

Cloud Computing

Tema 4. Plataformas PaaS

© 2023 Javier Esparza Peidro - jesparza@dsic.upv.es

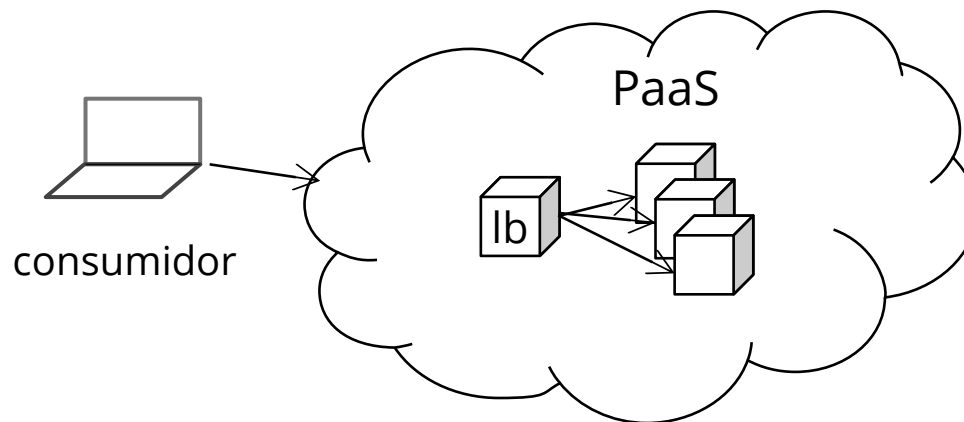
Contenido

- Introducción
- Kubernetes

Introducción

¿Qué es?

- El consumidor desarrolla y despliega aplicaciones
- El consumidor utiliza las **herramientas** que el proveedor facilita: frameworks, toolkits, etc.
- El entorno gestiona la infraestructura necesaria



Introducción

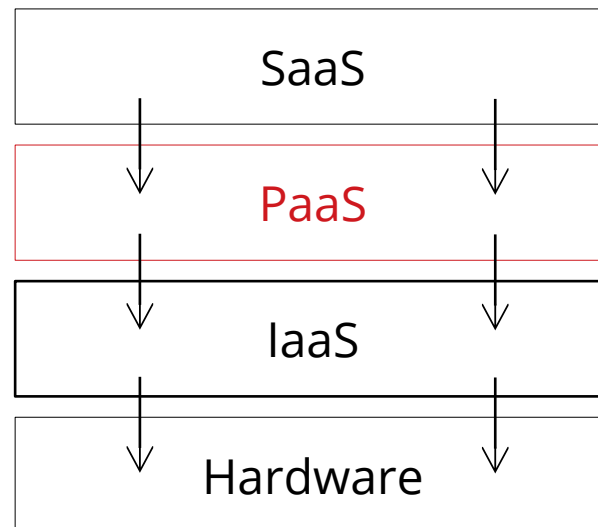
Herramientas

- Runtimes, APIs, frameworks, bases de datos, colas de mensajes, testing, despliegue, etc.
- El consumidor desarrolla su solución de acuerdo a las reglas/restricciones impuestas por el PaaS
- El PaaS ofrece garantías de rendimiento, seguridad, escalabilidad, resiliencia, etc.

Introducción

Estructura en capas

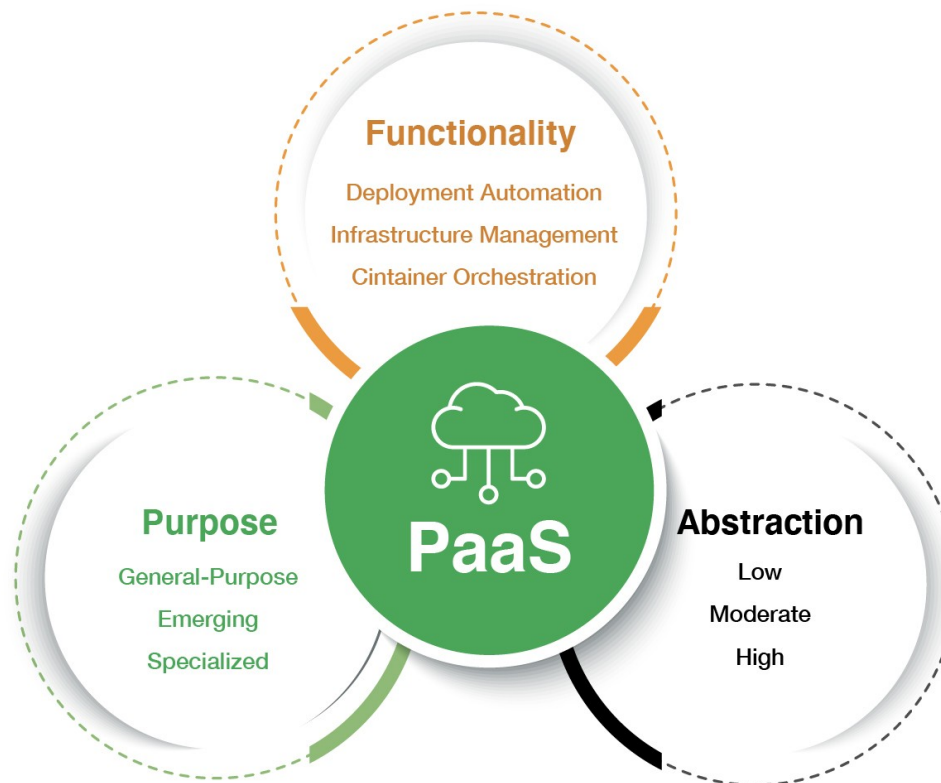
- Es común que un PaaS se construya por encima de un IaaS



Introducción

Tipos

- Tres criterios de clasificación



Introducción

Tipos > Según su propósito

- De propósito general
- Emergente
- Especializado

Introducción

Tipos > Según su propósito

- De propósito general
 - Para desarrollar cualquier tipo de solución
 - Azure, App Engine, OpenShift, etc.
- Emergente
- Especializado

Introducción

Tipos > Según su propósito

- De propósito general
- Emergente
 - Plataformas de experimentación, aportan nuevas aproximaciones o ideas: serverless, funciones (FaaS), machine learning (MLaaS), etc.
- Especializado

Introducción

Tipos > Según su propósito

- De propósito general
- Emergente
- Especializado
 - Muy específicas, generalmente giran alrededor de un producto consolidado, o un nicho de mercado.
 - Magento (ecommerce), Hadoop (Big data), Bases de datos (DBaaS), etc.

Introducción

Tipos > Según su abstracción

- Bajo nivel
- Nivel intermedio
- Alto nivel

Introducción

Tipos > Según su abstracción

- Bajo nivel
 - Más control sobre la configuración de la infraestructura subyacente, menos automatización
 - Por ejemplo, contenedores (CaaS)
- Nivel intermedio
- Alto nivel

Introducción

Tipos > Según su abstracción

- Bajo nivel
- Nivel intermedio
 - No se controla la infraestructura. Acceso a APIs, IDEs, frameworks, software stacks, etc.
 - Se automatiza el balanceo de carga, recuperación, escalado, etc.
- Alto nivel

Introducción

Tipos > Según su abstracción

- Bajo nivel
- Nivel intermedio
- Alto nivel
 - Abstraen cualquier proceso de configuración, incluso codificación.
 - El objetivo es acelerar el proceso de desarrollo y reducir costes.
 - Puede ser utilizada incluso por usuarios no técnicos

Introducción

Tipos > Según su funcionalidad

- Automatización del despliegue
- Gestión de la infraestructura
- Orquestación de contenedores

Introducción

Tipos > Según su funcionalidad

- Automatización del despliegue
 - Herramientas y servicios para automatizar el despliegue (one-click, git-push)
 - Entornos preconfigurados de programación, CD/CI y gestión de proyectos
- Gestión de la infraestructura
- Orquestación de contenedores

Introducción

Tipos > Según su funcionalidad

- Automatización del despliegue
- Gestión de la infraestructura
 - Configuración de la infraestructura, monitorización, seguridad, rendimiento
 - Actualizaciones automatizadas de servicios
- Orquestación de contenedores

Introducción

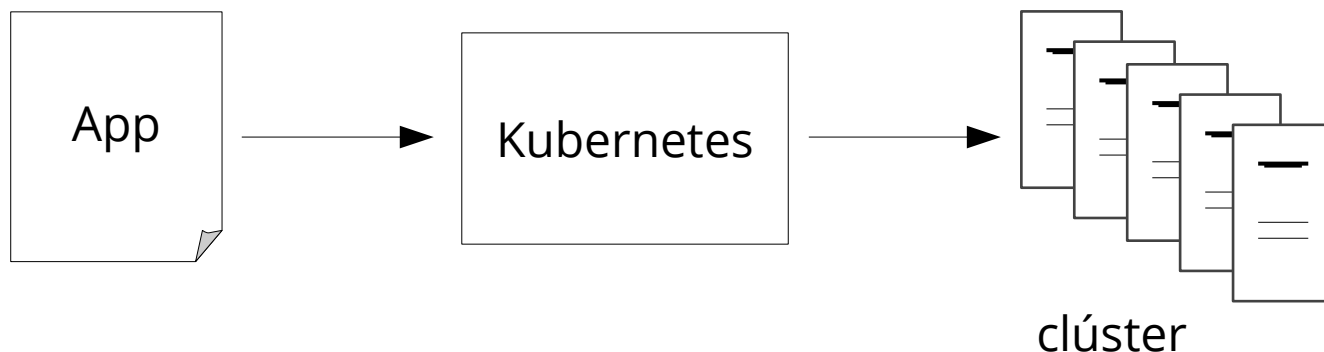
Tipos > Según su funcionalidad

- Automatización del despliegue
- Gestión de la infraestructura
- Orquestación de contenedores
 - Plataformas de gestión de contenedores

Kubernetes - k8s

¿Qué es?

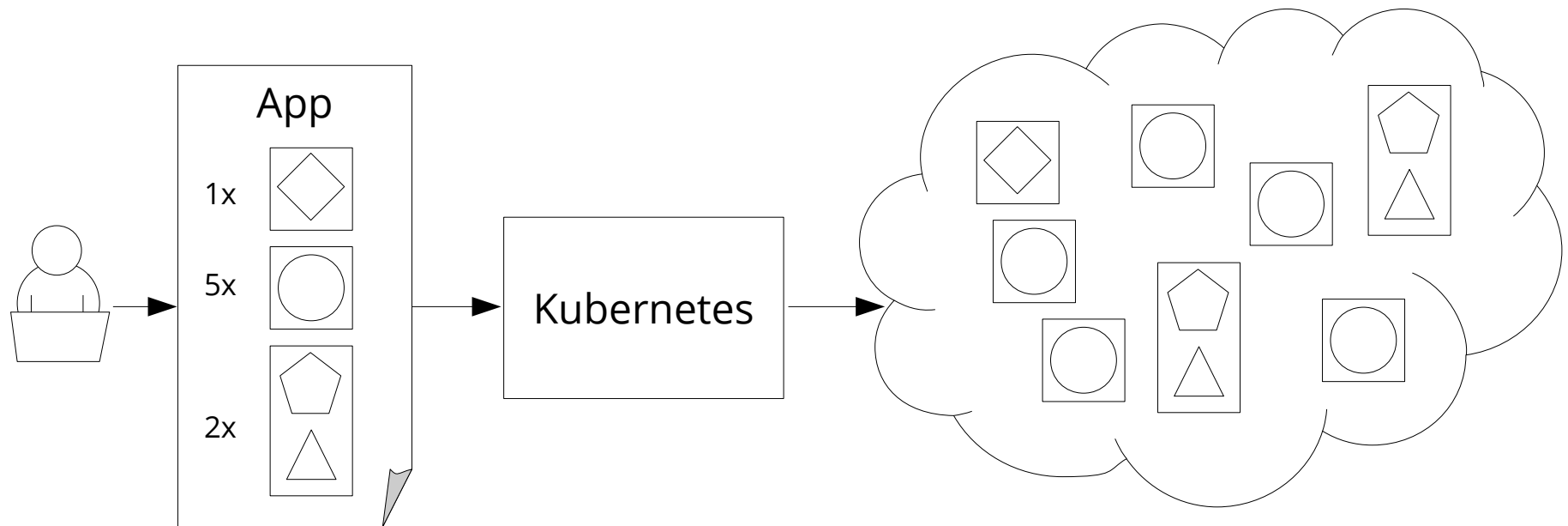
- <https://kubernetes.io/>
- Plataforma que permite desplegar y gestionar aplicaciones basadas en **contenedores**
- Se ejecuta en un clúster de máquinas y da la imagen de una única máquina muy potente



Kubernetes - k8s

¿Cómo funciona?

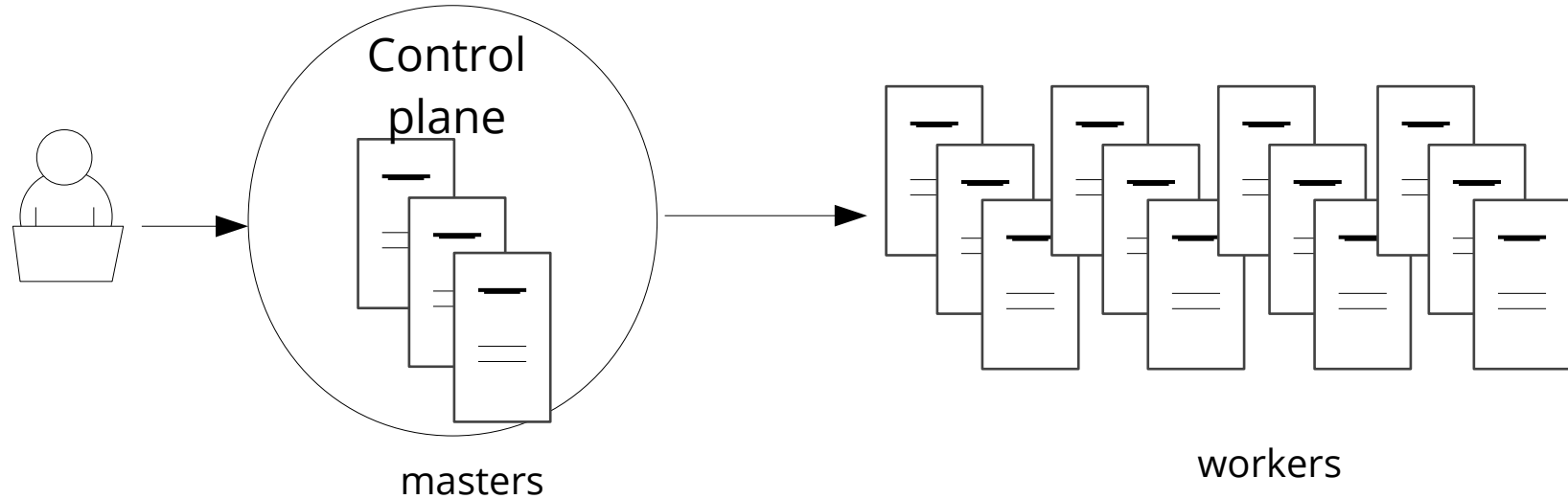
- Publica una API: el usuario define su aplicación
- Los nodos se añaden/eliminan dinámicamente
- Las aplicaciones no se resienten



Kubernetes - k8s

Arquitectura

- Nodos master vs nodos worker



Kubernetes - k8s

Arquitectura > Control plane

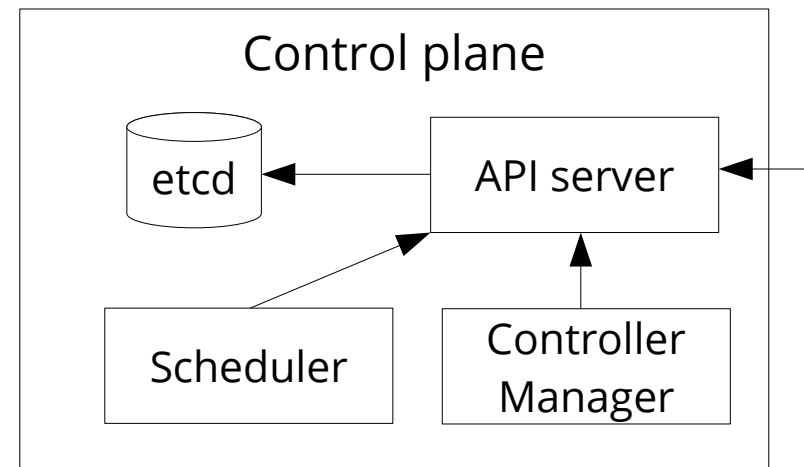
- Los nodos master constituyen en **control plane**
- El **control plane** gestiona y controla el clúster
- El usuario interacciona con el control plane a través de una **API**
- El control plane ejecuta las operaciones necesarias para que las aplicaciones alcancen el estado deseado

Kubernetes - k8s

Arquitectura > Control plane

Componentes (en un único master o varios)

- API server
- Scheduler
- Controller Manager
- etcd

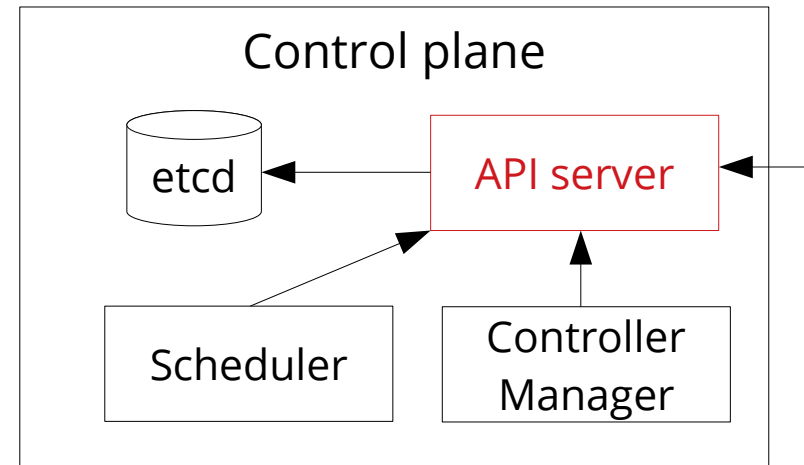


Kubernetes - k8s

Arquitectura > Control plane

Componentes (en un único master o varios)

- **API server**
 - kube-apiserver
 - Publica la API
 - Es el front-end del control plane
 - Se escala horizontalmente

