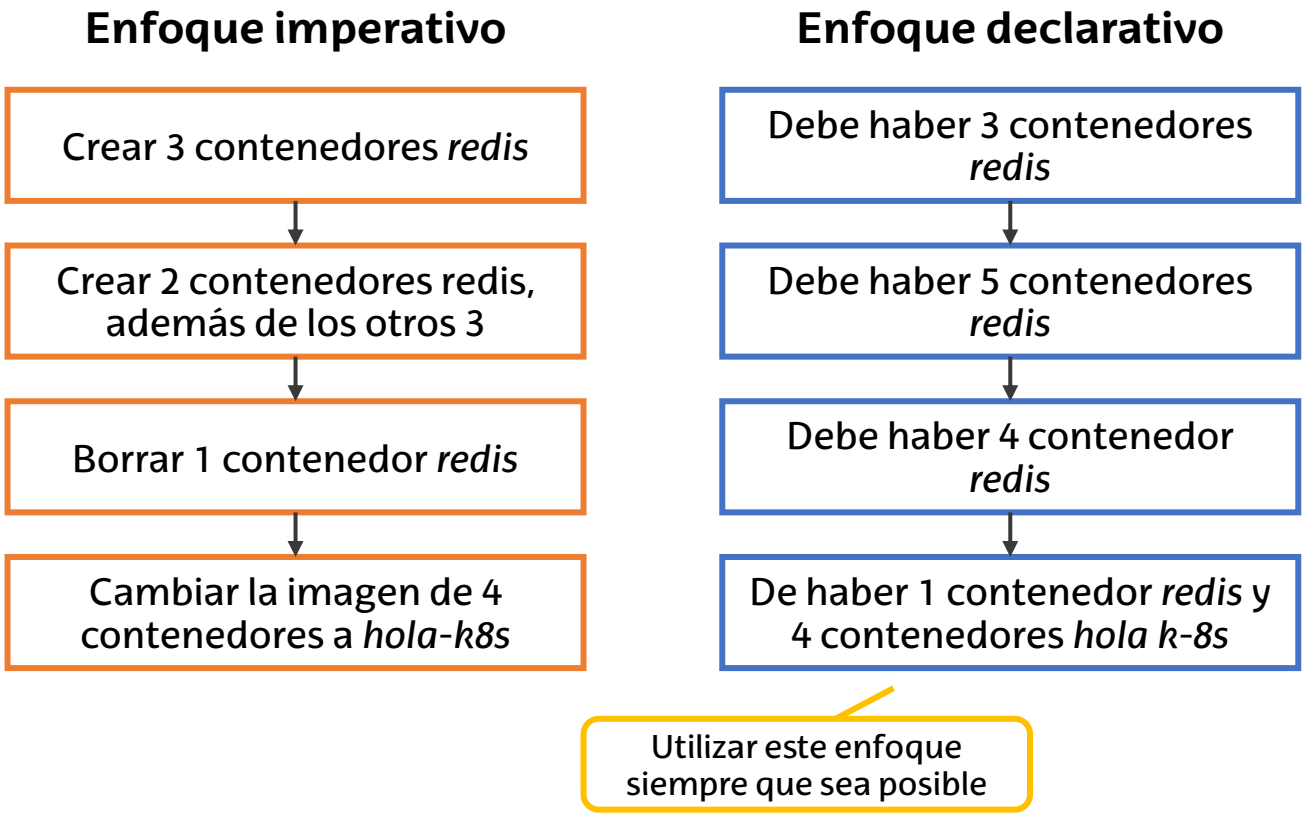
- Hay 2 formas de interactuar con k8s:

- Despliegues imperativos
  - Describir los pasos necesarios para llegar a un estado
  - P.e.:
    - Crear N contenedores, después crear una red entre ellos, ...

- Despliegues declarativos
  - Describir el estado deseado
  - P.e.:
    - Necesito N contenedores y una red entre ellos.

Despliegues

- Existen comandos para gestionar k8s de ambas formas.
  - Ejemplos:



Despliegues declarativos

- Tarea de ejemplo:
  - Actualizar la imagen en ejecución en un Pod

- Pasos:
  1) Modificar el fichero de configuración del objeto
     - Fichero mi-pod.yml

```
apiVersion: v1
kind: Pod
…
spec:
  containers:
    - name: client
      image: ulopeznovoa/hola-k8s
      ports:
        - containerPort: 1080
```

```
apiVersion: v1
kind: Pod
…
spec:
  containers:
    - name: client
      image: redis
      ports:
        - containerPort: 1080
```

Despliegues declarativos

- Tarea de ejemplo:
  - Actualizar la imagen en ejecución en un Pod

- Pasos:
  2) Utilizar kubectl con el fichero actualizado:

```
kubectl apply -f mi-pod.yml
```

Despliegues declarativos

- Tarea de ejemplo:
  - Actualizar la imagen en ejecución en un Pod

- Pasos:
  3) (Opcional) Verificar el estado del cambio:

```
kubectl describe <tipo-objeto> <nombre-objeto>
```

  - donde:
    - ‹tipo-objeto› Tipo de objeto: pod, service, ...
    - ‹nombre-objeto› Nombre del objeto (opcional)

  - Ejemplo:

```
kubectl describe pod
```

Despliegues declarativos

- Sin embargo, hay cambios que no son posibles de manera declarativa.

- Ejemplo:
  1) Cambiar containerPort en el fichero anterior

```
...
  image: ulopeznovoa/hola-k8s
  ports:
    - containerPort: 1080
```

```
...
  image: ulopeznovoa/hola-k8s
  ports:
    - containerPort: 1081
```

  2) Actualizar estado con kubectl -- devuelve un error

```
unai@unai-server:~/k8s$ kubectl apply -f mi-pod.yml
The Pod "client-pod" is invalid: spec: Forbidden: pod updates may not change
fields other than `spec.containers[*].image`, `spec.initContainers[*].image`,
`spec.activeDeadlineSeconds` or `spec.tolerations` (only additions to existing
tolerations)
```

Despliegues declarativos

- El objeto *Pod* sólo permite cambiar la propiedad “image” cuando está en marcha.
  - No permite cambiar “containers”, “name” o “port”
- Necesitamos utilizar un objeto más flexible.
  - Utilizaremos objetos *Deployment*.

Objeto *deployment*

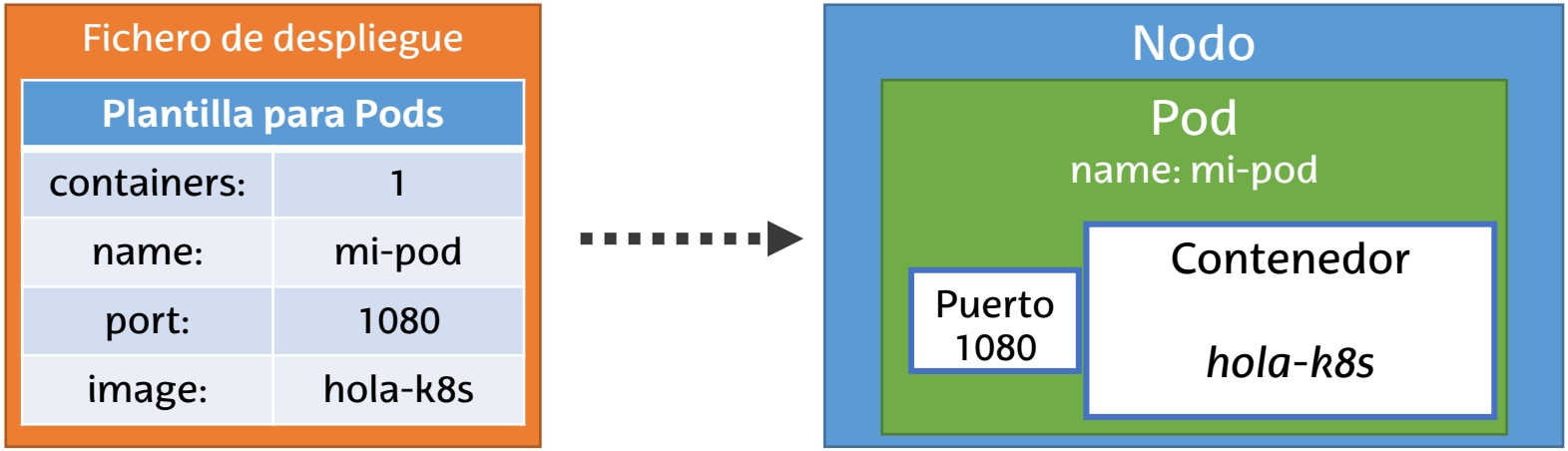
- Es un objeto de Kubernetes que mantiene un conjunto de Pods idénticos y asegura que:
  - Todos tienen la configuración correcta
  - Todos están en marcha