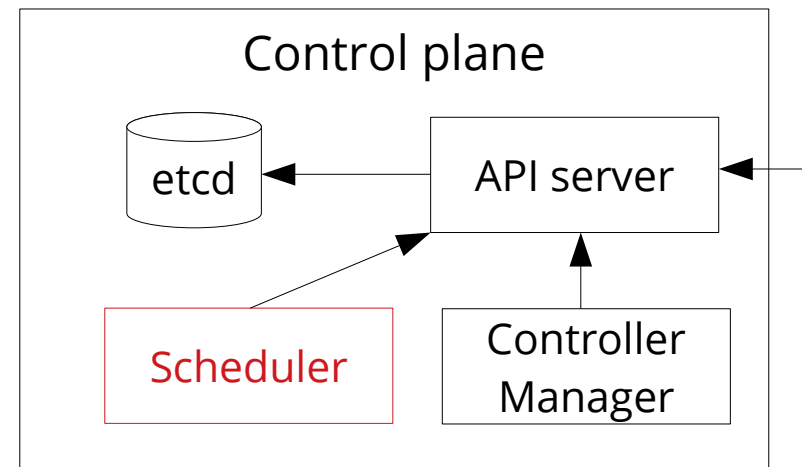


Kubernetes - k8s

Arquitectura > Control plane

Componentes (en un único master o varios)

- **Scheduler**
 - [kube-scheduler](#)
 - Determina dónde se ejecuta la carga
 - Tiene en cuenta muchos factores

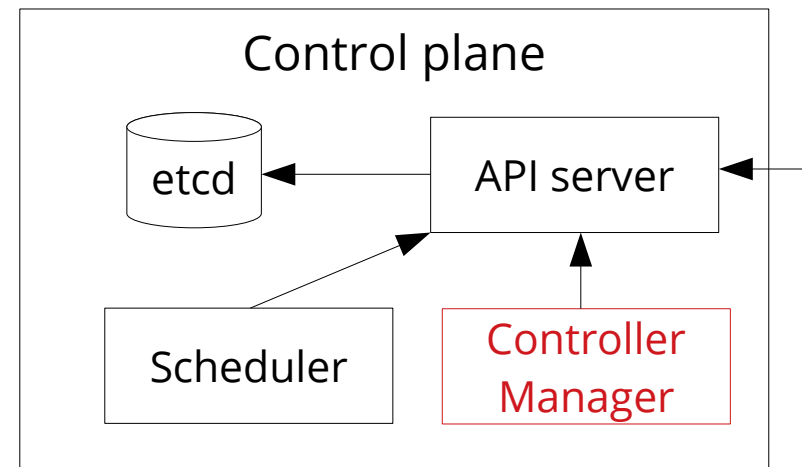


Kubernetes - k8s

Arquitectura > Control plane

Componentes (en un único master o varios)

- **Controller Manager**
 - [kube-controller-manager](#)
 - Contiene los [controllers](#)
 - Son **bucles de control** que controlan el estado del clúster

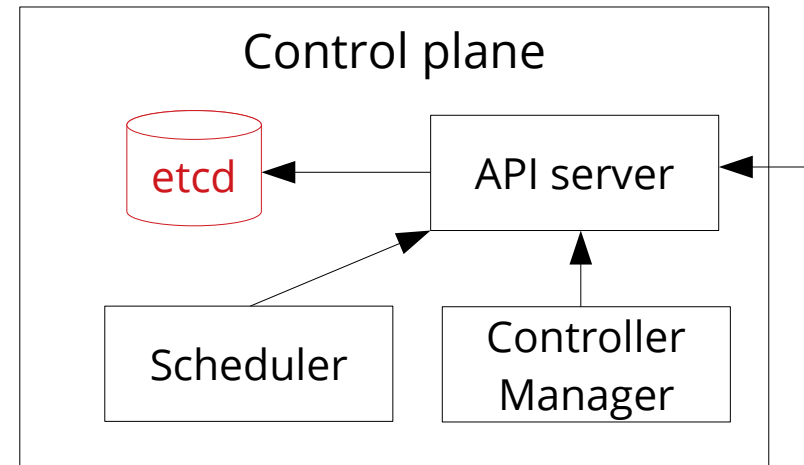


Kubernetes - k8s

Arquitectura > Control plane

Componentes (en un único master o varios)

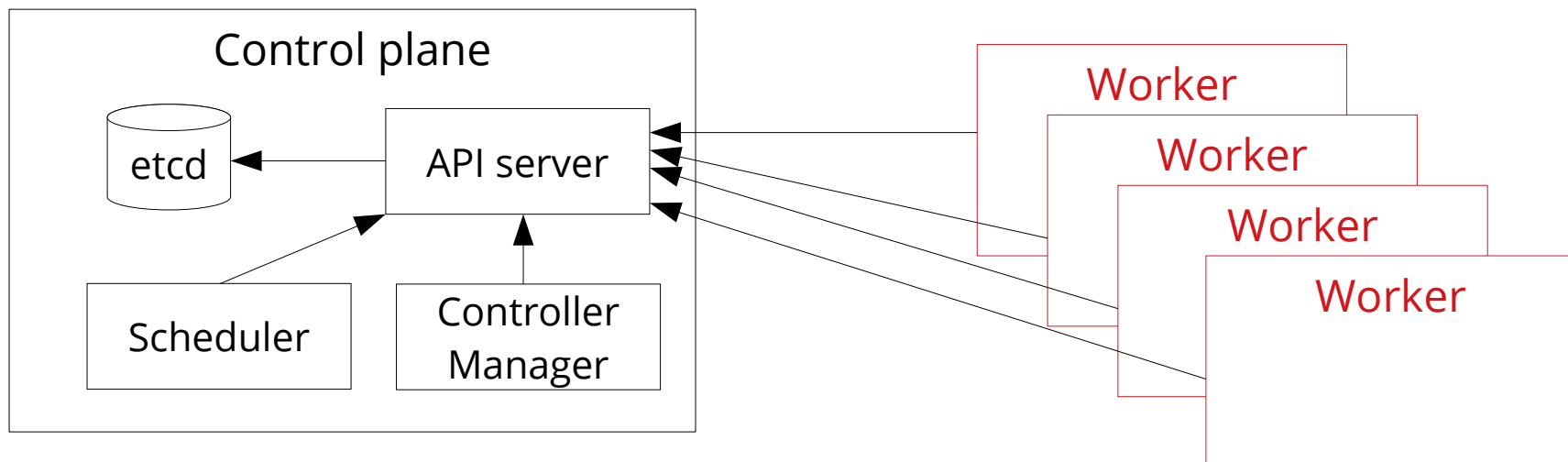
- etcd
 - Almacén clave-valor
 - Replicado, altamente disponible
 - Almacena toda la info del clúster



Kubernetes - k8s

Arquitectura > Workers

- Nodos (físicos, virtuales) que ejecutan y monitorizan los contenedores
- Controlados por el control plane

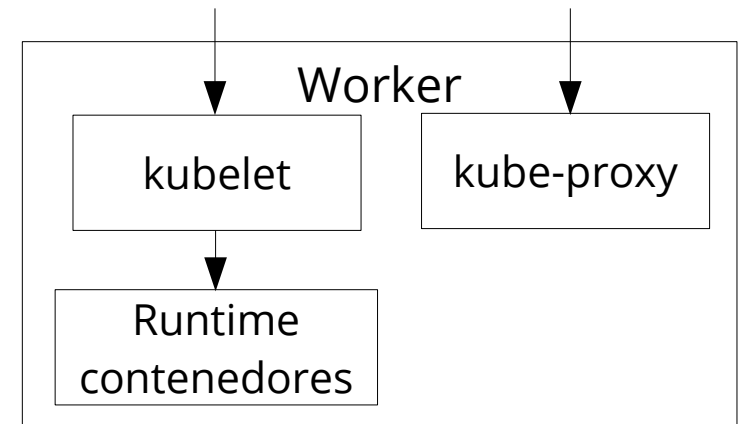


Kubernetes - k8s

Arquitectura > Workers

Componentes:

- Runtime de contenedores
- kubelet
- kube-proxy

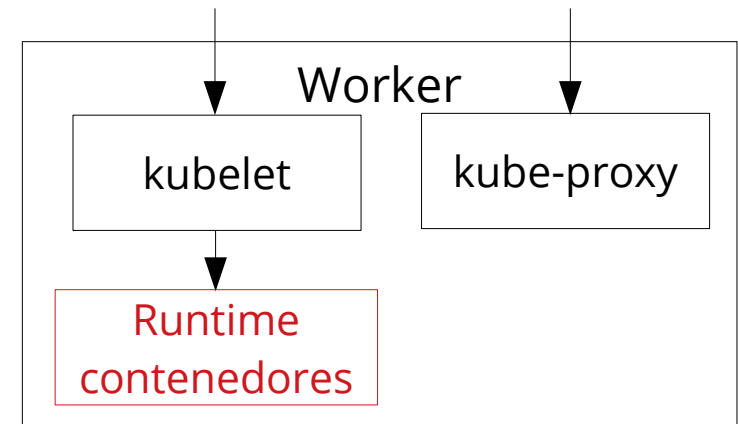


Kubernetes - k8s

Arquitectura > Workers

Componentes:

- Runtime de contenedores
 - Docker, containerd, rkt, etc.)
- Cualquier implementación que se integre con CRI (the Container Runtime Interface)

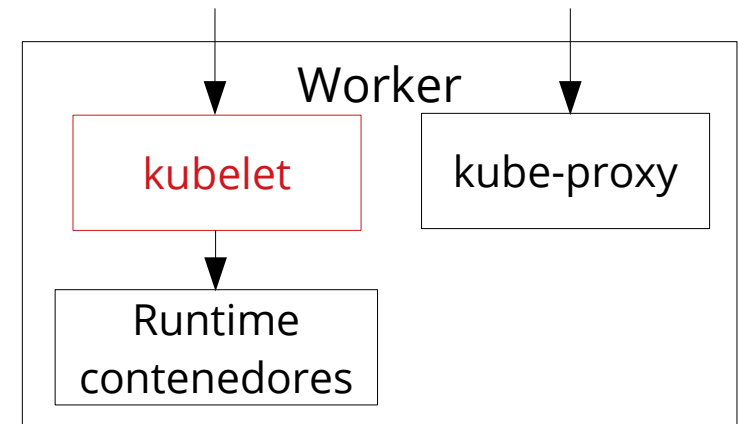


Kubernetes - k8s

Arquitectura > Workers

Componentes:

- kubelet
 - 1x agente en cada nodo
 - Se comunica con el API server
 - Garantiza que los contenedores funcionen correctamente

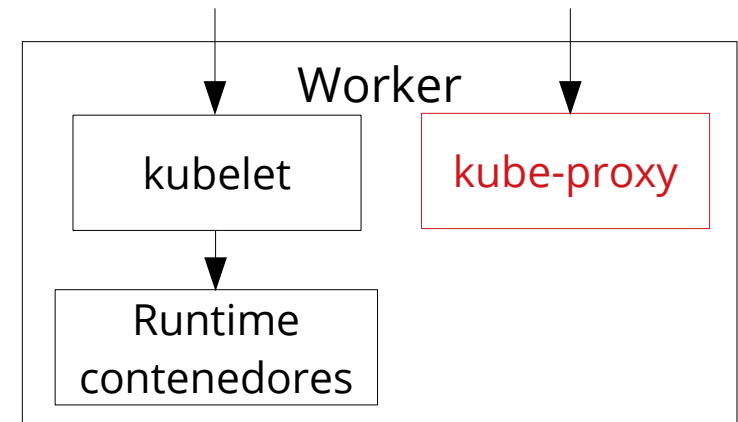


Kubernetes - k8s

Arquitectura > Workers

Componentes:

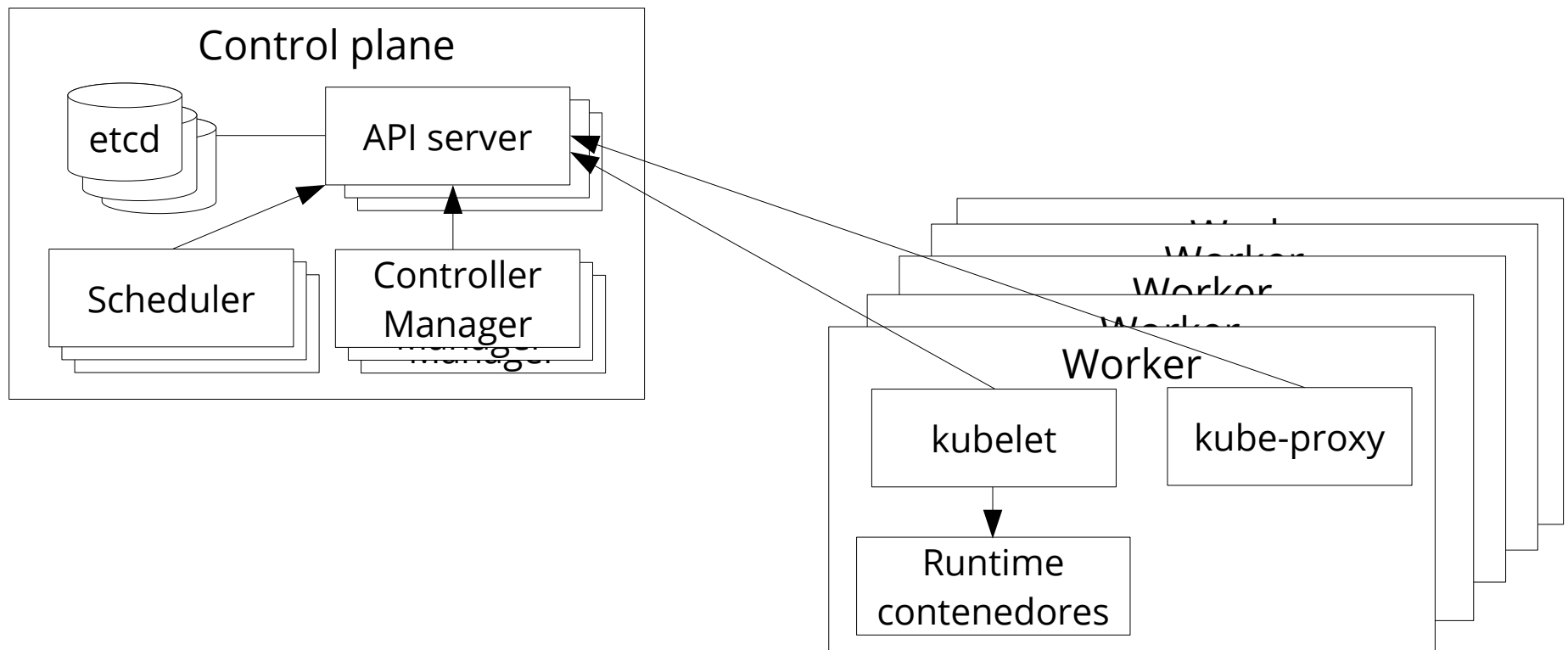
- kube-proxy
 - 1x agente en cada nodo
 - Habilita la comunicación con los contenedores
 - Utiliza reglas de filtrado y/u otros elementos de reenvío de tráfico



Kubernetes - k8s

Arquitectura

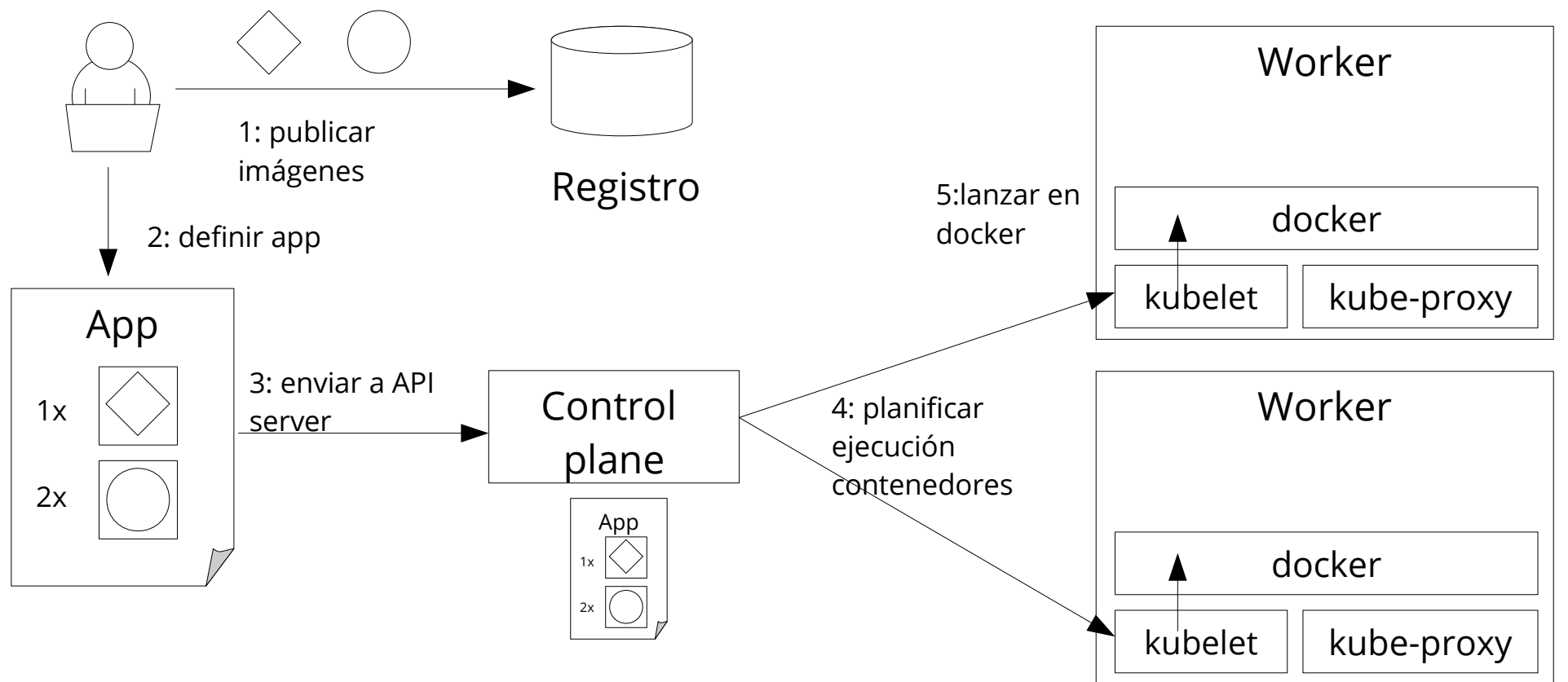
- Esquema global



Kubernetes - k8s

Funcionamiento

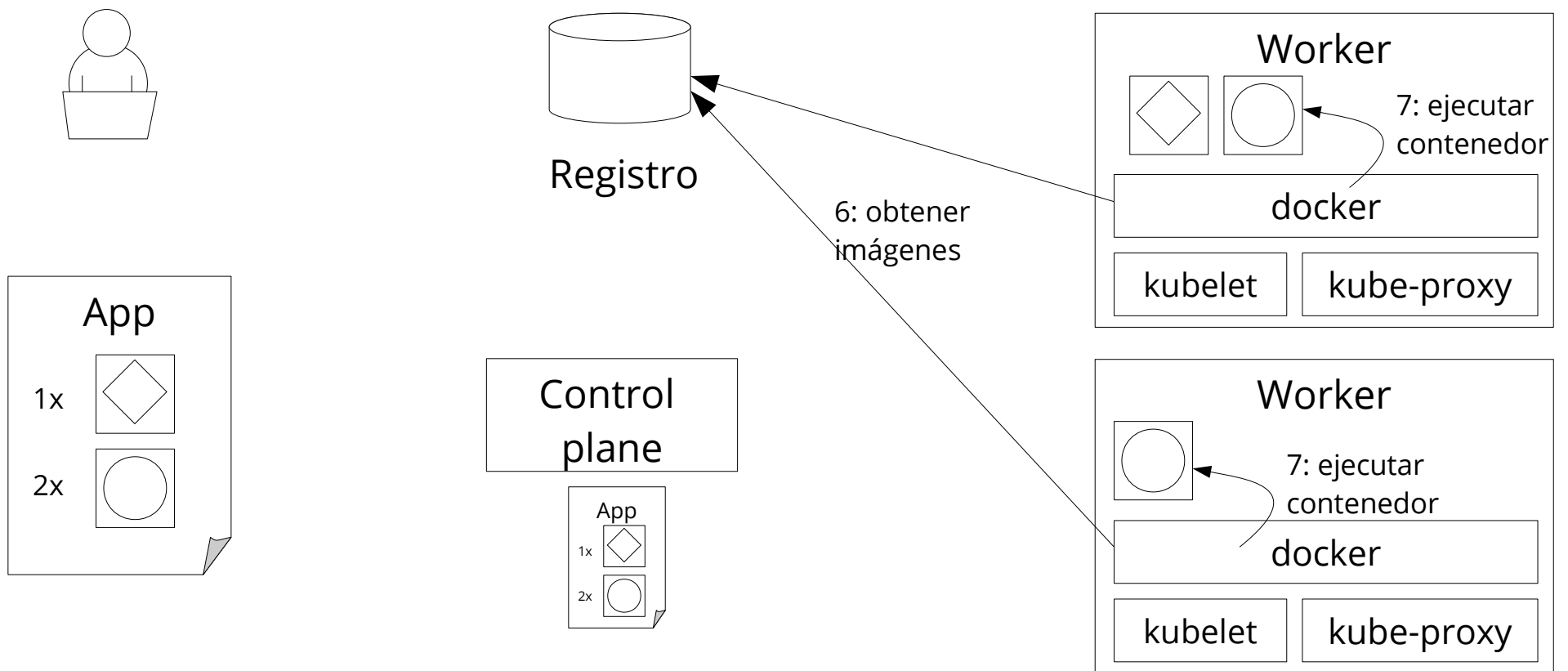
- Desplegar una aplicación



Kubernetes - k8s

Funcionamiento

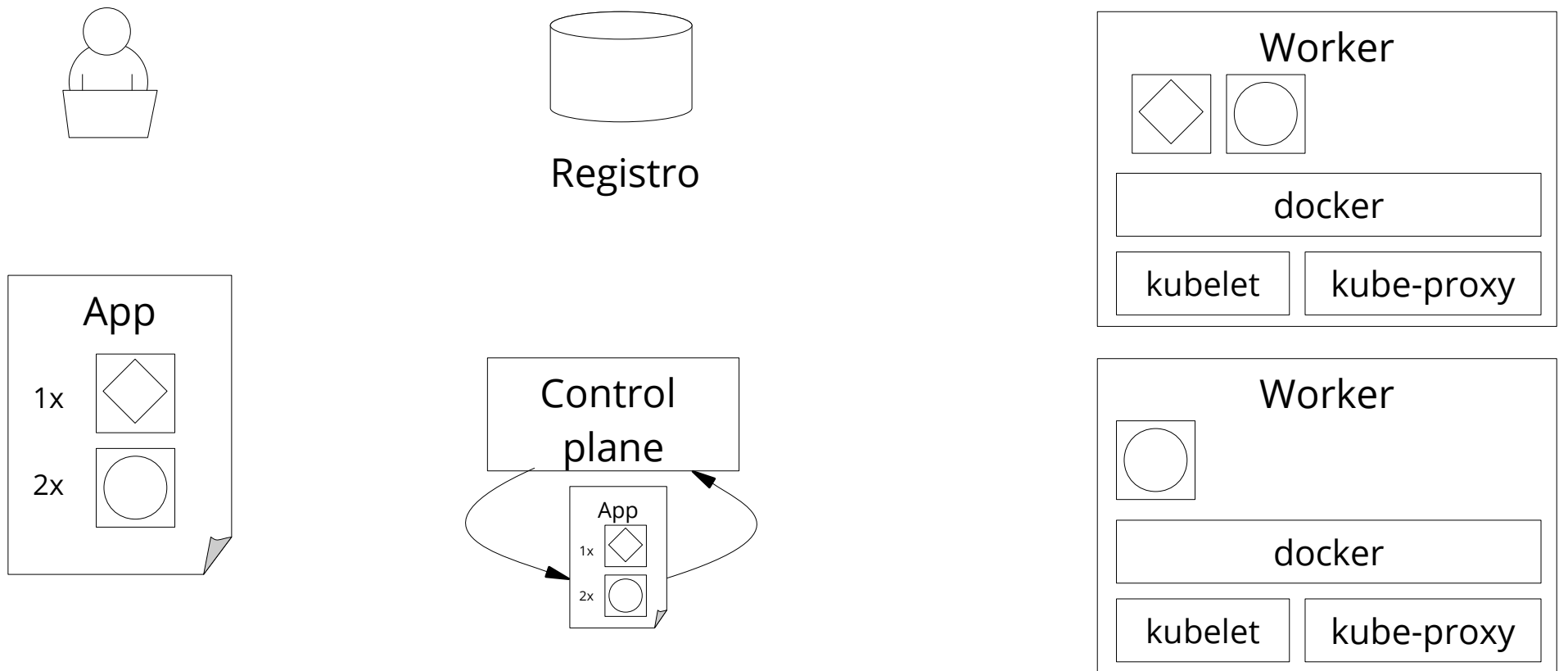
- Desplegar una aplicación



Kubernetes - k8s

Funcionamiento

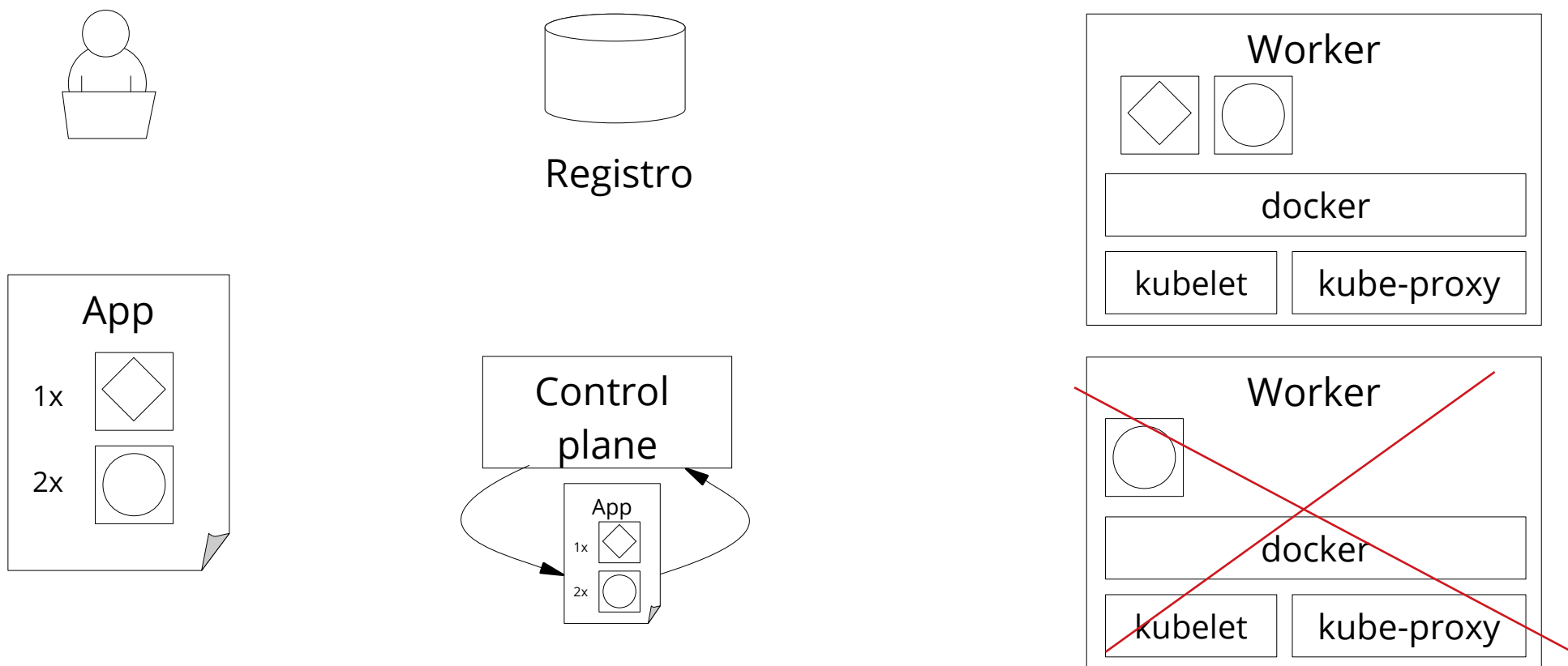
- Garantizar estado de app coincide con spec



Kubernetes - k8s

Funcionamiento

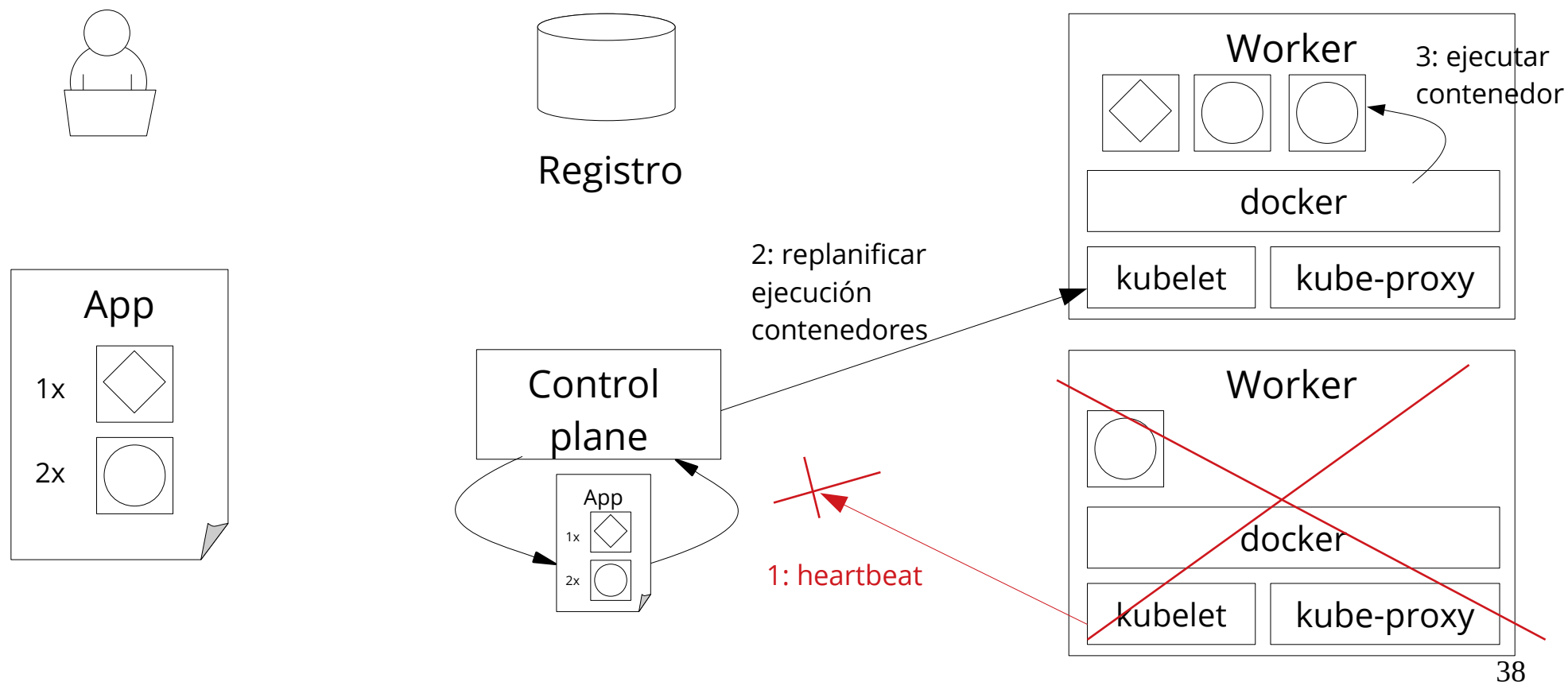
- Garantizar estado de app coincide con spec



Kubernetes - k8s

Funcionamiento

- Garantizar estado de app coincide con spec



Kubernetes - k8s

Instalación

- En producción: configuración multi-nodo
 - En desarrollo: configuración mono-nodo
 - [minikube](#), [kind](#), [k3s](#), [k3d](#), [Microk8s](#), ...
 - [k3s](#)
 - Kubernetes minimalista (512 Mb vs 8Gb)
 - En lugar de etcd utiliza sqlite
- ```
> curl -sfL https://get.k3s.io |
 sh -s - --write-kubeconfig-mode 644
```

# Kubernetes - k8s

## API

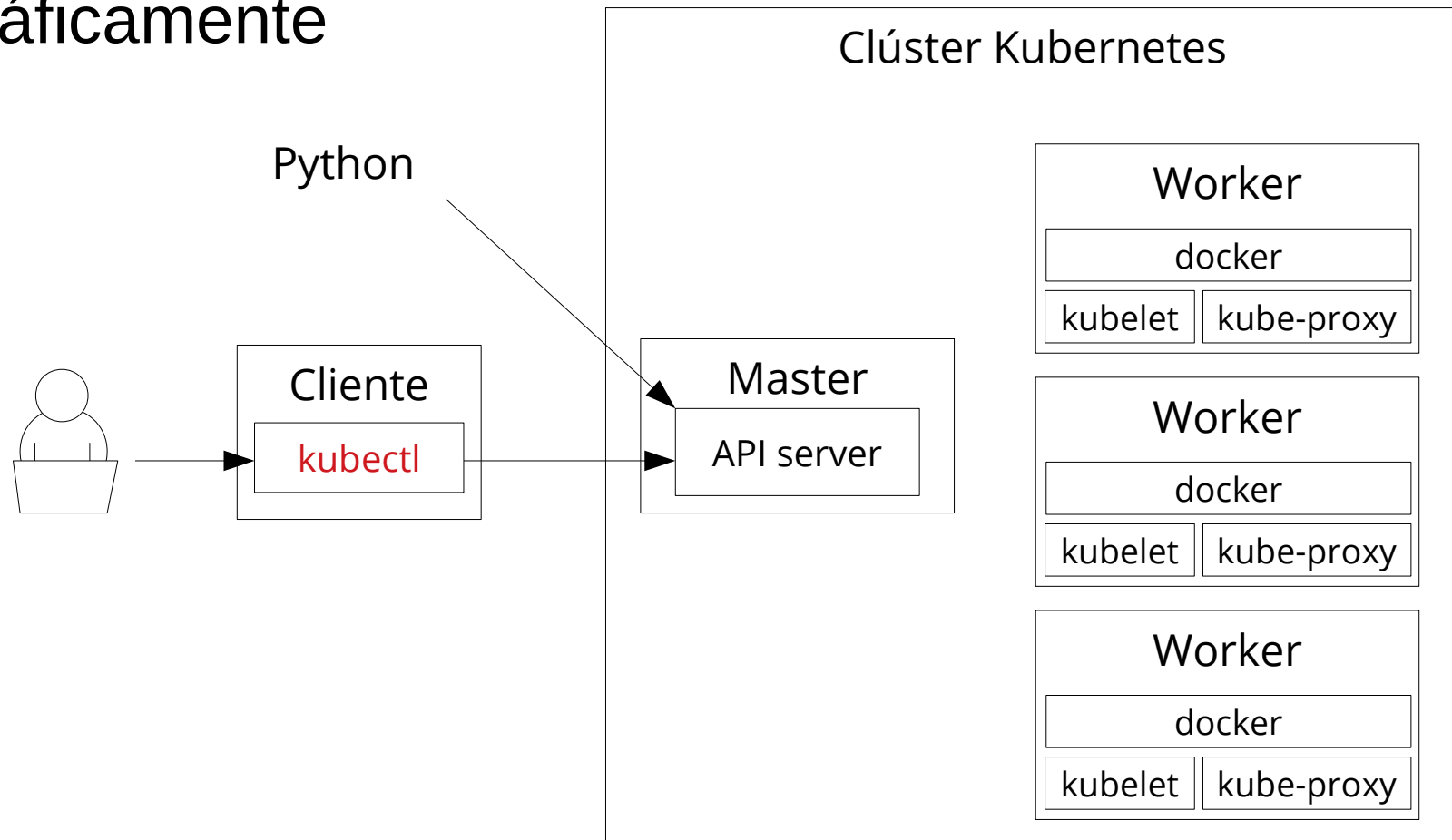
- Punto de entrada a k8s
- API RESTful publicada por el API server
- Publica los objetos (recursos) que k8s gestiona
- La API permite consultarlos/manipularlos
- Accesible a través de kubectl, kubeadm, Python, ...