- Comparación:

Creación manual de Pod	Deployment
Ejecuta un conjunto de contenedores	Ejecuta un conjunto de Pods idénticos
Apropiado para usos puntuales en un entorno de desarrollo o pruebas	Monitoriza el estado de cada Pod y lo actualiza si es necesario
Raramente usado en producción	Apropiado para desarrollo y producción

Objeto *deployment*

- *Deployment* crea un conjunto de Pods y monitoriza su estado.



- Al modificar el fichero y enviarlo al maestro:
  - Si el cambio es posible, aplica el cambio.
  - Si no es posible, mata los Pods y los crea de nuevo.

Objeto *deployment*

- Ejemplo de fichero de configuración para 1 *Deployment*.
  - Fichero mi-deployment.yml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mi-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      component: web
  template:
    metadata:
      labels:
        component: web
    spec:
      containers:
      - name: mi-pod
        image: ulopeznovoa/hola-k8s
        ports:
        - containerPort: 1080
```

Objeto *deployment*

- Ejemplo de fichero de configuración para 1 *Deployment*.
  - client-deployment.yml
  - La sección "template" describe la configuración de los Pod que se van a crear.

```
…
  template:
    metadata:
      labels:
        component: web
    spec:
      containers:
        - name: mi-pod
          image: ulopeznovoa/hola-k8s
          ports:
            - containerPort: 1080
```

Todos los Pods gestionados por este Deployment tendrán:

- 1 label "component: web"
- 1 contenedor con:
  - Nombre: "mi-pod"
  - Imagen: "hola-k8s"
  - Puerto 1080 abierto

Objeto *deployment*

- Ejemplo de fichero de configuración para 1 *Deployment*.
  - client-deployment.yml

  - El resto de la sección "spec:" define:
    - El número de Pods (réplicas) a crear, siguiendo la plantilla en "template:"
    - Etiquetas de búsqueda. Para realizar operaciones de mantenimiento, *deployment* buscará Pods con las etiquetas aquí definidas.

```
…
spec:
  replicas: 1
  selector:
    matchLabels:
      component: web
  template:
    …
```

Se creará 1 Pod

Para el mantenimiento, buscará Pods con la etiqueta "component: web"

Comandos

- Obtener información de los objetos *deployment*:

```
kubectl get deployments
```

  - Por cada *deployment* activo, devuelve:
    - READY — Número de réplicas activas / Total solicitadas
    - UP-TO-DATE — Número de réplicas utilizando la última configuración
    - AVAILABLE — Número de replicas activas en el momento
    - AGE — Tiempo desde la creación del *deployment*

Ejercicio 3

- Modificar el ejercicio 2 para utilizar un objeto *Deployment* en lugar de un *Pod*.
  - Verificar que la aplicación funciona después del cambio.

- Modicar el *Deployment* para que sirva la imagen "httpd:alpine" en lugar de "simple-web-80".
  - Verificar que la aplicación funciona después del cambio.
    - Se debería mostrar el mensaje "It Works!" en el navegador.
    - Si se muestra la web anterior, probar a vaciar la caché del navegador.

Bibliografía

- Stephen Grider. “Docker and Kubernetes: The complete guide”, Udemy, 2020[1].
  - https://www.udemy.com/course/docker-and-kubernetes-the-complete-guide
- Kubernetes Docs, 2021[2].
  - https://kubernetes.io/docs

- Consultados en noviembre 2020[1] y noviembre 2021[2].