# Kubernetes - k8s

## API

- Gráficamente



# Kubernetes - k8s

## API > kubectl

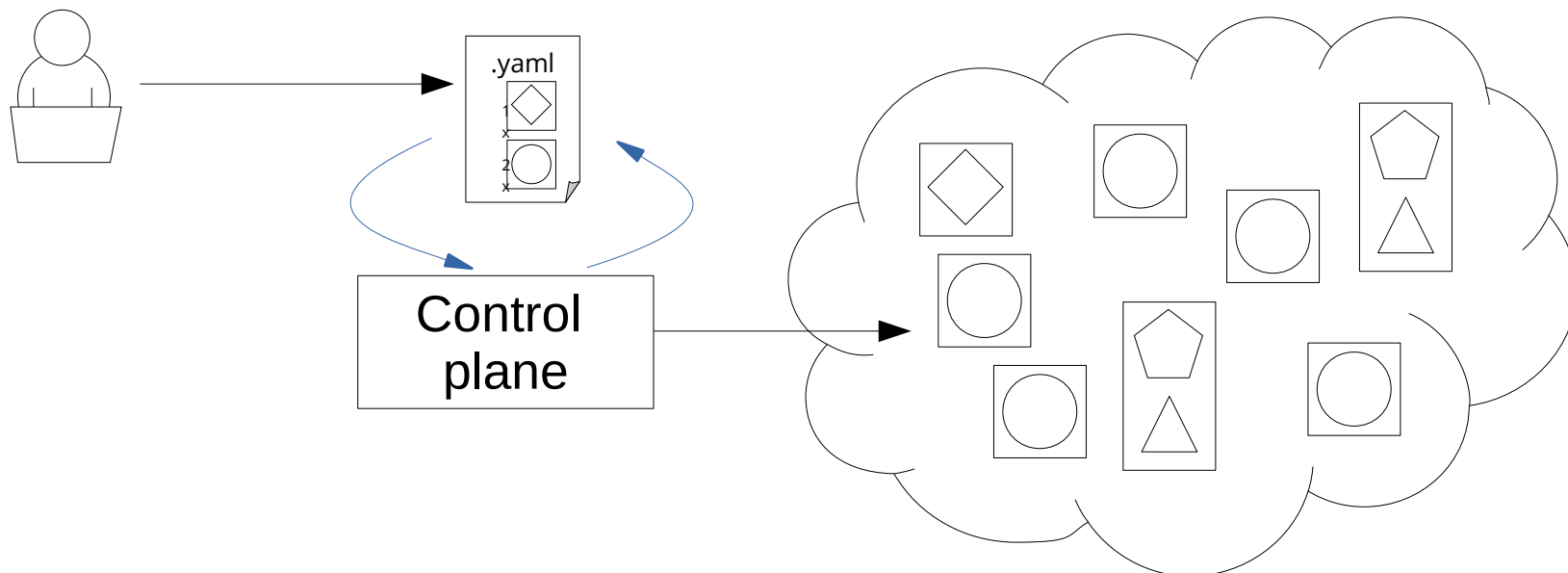
- CLI para consultar/manipular el clúster
  - > kubectl [cmd] [TYPE] [NAME] [flags]
- cmd: create, get, describe, delete, ... TYPE: node/s, pod/s, service/s, ..., NAME: el nombre del recurso
  - > kubectl cluster-info
  - > kubectl **get** nodes
  - > kubectl **describe** node [<node-name>]
  - > kubectl **run** test --image=nginx

# Kubernetes - k8s

## API > kubectl

- Es más habitual especificar el estado deseado en un fichero **.yaml**

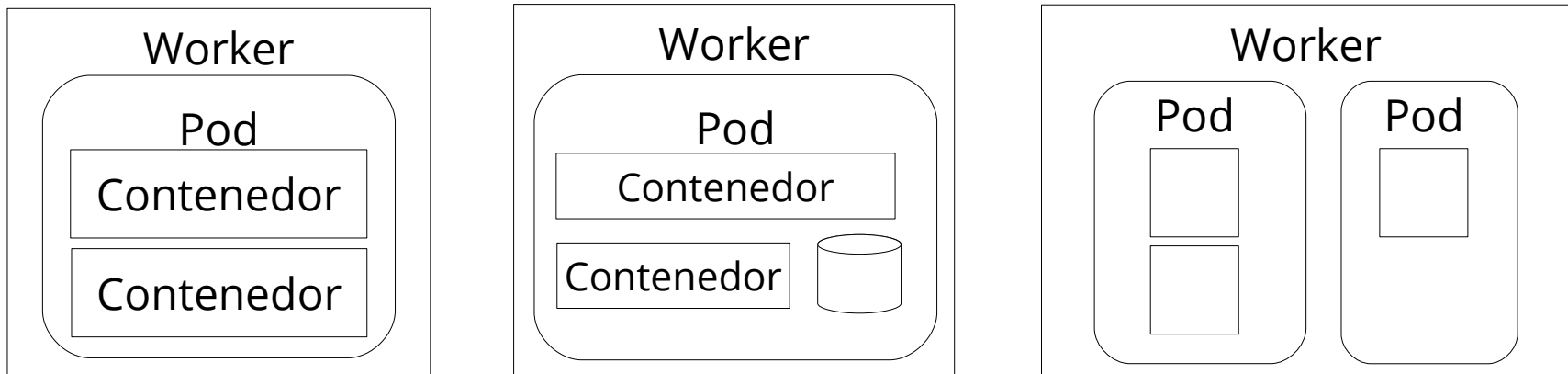
```
> kubectl apply -f test.yaml
```



# Kubernetes - k8s

## Pod

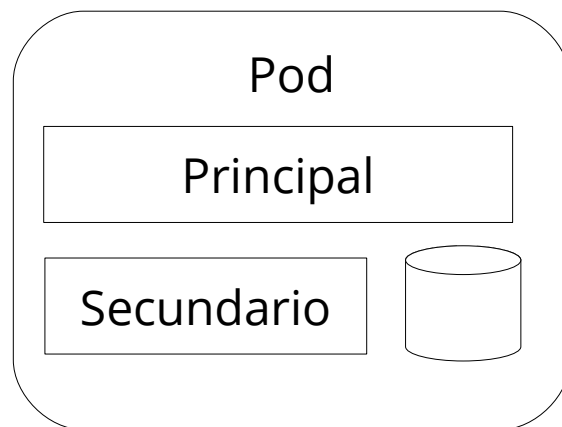
- Unidad básica de despliegue en k8s
- Grupo de contenedores (1 ó más) con alto acoplamiento que se ejecutan en el mismo nodo
- Comparten namespace, pila de red y almacenamiento



# Kubernetes - k8s

## Pod

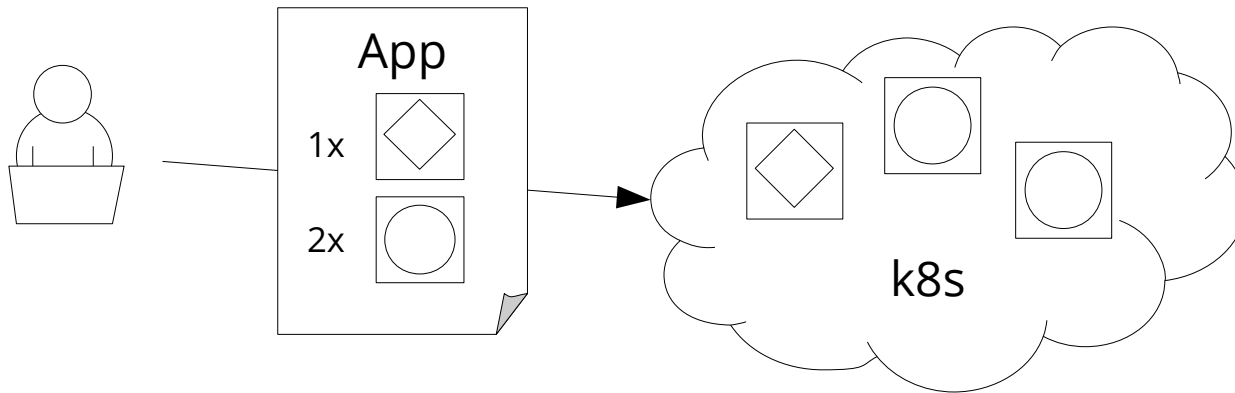
- Ejemplos: contenedor principal + secundario/s
  - Servidor web + contenedor que sync ficheros
  - Servidor + contenedor que rota logs
  - Servidor + procesador de datos (entrada/salida)



# Kubernetes - k8s

## Pod

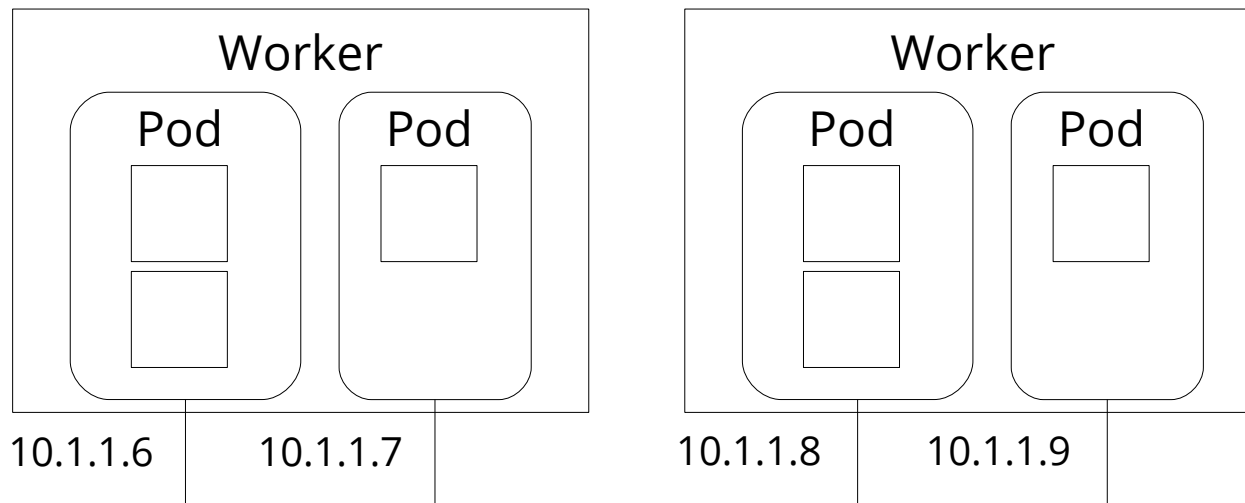
- Kubernetes decide en qué workers se ejecutan los pods



# Kubernetes - k8s

## Pod

- Kubernetes decide en qué workers se ejecutan los pods
- Todos los pods se ven en un espacio de red plano y tienen acceso al exterior



# Kubernetes - k8s

## Pod > Creación

- Con kubectl run (limitado, pocas opciones)
  - > `kubectl run nginx --image nginx --port 80`



# Kubernetes - k8s

## Pod > Creación

- Con kubectl run (limitado, pocas opciones)
- Con especificación de objeto de la API en YAML

```
> kubectl create/apply -f nginx-pod.yaml
```

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
 name: nginx2
```

```
spec:
```

```
 containers:
```

```
 - name: nginx
```

```
 image: nginx
```

```
 ports:
```

```
 - containerPort: 80
```

```
 protocol: TCP
```

Metadatos: nombre, namespace, etiquetas, etc.

Especificación: contenedores, volúmenes, etc.

# Kubernetes - k8s

## Pod > Listado

- Listado

```
> kubectl get pods
> kubectl get pod nginx
> kubectl describe pod nginx
```

- Etiquetas: agrupar pods

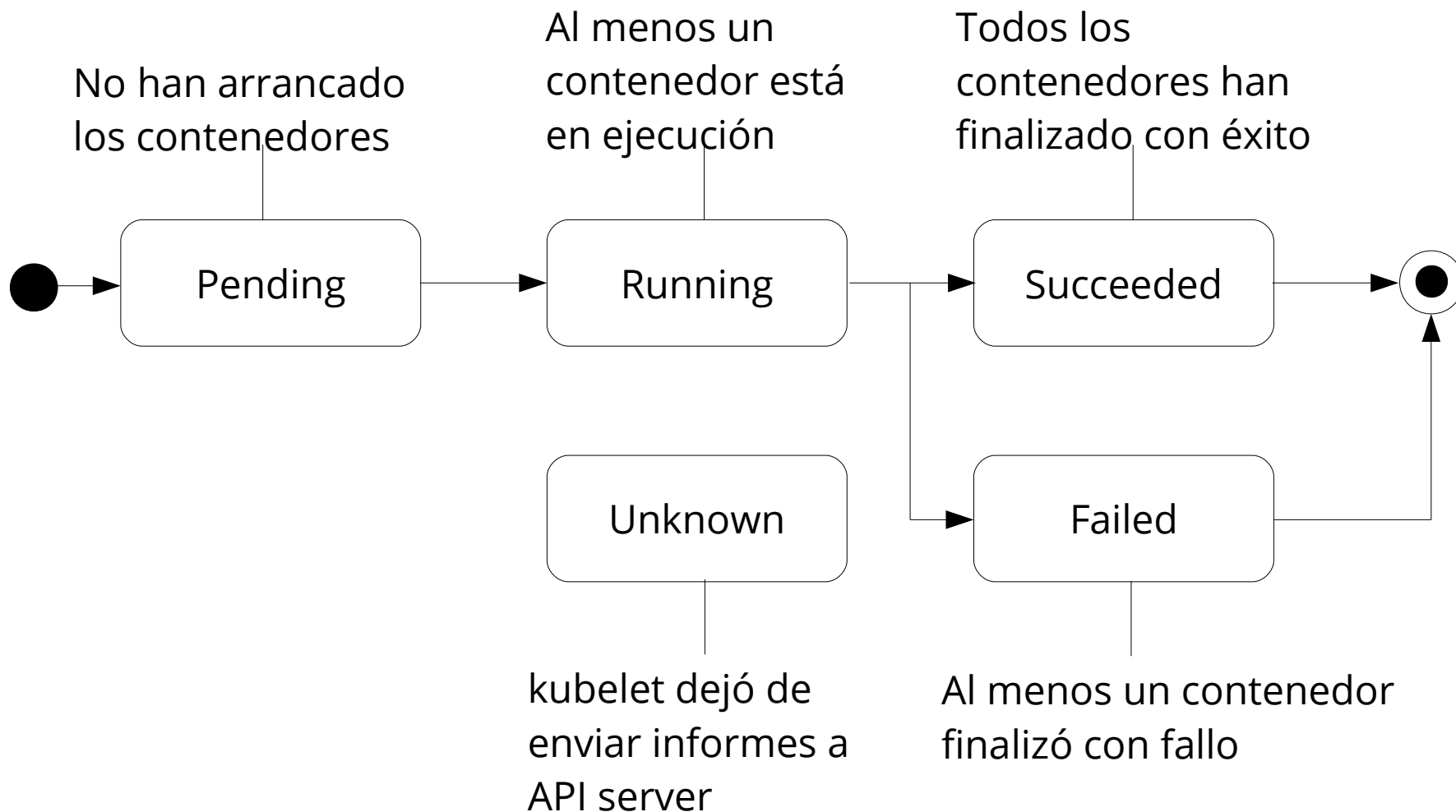
```
> kubectl create/apply -f example-pod.yaml
> kubectl get pods -l app=web-server
> kubectl get pods -l version
> kubectl get pods -l 'version in (1.0,2.0)'
> kubectl get pods -l app=web-server,version=1.0
```

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx
 labels:
 app: web-server
 version: "1.0"
spec:
 ...
```



# Kubernetes - k8s

## Pod > Ciclo de vida



# Kubernetes - k8s

## Pod > Ciclo de vida

- ¿Cómo sabe k8s el estado de un contenedor/pod?
- kubelet monitoriza de manera constante el estado de los contenedores
  1. Monitoriza el estado de ejecución del contenedor
  2. A través de **pruebas** periódicas, definidas por el usuario, que comprueban el estado de los contenedores

# Kubernetes - k8s

## Pod > Pruebas

- Una prueba ejecuta un **manejador** definido por el usuario
- Cada prueba obtiene un resultado (Success, Failure, Unknown)
- Si la prueba falla, k8s reinicia (o no) dependiendo del valor de spec.**restartPolicy** (**Always**, OnFailure, Never)

# Kubernetes - k8s

## Pod > Pruebas > Tipos

- **Liveness**: ¿el contenedor está en ejecución?
- **Readiness**: ¿el contenedor está listo para responder peticiones?
- **Startup**: ¿el contenedor ha arrancado?

```
apiVersion: v1
kind: Pod
metadata:
 name: test
spec:
 containers:
 - image: nginx
 livenessProbe: ...
 readinessProbe: ...
 startupProbe: ...
```

# Kubernetes - k8s

## Pod > Pruebas > Manejadores

- **ExecAction**: ejecuta un comando dentro del contenedor; 0 es éxito
- **TCPSocketAction**: conexión TCP sobre un puerto; si está abierto éxito.
- **HTTPGetAction**: petición HTTP sobre puerto/path; si 200 <=código<400 éxito.

# Kubernetes - k8s

## Pod > Pruebas > Ejemplo

```
apiVersion: v1
kind: Pod
metadata:
 name: nginx3
spec:
 containers:
 - name: nginx
 image: nginx
 livenessProbe:
 httpGet:
 path: /
 port: 80
 readinessProbe:
 exec:
 command: ["ls", "/www"]
 startupProbe:
 tcpSocket:
 port: 80
```



# Kubernetes - k8s

## Pod > Depuración

- Logs

- > `kubectl logs nginx`

- Ejecutar pod interactivo

- > `kubectl run -it debian --image=debian`

- > `kubectl attach nginx`

- Ejecutar comandos

- > `kubectl exec nginx [-c <contenedor>] -- ls -la`

# Kubernetes - k8s

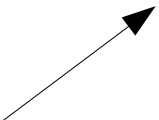
## Pod > Namespace

- Permite agrupar objetos de la API.
- Múltiples usuarios, proyectos, etc. (multi-tenancy)
- Namespace por defecto **default**

- Comandos:

```
> kubectl get ns/namespaces
> kubectl create/apply -f example-ns.yaml
> kubectl run nginx --image=nginx --namespace example-ns
> kubectl get pods --namespace=example-ns
```

```
apiVersion: v1
kind: Namespace
metadata:
 name: example-ns
```



# Kubernetes - k8s

## Pod > Eliminación

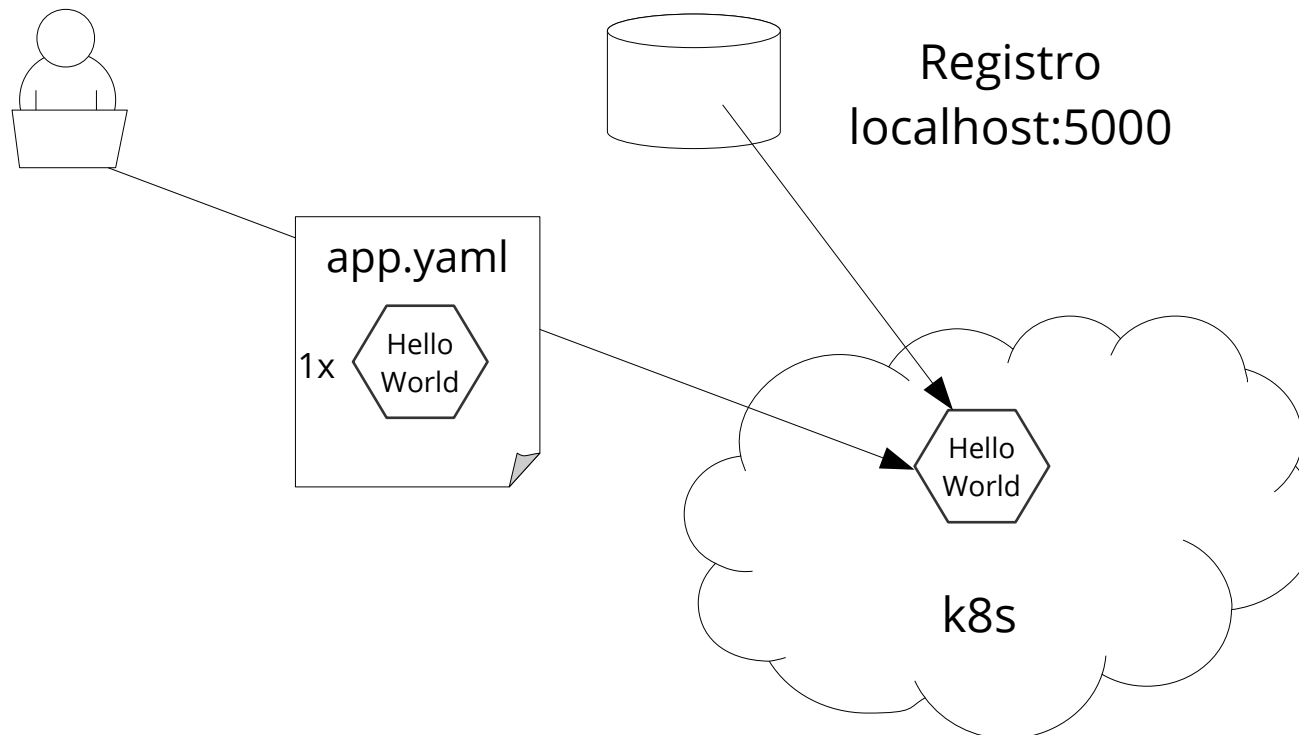
- Parar y eliminar (por nombre, etiqueta)
  - > `kubectl delete pod test`
  - > `kubectl delete pod -l app=web-server`
- Eliminar recursos de namespace
  - > `kubectl delete pod --all`
  - > `kubectl delete pod --all --namespace example-ns`

# Kubernetes - k8s



## Ejercicio 2

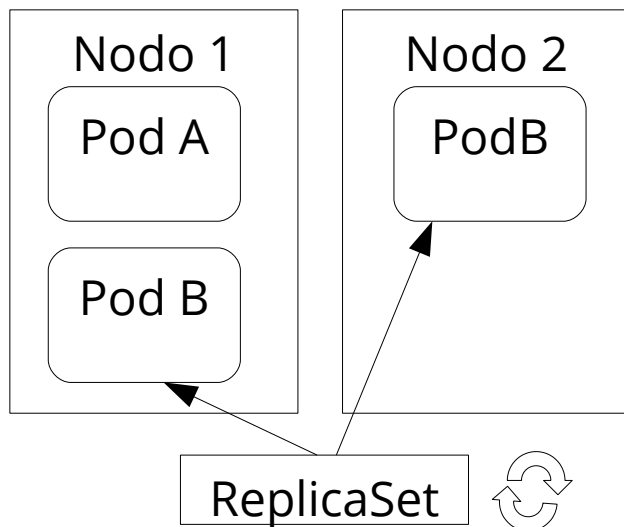
- Crear app.yaml con el pod HelloWorld FLASK registrado en un registro local



# Kubernetes - k8s

## ReplicaSet

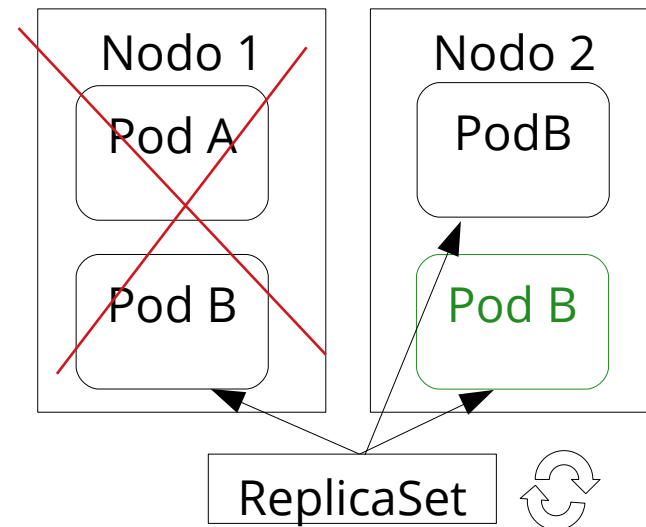
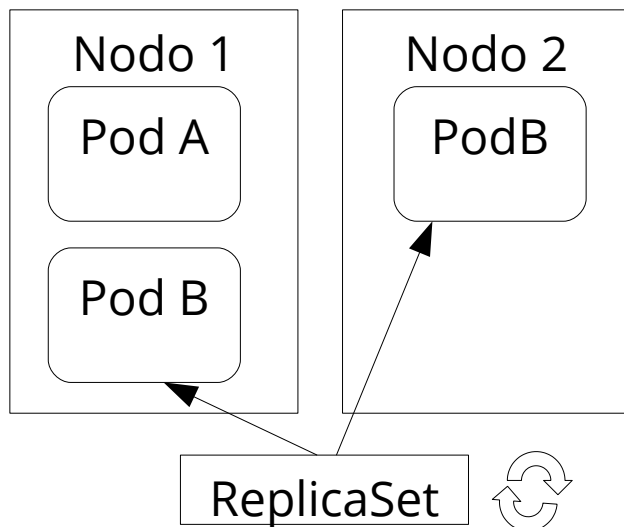
- Controlador que arranca y mantiene en ejecución un conjunto replicado de pods
- Comprueba **en bucle** el número de réplicas vivas



# Kubernetes - k8s

## ReplicaSet

- Controlador que arranca y mantiene en ejecución un conjunto replicado de pods
- Comprueba **en bucle** el número de réplicas vivas
- Si una réplica falla, lo detecta y crea otra nueva



# Kubernetes - k8s

## ReplicaSet

- Controlador que arranca y mantiene en ejecución un conjunto replicado de pods
- Comprueba **en bucle** el número de réplicas vivas
- Si una réplica falla, lo detecta y crea otra nueva
- Las réplicas son **efímeras**, se crean y destruyen dinámicamente
- Cada réplica nueva posee su propio id, dirección IP, etc.

# Kubernetes - k8s

## ReplicaSet > Especificación

```
> kubectl create/apply -f example-rs.yaml
```

- Número de réplicas: si hay menos, se crean; si hay más, se destruyen

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: test
spec:
 replicas: 3
 selector:
 matchLabels:
 app: web-server
 template:
 ...
```




# Kubernetes - k8s

## ReplicaSet > Especificación

> kubectl create/apply -f example-rs.yaml

- Selector: determina qué réplicas controla
  - matchLabels, matchExpressions

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: test
spec:
 replicas: 3
 selector:
 matchLabels:
 app: web-server
 template:
 ...
```



```
selector:
 matchExpressions:
 - {key: app, operator: In,
 values: [web-server, server]}
 - {key: version, operator: NotIn,
 values: ["2.0", "3.0"]}
```


# Kubernetes - k8s

## ReplicaSet > Especificación

> kubectl create/apply -f example-rs.yaml

- Plantilla: contiene la especificación para crear el pod (las etiquetas deben coincidir con selector)

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: test
spec:
 replicas: 3
 selector:
 matchLabels:
 app: web-server
 template:
 metadata:
 labels:
 app: web-server
 version: "1.0"
 spec:
 containers:
 - name: nginx
 image: nginx
 ports:
 - containerPort: 80
 protocol: TCP
```



# Kubernetes - k8s

## ReplicaSet > Listado

- > `kubectl get rs [<name>]`
- > `kubectl describe rs [<name>]`

# Kubernetes - k8s

## ReplicaSet > Escalado

- Editando `.spec.replicas` en la especificación
  - > `kubectl edit rs <name>`
- Con el comando `kubectl scale`
  - > `kubectl scale rs <name> --replicas=10`
- Volviendo a aplicar la especificación
  - > `kubectl apply -f example-rs.yaml`

# Kubernetes - k8s

## ReplicaSet > Eliminación

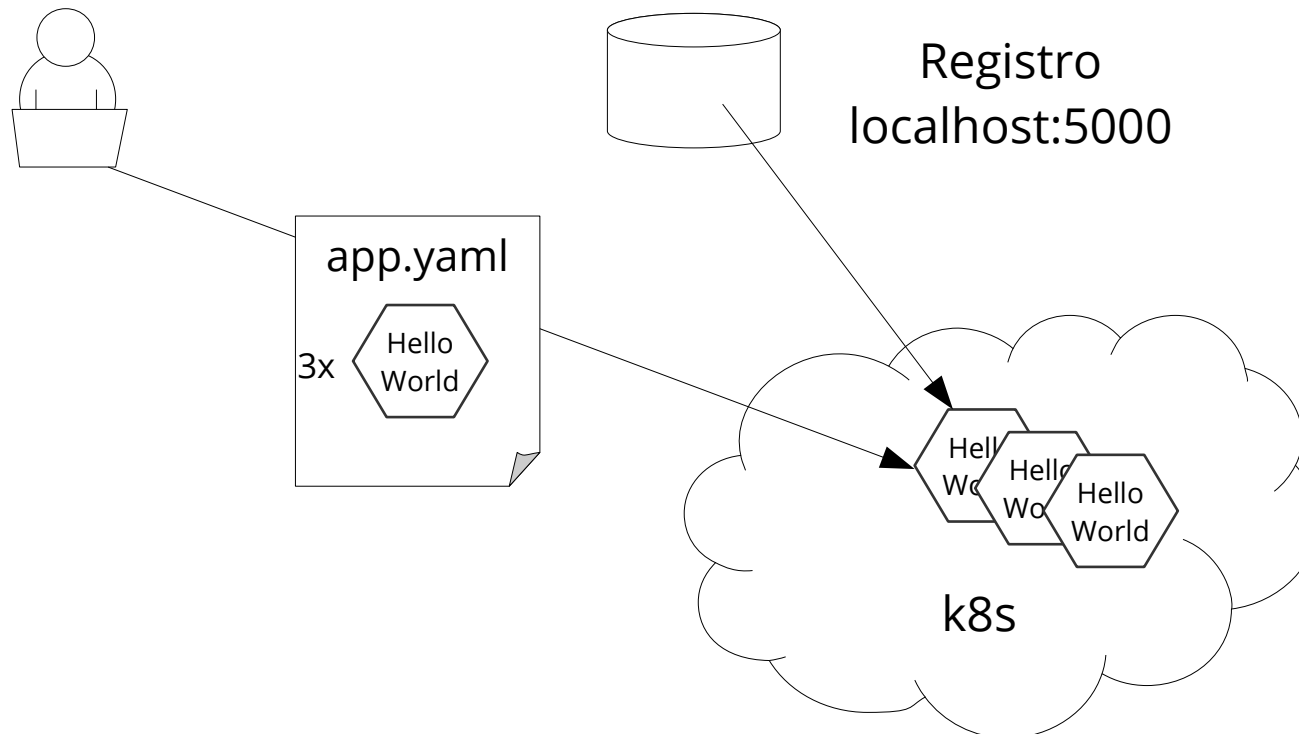
- Se elimina en cascada junto con todas las replicas  
> `kubectl delete rs <name>`
- Para evitarlo `--cascade=false`:  
> `kubectl delete rs --cascade=false <name>`

# Kubernetes - k8s



## Ejercicio 3

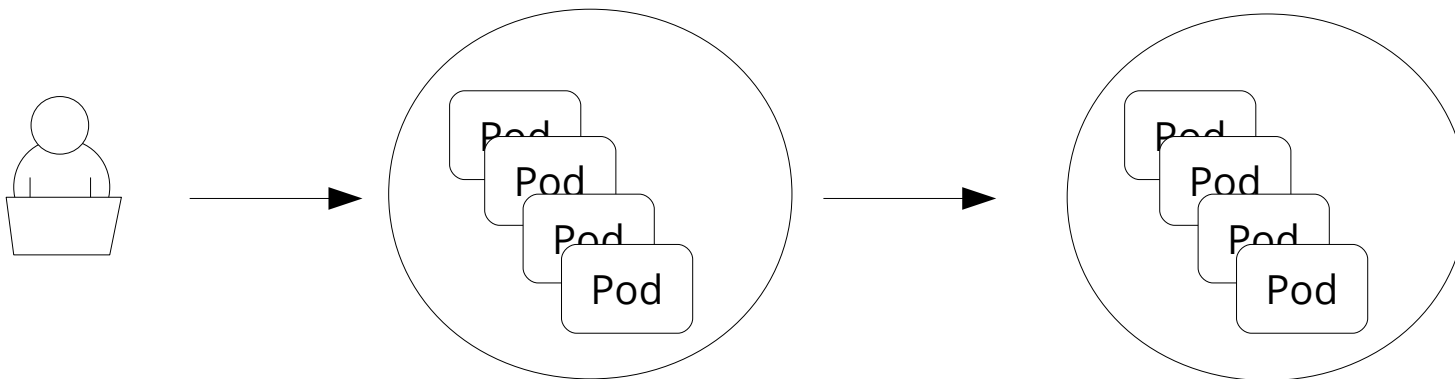
- Crear app.yaml con 3 réplicas del pod HelloWorld FLASK registrado en un registro local



# Kubernetes - k8s

## Service

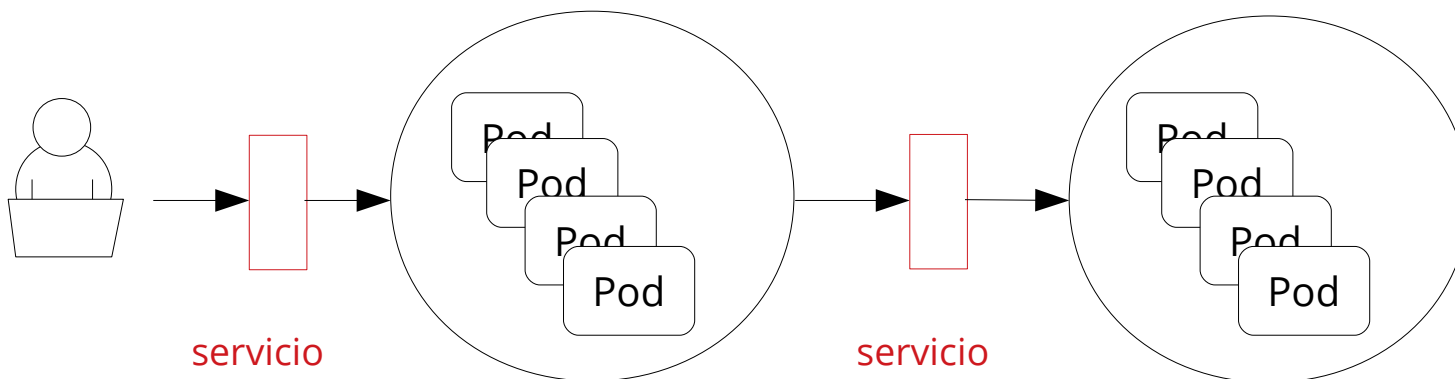
- Los pods se crean y se destruyen, son efímeros
- Algunos pods deben ser accedidos por otros pods, o externamente



# Kubernetes - k8s

## Service

- Los pods se crean y se destruyen, son efímeros
- Algunos pods deben ser accedidos por otros pods, o externamente
- Un **servicio** oculta una colección de pods bajo una **dirección IP fija**





# Kubernetes - k8s

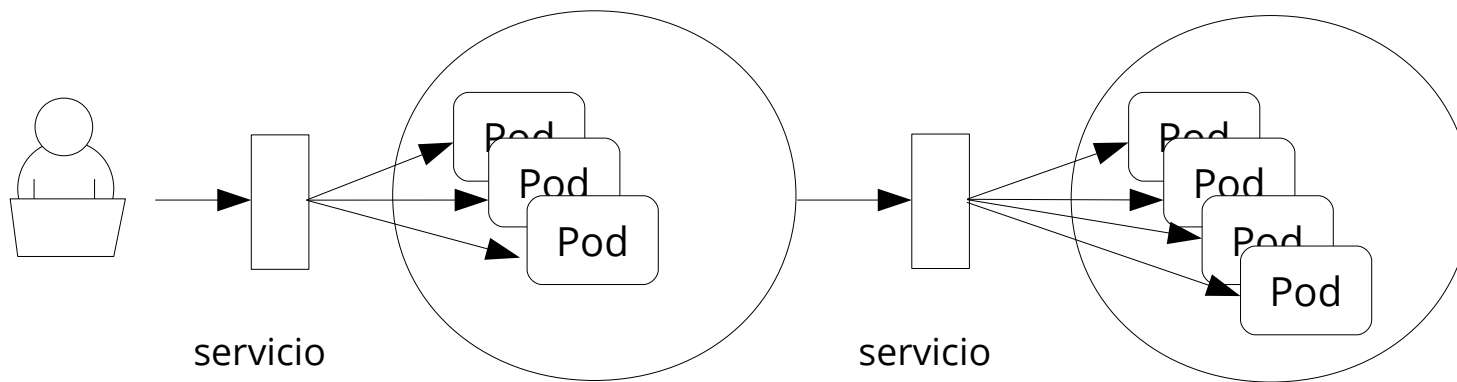
## Service

- Los pods se crean y se destruyen, son efímeros
- Algunos pods deben ser accedidos por otros pods, o externamente
- Un **servicio** oculta una colección de pods bajo una **dirección IP fija**
- Los pods que gestiona el servicio se definen por medio de un **selector** (**etiquetas**)

# Kubernetes - k8s

## Service

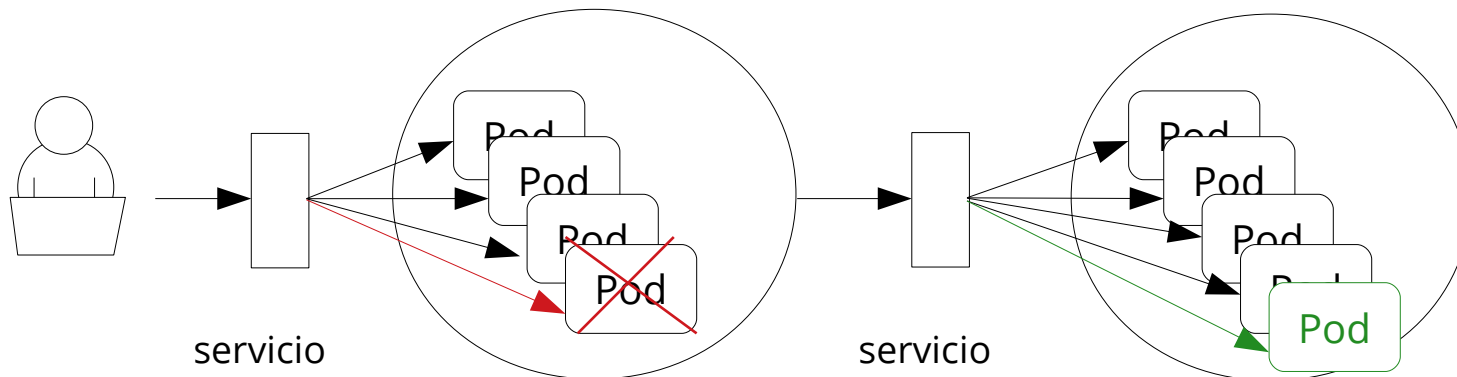
- El servicio balancea la carga entre los distintos pods



# Kubernetes - k8s

## Service

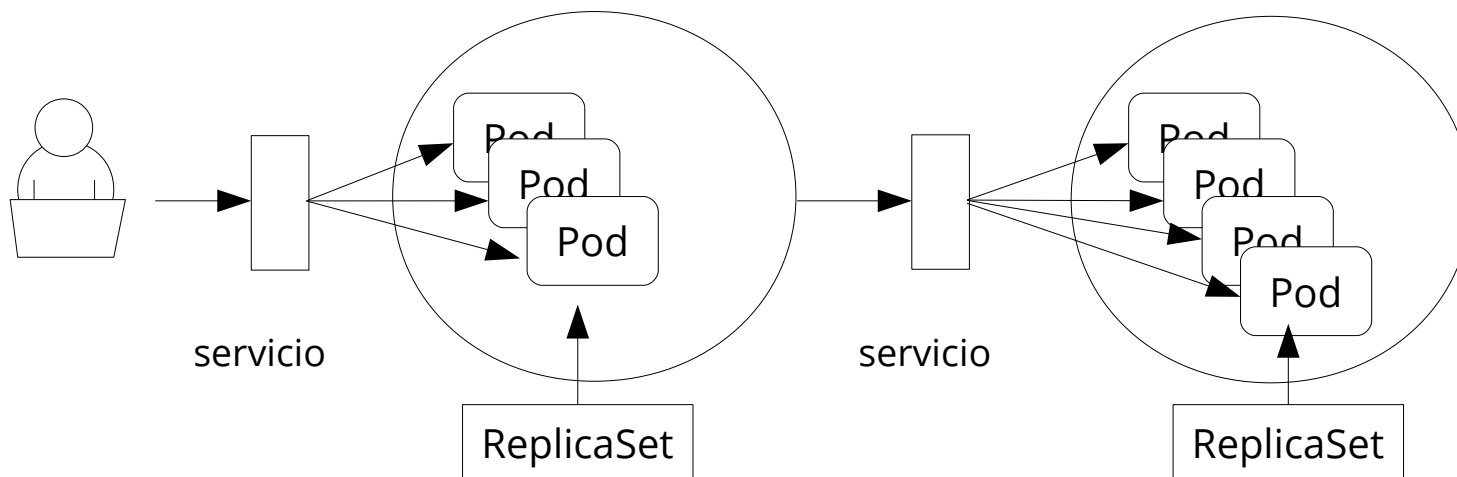
- El servicio balancea la carga entre los distintos pods
- El servicio efectúa un seguimiento de los pods que se crean/destruyen



# Kubernetes - k8s

## Service

- El servicio balancea la carga entre los distintos pods
- El servicio efectúa un seguimiento de los pods que se crean/destruyen
- Los pods suelen estar controlados por ReplicaSet



# Kubernetes - k8s

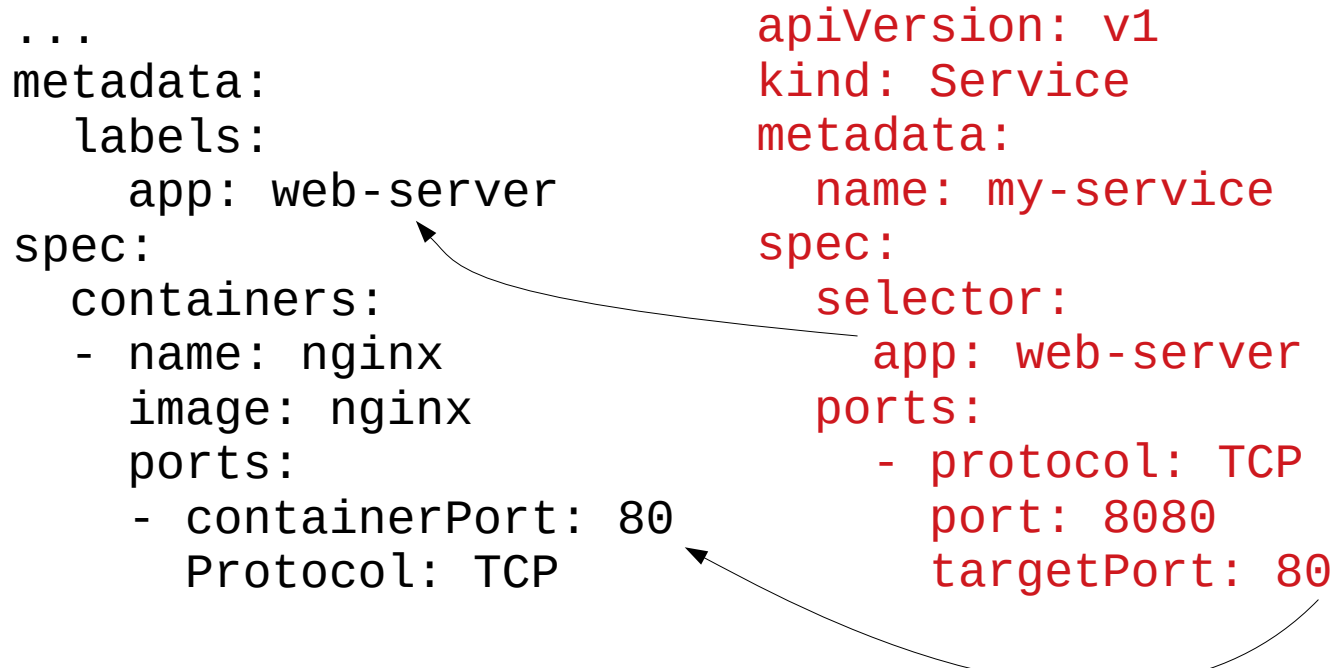
## Service > Especificación

- selector, ports

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: test
spec:
 replicas: 3
 selector:
 matchLabels:
 app: web-server
 template:
 ...
```

```
...
metadata:
 labels:
 app: web-server
spec:
 containers:
 - name: nginx
 image: nginx
 ports:
 - containerPort: 80
 Protocol: TCP
```

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
 app: web-server
 ports:
 - protocol: TCP
 port: 8080
 targetPort: 80
```



# Kubernetes - k8s

## Service > Especificación

- Crear servicio, visible desde cualquier pod

```
> kubectl create/apply -f my-service.yaml
```

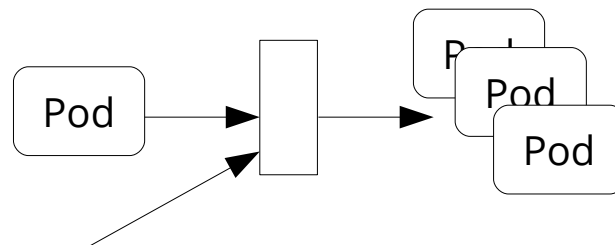
```
> kubectl get svc
```

| NAME       | CLUSTER-IP   | EXTERNAL-IP | PORT(S)  |
|------------|--------------|-------------|----------|
| my-service | 10.111.240.1 | <none>      | 8080/TCP |

```
> curl 10.111.240.1:8080
```

```
> kubectl run -it debian --image=debian
```

```
> kubectl exec nginx -- curl http://10.111.240.1:8080
```



# Kubernetes - k8s

## Service > Descubrimiento

- Los pods deben conocer la IP de los servicios
- Los servicios se publican utilizando dos mecanismos:
  - Variables de entorno
  - DNS

# Kubernetes - k8s

## Service > Descubrimiento por variables

- Cuando un pod arranca, kubelet define en sus variables de entorno todos los servicios activos

{SVCNAME}\_SERVICE\_HOST  
{SVCNAME}\_SERVICE\_PORT

minúsculas → mayúsculas

'-' → '\_'

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 selector:
 app: web-server
 ports:
 - protocol: TCP
 port: 8080
 targetPort: 80
```

MY\_SERVICE\_SERVICE\_HOST  
MY\_SERVICE\_SERVICE\_PORT



# Kubernetes - k8s

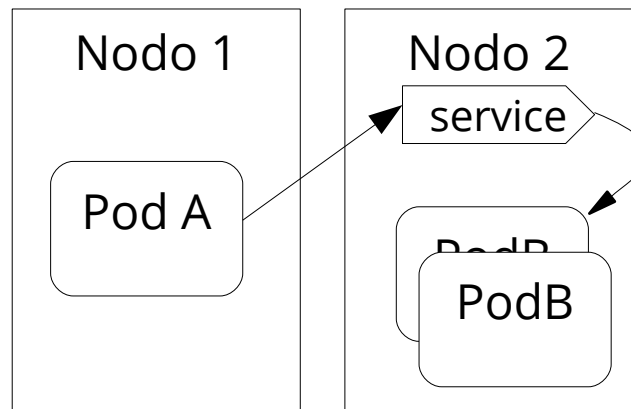
## Service > Descubrimiento por DNS

- K8s arranca un servidor DNS en el namespace kube-system (coredns)
  - > `kubectl get pods --namespace kube-system`
- Registra los nombres de todos los servicios
- Los pods pueden acceder al servicio por nombre
  - > `kubectl exec nginx -- curl my-service:8080`

# Kubernetes - k8s

## Service > Publicación

- Un servicio sólo es visible dentro del clúster
- Hay varias maneras de publicarlo externamente:
  - NodePort
  - LoadBalancer
  - Ingress

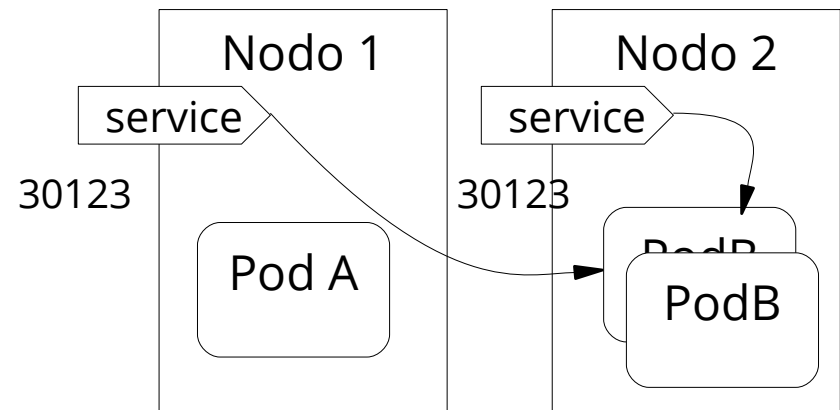


# Kubernetes - k8s

## Service > NodePort

- Cada nodo del clúster abre un puerto (30000-32767) y redirige el tráfico recibido al servicio
- El puerto estará ocupado en todos los nodos

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 type: NodePort
 ports:
 - port: 80
 targetPort: 8080
 nodePort: 30123
 selector:
 app: web-app
```

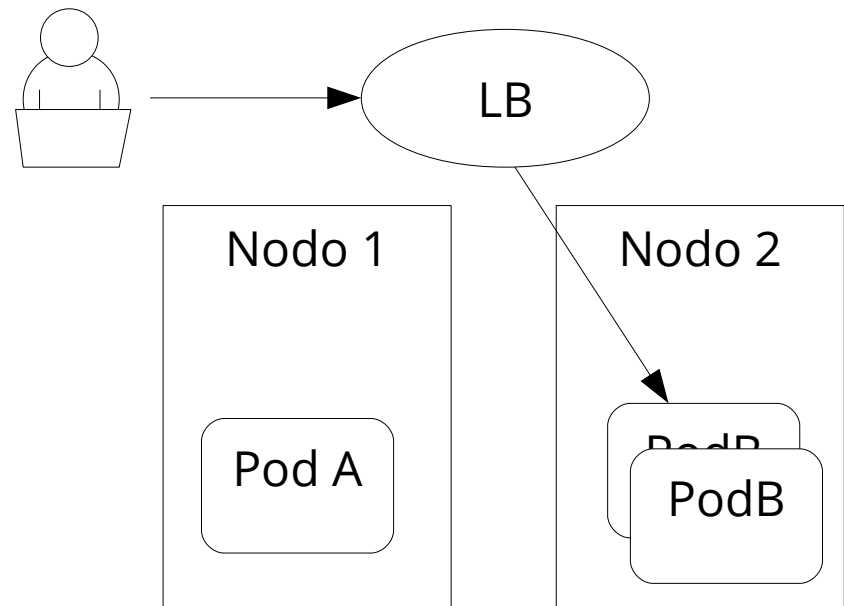


# Kubernetes - k8s

## Service > LoadBalancer

- El proveedor de cloud posee soluciones de LB
- El servicio se publica en dichas soluciones

```
apiVersion: v1
kind: Service
metadata:
 name: my-service
spec:
 type: LoadBalancer
 ports:
 - port: 80
 targetPort: 8080
 selector:
 app: web-app
```

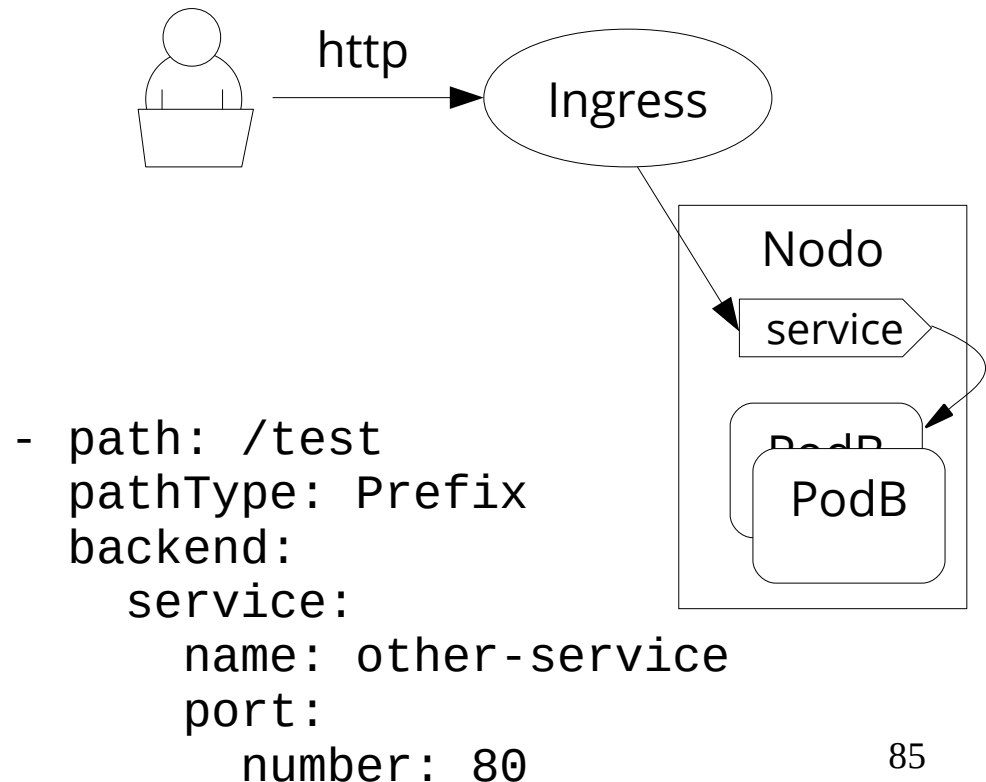


# Kubernetes - k8s

## Service > Ingress

- Sólo para servicios HTTP/HTTPS
- Redirige peticiones HTTP a los servicios

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
 name: my-ingress
spec:
 rules:
 - http:
 paths:
 - path: /
 pathType: Prefix
 backend:
 service:
 name: my-service
 port:
 number: 8080
```

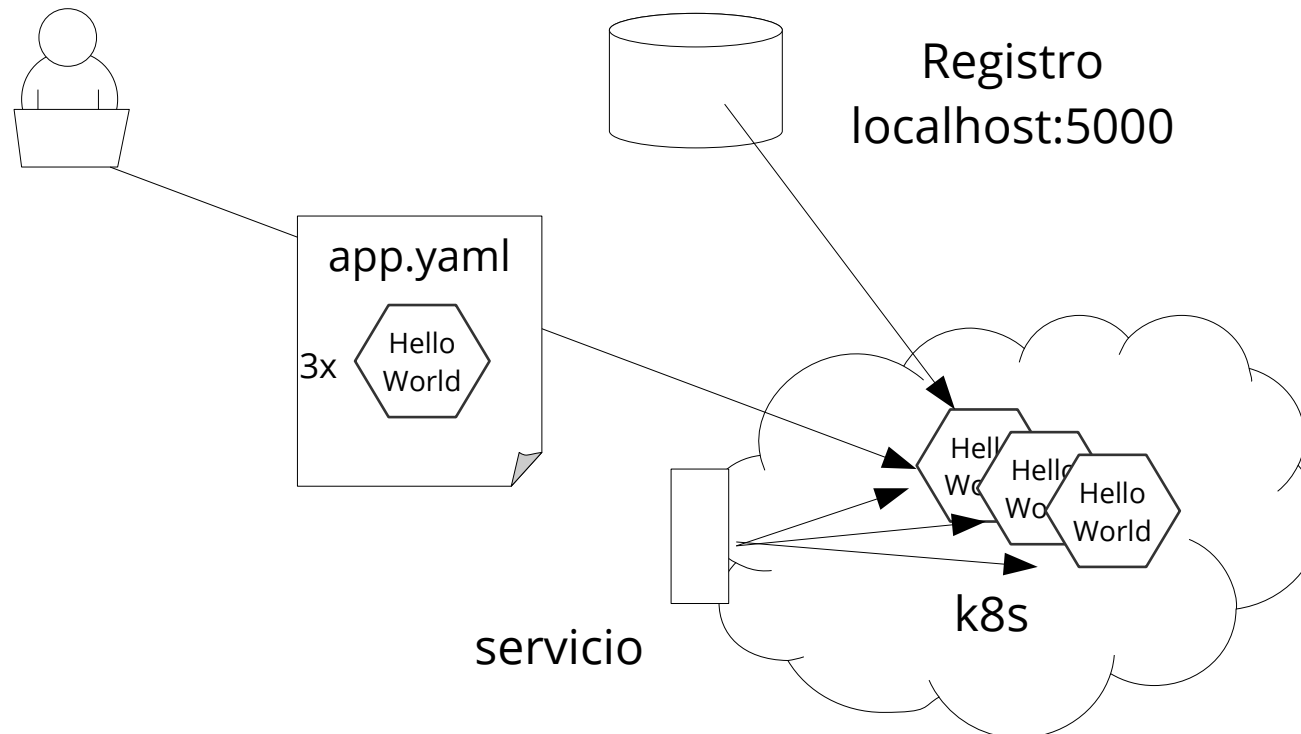


# Kubernetes - k8s



## Ejercicio 4

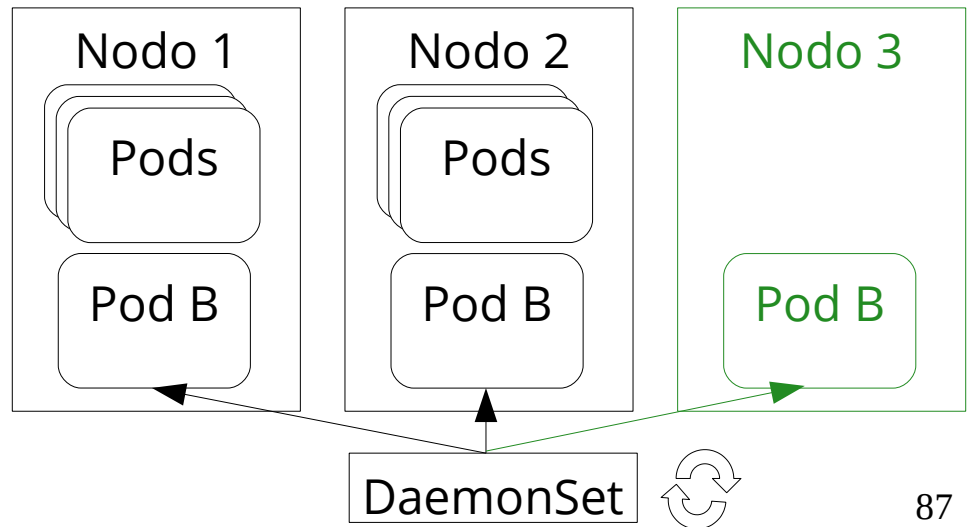
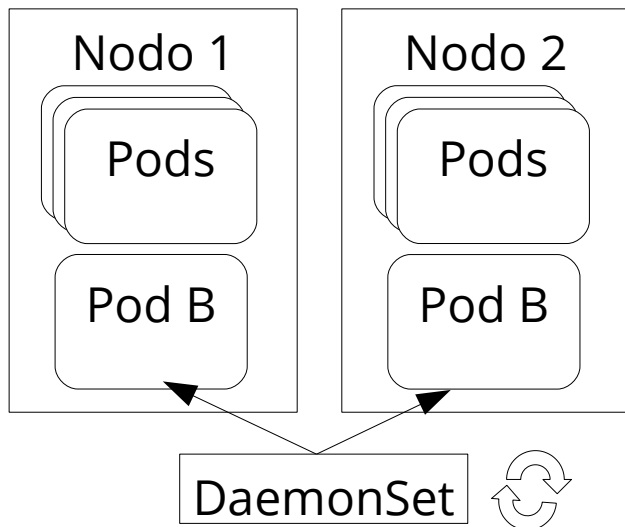
- Crear app.yaml con un servicio NodePort que apunta a las 3 réplicas del pod HelloWorld



# Kubernetes - k8s

## DaemonSet

- Crea un pod en cada nodo del clúster
- Comprueba **en bucle** todos los nodos del clúster
- Útil para servicios de infraestructura (log, monitorización, etc.)



# Kubernetes - k8s

## DaemonSet > Especificación

- En todos los nodos del clúster

```
apiVersion: apps/v1
```

```
kind: DaemonSet
```

```
metadata:
```

```
 name: test
```

```
spec:
```

```
 selector:
```

```
 matchLabels:
```

```
 app: log
```

```
 template:
```

```
 metadata:
```

```
 labels:
```

```
 app: log
```

```
 spec:
```

```
 containers:
```

```
 - name: main
```

```
 image: xxx/log
```

```
> kubectl apply -f test.yaml
```

```
> kubectl get ds
```



# Kubernetes - k8s

## DaemonSet > Especificación

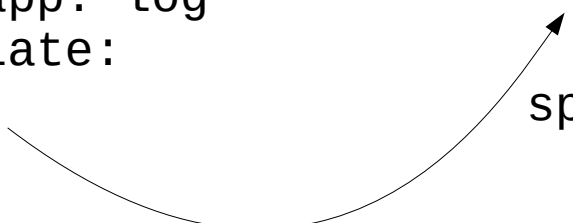
- Sólo en algunos nodos del clúster: etiquetas

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
 name: test
```

```
spec:
 selector:
 matchLabels:
 app: log
```

```
template:
```

```
metadata:
 labels:
 app: log
spec:
 nodeSelector:
 disk: big
 containers:
 - name: main
 image: xxx/log
```



# Kubernetes - k8s

## Job

- ReplicaSet/DaemonSet ejecutan tareas que no finalizan
- Job es un controlador que ejecuta una tarea, y se asegura de que finaliza correctamente
- Si la tarea no finaliza correctamente el Job la puede reiniciar
- Se pueden ejecutar múltiples tareas en paralelo
- Muy útil para **trabajos batch**

# Kubernetes - k8s

## Job > Especificación

- restartPolicy (Never, OnFailure), completions, parallelism

```
apiVersion: batch/v1
kind: Job
metadata:
 name: batch-job
spec:
```

```
 completions: 5
```

```
 parallelism: 2
```

```
 template:
```

```
 metadata:
```

```
 labels:
```

```
 app: batch-job
```

```
 spec:
```

```
 restartPolicy: OnFailure
```

```
 containers:
```

```
 - name: main
```

```
 image: debian
```

```
 command: ["echo", "batch-job"]
```

```
> kubectl apply -f test.yaml
```

```
> kubectl get job
```

# Kubernetes - k8s

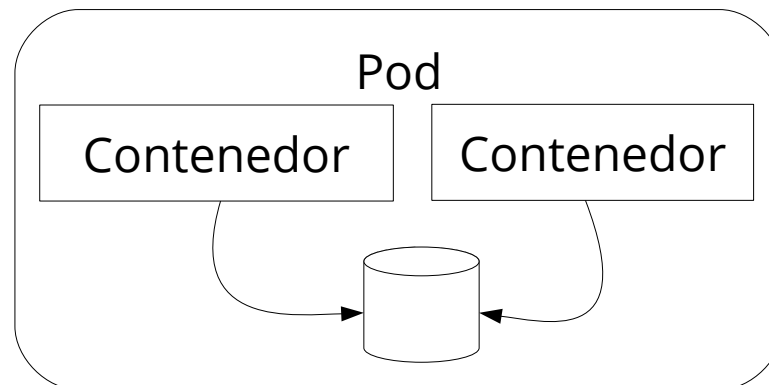
## Almacenamiento

- Sistema de ficheros del contenedor
  - Ligado al ciclo de vida del **contenedor**
- Volumen
  - Ligado al ciclo de vida del **pod**
- Volumen Persistente:
  - **Independiente** del ciclo de vida del pod

# Kubernetes - k8s

## Volumen

- Espacio de almacenamiento en k8s
- **Efímero**, ligado al ciclo de vida del pod
- Un volumen está disponible para todos los contenedores del pod
- Cada contenedor debe montarlo en un directorio



# Kubernetes - k8s

## Volumen > Tipos

- emptyDir: directorio vacío
- gitRepo: directorio vacío + git clone
- hostPath: monta un directorio del host
- ...

# Kubernetes - k8s

## Volumen > Especificación

- `.spec.volumes`, `.spec.volumeMounts`

```
apiVersion: v1
```

```
kind: Pod
```

```
metadata:
```

```
 name: fortune
```

```
spec:
```

```
 volumes:
```

```
 - name: html
 emptyDir: {}
```

El pod declara el volumen

```
 containers:
```

```
 - image: nginx
```

```
 volumeMounts:
```

```
 - name: html
 mountPath: /usr/share/nginx/html
 readOnly: true
```

El contenedor monta el volumen

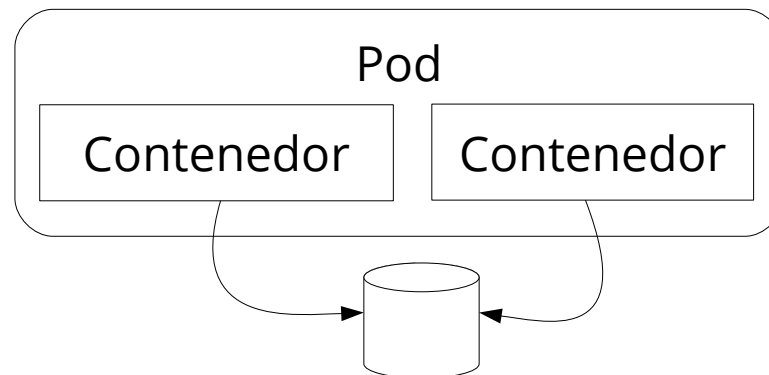
```
 ports:
```

```
 - containerPort: 80
 protocol: TCP
```

# Kubernetes - k8s

## Volumen persistente

- Espacio de almacenamiento independiente del ciclo de vida del pod
- El administrador crea un volumen persistente de manera externa (Google, AWS, NFS, etc.)
- El pod declara el volumen y los contenedores lo montan





# Kubernetes - k8s

## Volumen persistente > Tipos

- awsElasticBlockStore: Amazon AWS
- azureDisk: Azure
- gcePersistentDisk : Google
- nfs: NFS
- ...

# Kubernetes - k8s

## Volumen persistente > Especificación

- Cada tipo utiliza unas propiedades

```
apiVersion: v1
kind: Pod
metadata:
 name: fortune
spec:
```

```
 volumes:
```

```
 - name: mongodb-data
 awsElasticBlockStore:
 volumeId: my-volume
 fsType: ext4
```

Volumen creado en AWS

```
 containers:
```

```
 - image: mongodb
 volumeMounts:
 - name: mongodb-data
 mountPath: /data/db
```

```
 ports: [{containerPort: 27017, protocol: TCP}]
```

# Kubernetes - k8s

## Ejemplo (nfs)

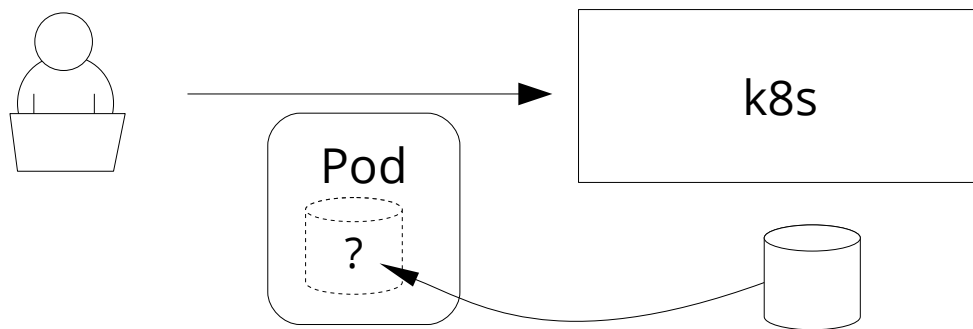
```
apiVersion: v1
kind: Pod
metadata:
 name: ejemplo
spec:
 volumes:
 - name: nfs-vol
 nfs:
 server: my-nfs-server.com
 path: /somepath
 containers:
 - name: nginx
 image: nginx
 volumeMounts:
 - name: nfs-vol
 mountPath: /usr/share/nginx/html
```

Directorio exportado por NFS

# Kubernetes - k8s

## PersistentVolume(Claim)

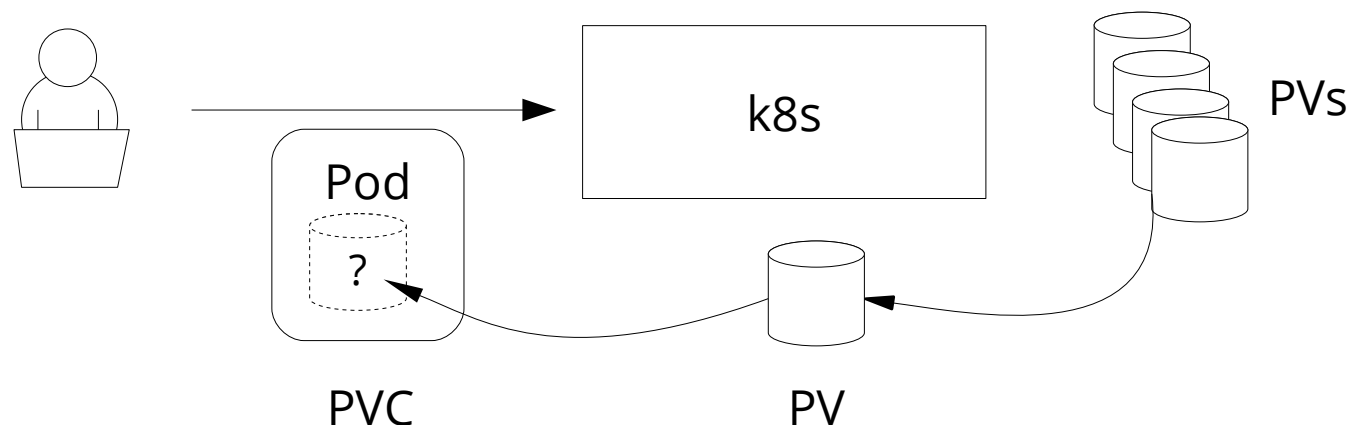
- Con volúmenes, el usuario debe conocer los detalles de implementación (tipo, servidor, etc.)
- Una mejor opción sería que el pod solicite cierto espacio de almacenamiento y el sistema lo asigne automáticamente



# Kubernetes - k8s

## PersistentVolume(Claim)

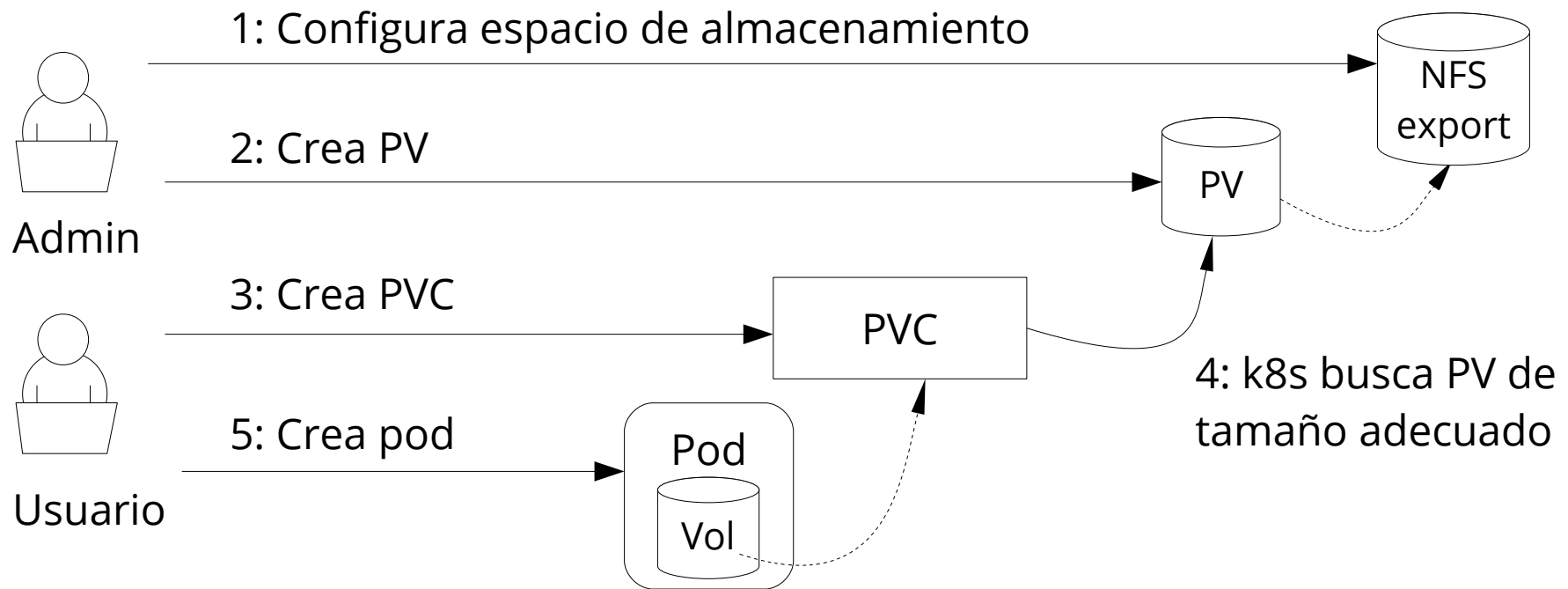
- El pod únicamente describe sus requisitos de almacenamiento en **PersistentVolumeClaim**
- El sistema dispone de varios PersistentVolumes disponibles; asigna uno al pod



# Kubernetes - k8s

## PersistentVolume(Claim)

- Esquema



# Kubernetes - k8s

## PersistentVolume > Especificación

- .capacity, .volumeMode, .accessModes, ...

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: nfs-vol
spec:
 capacity:
 storage: 10Gi
 volumeMode: Filesystem
 accessModes:
 - ReadWriteMany
 persistentVolumeReclaimPolicy: Recycle
 nfs:
 path: /exported-folder
 server: my-nfs-server
```

The diagram consists of three vertical lines on the right side, each with a horizontal tick mark. From the top tick mark, a line goes left to the 'volumeMode' field and then up to the text 'Filesystem, Block'. From the middle tick mark, a line goes left to the 'accessModes' field and then up to the text '[ReadWrite | ReadOnly][Once | Many]'. From the bottom tick mark, a line goes left to the 'persistentVolumeReclaimPolicy' field and then up to the text 'Recycle, Retain, Delete'.

# Kubernetes - k8s

## PersistentVolumeClaim > Especificación

- .accessModes, .volumeMode, .resources, .selector

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: myclaim
spec:
 accessModes:
 - ReadWriteOnce
 volumeMode: Filesystem
 resources:
 requests:
 storage: 8Gi
 selector:
 matchLabels:
 release: "stable"
```

|  |                      |
|--|----------------------|
|  | Requisitos           |
|  | Etiquetas (opcional) |



# Kubernetes - k8s

## PersistentVolumeClaim > Especificación pod

- .volumes.persistentVolumeClaim

```
apiVersion: v1
kind: Pod
metadata:
 name: ejemplo
spec:
 volumes:
 - name: myvol
 persistentVolumeClaim:
 claimName: myclaim
 containers:
 - name: nginx
 image: nginx
 volumeMounts:
 - name: myvol
 mountPath: /usr/share/nginx/html
```

# Kubernetes - k8s

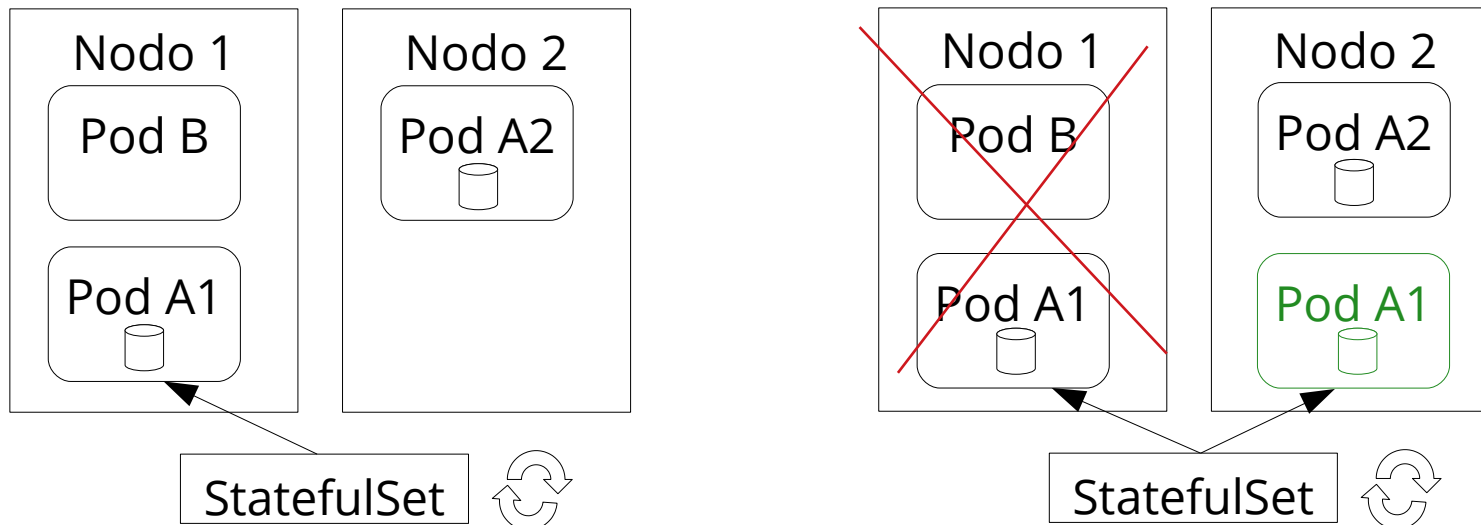
## StatefulSet

- Los pods son efímeros: cada nuevo pod tiene su propio id, IP, vols, etc.
- ReplicaSet crea réplicas idénticas a partir de su plantilla: si una falla, la desecha y crea otra
- A veces necesitamos réplicas con un identificador/estado **estable**: si falla queremos que se recupere (almacenamiento replicado, masters, etc.)
- Pet vs Cattle (stateful vs stateless)

# Kubernetes - k8s

## StatefulSet

- StatefulSet mantiene una colección de **réplicas stateful** funcionando de manera ininterrumpida
- Si una réplica falla se reinicia con la misma configuración (id, IP, vols, etc.)



# Kubernetes - k8s

## StatefulSet

- StatefulSet mantiene una colección de **réplicas stateful** funcionando de manera ininterrumpida
- Si una réplica falla se reinicia con la misma configuración (id, IP, vols, etc.)
- Cada réplica tiene un id ordinal único {0 ... N-1}
- El hostname se construye con <name>-id

# Kubernetes - k8s

## StatefulSet > Especificación

- Similar a ReplicaSet

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
 name: web
spec:
 selector:
 matchLabels:
 app: nginx
 replicas: 3
 template:
```

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

Pod template



# Kubernetes - k8s

## Cuotas

- Un pod puede especificar los recursos mínimos (**requests**) y máximos (**limits**) por contenedor
- El Scheduler planifica dónde ejecutar el pod a partir de esta información y de sus políticas

# Kubernetes - k8s

## Cuotas > Especificación

- `.spec.containers.resources`

```
apiVersion: v1
kind: Pod
metadata:
 name: ejemplo
spec:
 containers:
 - image: busybox
 command: ["dd", "if=/dev/zero", "of=/dev/null"]
 name: main
 resources:
 limits:
 cpu: "1" # 1000m
 memory: "20Mi"
 requests:
 cpu: "0.5"
```

# Kubernetes - k8s

## Escalado automático

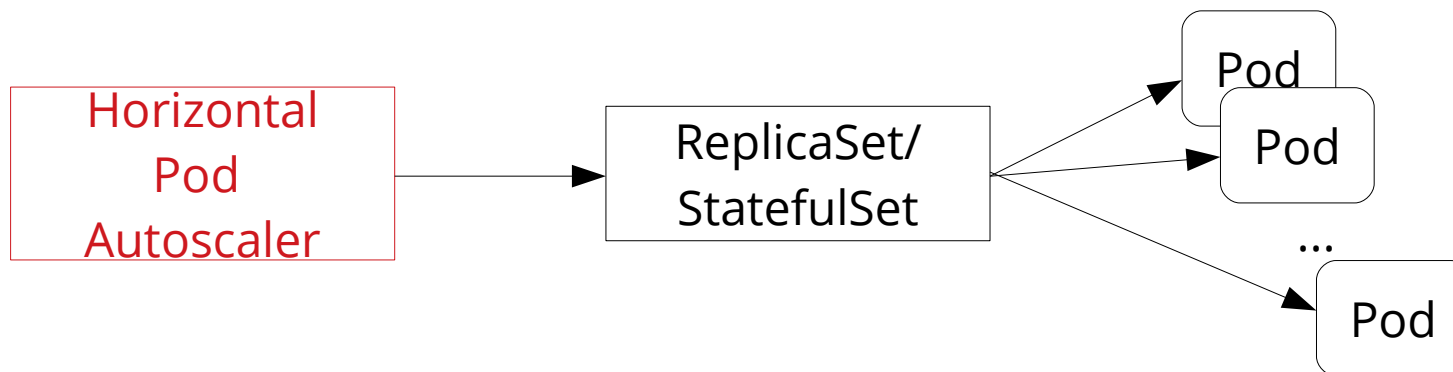
- Con `resources.limits` se efectúa escalado vertical **manual**
- ReplicaSet/StatefulSet efectúan un escalado horizontal **manual** con `spec.replicas`
- No es útil en picos de carga inesperada
- k8s permite efectuar **escalado automático** de nodos y de pods:
  - [HorizontalPodAutoscaler](#)
  - [VerticalPodAutoscaler](#)



# Kubernetes - k8s

## HorizontalPodAutoscaler (HPA)

- Controlador que:
  1. Comprueba periódicamente las métricas de los pods
  2. Calcula el número pods para alcanzar objetivos
  3. Actualiza **.spec.replicas** del ReplicaSet/StatefulSet



# Kubernetes - k8s

## HPA > Especificación

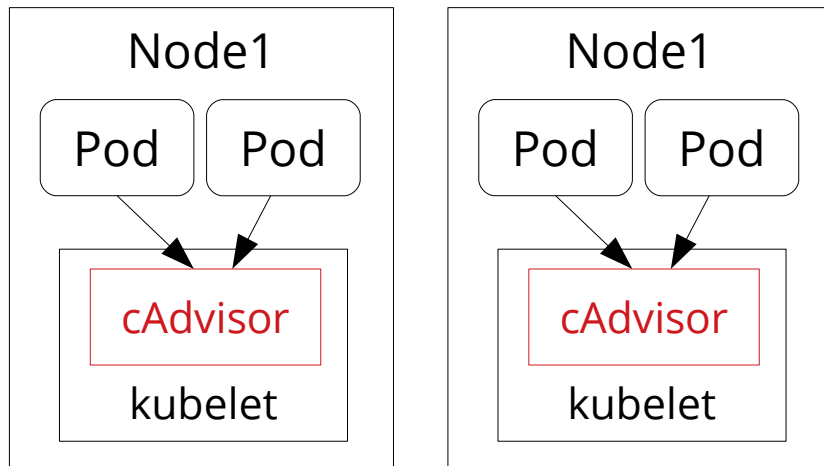
- .maxReplicas,.minReplicas,.metrics,.scaleTargetRef

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
 name: ejemplo-hpa
spec:
 maxReplicas: 5
 minReplicas: 1
 metrics:
 - type: Resource
 resource:
 name: cpu
 target: {type: Utilization, averageUtilization: 60}
 scaleTargetRef:
 apiVersion: apps/v1
 kind: ReplicaSet
 name: ejemplo-rs
```

# Kubernetes - k8s

## HPA > Comprobar las métricas

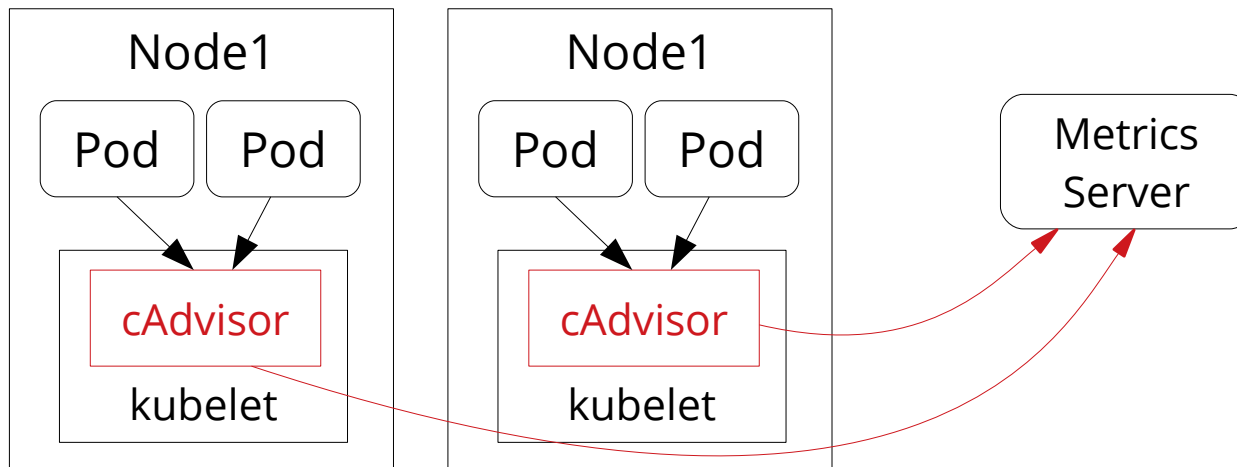
- kubelet incluye proceso **cAdvisor** que obtiene métricas de los pods



# Kubernetes - k8s

## HPA > Comprobar las métricas

- kubelet incluye proceso **cAdvisor** que obtiene métricas de los pods
- Metrics Server agrega todas las métricas



# Kubernetes - k8s

## HPA > Comprobar las métricas

- kubelet incluye proceso **cAdvisor** que obtiene métricas de los pods
- Metrics Server agrega todas las métricas
- Podemos consultarlas con:
  - > `kubectl top node [<name>]`
  - > `kubectl top pod [<name>]`

# Kubernetes - k8s

## HPA > Calcular número de pods

- Calcula promedio de las métricas de todos los pods

$\text{currentMetricValue} = (100+80+30)/3 = 210/3 = 70\%$



# Kubernetes - k8s

## HPA > Calcular número de pods

- Calcula promedio de las métricas de todos los pods

$$\text{currentMetricValue} = (100 + 80 + 30) / 3 = 210 / 3 = 70\%$$

- Calcula el número de réplicas deseadas

$$\text{desiredReplicas} = \text{ceil}[\text{currentReplicas} * (\text{currentMetricValue} / \text{desiredMetricValue})]$$



# Kubernetes - k8s

## HPA > Calcular número de pods

- Calcula promedio de las métricas de todos los pods

$\text{currentMetricValue} = (100 + 80 + 30) / 3 = 210 / 3 = 70\%$

- Calcula el número de réplicas deseadas

$\text{desiredReplicas} = \text{ceil}[3 * (70 / 60)] = 4$

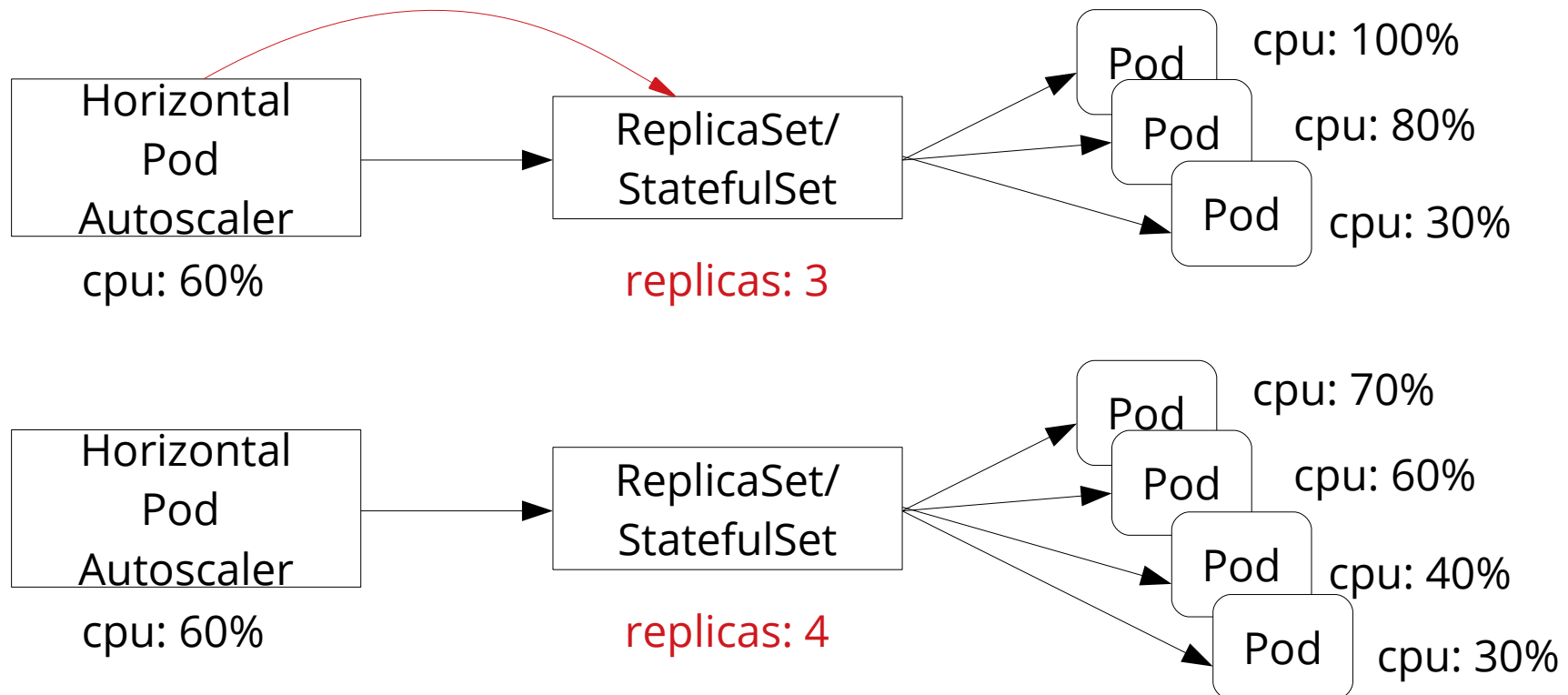




# Kubernetes - k8s

## HPA > Ajustar número de replicas

- Modifica el valor `.spec.replicas` del recurso controlado



# Kubernetes - k8s

## Mucho más ...

- Existen otros muchos recursos: ConfigMaps, Secrets, Deployments, etc.